Expensive Optimization with Model-Based Evolutionary Algorithms Applied to Medical Image Segmentation Using Deep Learning

Dushatskiy, A.

DOI
10.4233/uuid:7eacc7fc-523e-4172-9c62-ab28916398ef

Publication date
2023

Document Version
Final published version

Citation (APA)
Dushatskiy, A. (2023). *Expensive Optimization with Model-Based Evolutionary Algorithms Applied to Medical Image Segmentation Using Deep Learning*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:7eacc7fc-523e-4172-9c62-ab28916398ef

Expensive Optimization with Model-Based Evolutionary Algorithms Applied to Medical Image Segmentation Using Deep Learning

Expensive Optimization with Model-Based Evolutionary Algorithms Applied to Medical Image Segmentation Using Deep Learning

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Wednesday September 6 2023 at 17:30 o'clock

by

Arkadiy Dushatskiy

Master of Science in Applied Mathematics and Computer Science,
Lomonosov Moscow State University, Russia,
born in Stupino, Moscow Region, Russia.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. P.A.N. Bosman,	Centrum Wiskunde & Informatica, Delft University of Technology, promotor
Dr. T. Alderliesten,	Leiden University Medical Center, Delft University of Technology, copromotor
Independent members:	
Prof. dr. R. Miikkulainen,	University of Texas at Austin, USA
Prof. dr. I. Išgum,	Amsterdam University Medical Centers, University of Amsterdam
Dr. N. van Stein,	Leiden University
Prof. dr. ir. K.I. Aardal,	Delft University of Technology
Prof. dr. P.S. César Garcia,	Centrum Wiskunde & Informatica, Delft University of Technology, reserve member

This thesis was part of the research programme Commit2Data (project number 628.011.012), financed by the Netherlands Organisation for Scientific Research (NWO), and was carried out in collaboration with the Netherlands eScience Center; the Amsterdam University Medical Centers (UMC), location Academic Medical Center (AMC); Leiden University Medical Center (LUMC), and Mount Vernon Cancer Center.

SIKS Dissertation Series No. 2023-20.
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Keywords: evolutionary algorithms, expensive optimization, deep learning, medical image segmentation, neural architecture search
Printed by: Ipskamp Printing
Front & Back: The cover is generated by Midjourney (midjourney.com) using prompts with the keywords "DNA double helix" and "neurons".

ISBN 978-94-6473-182-8

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Computer science people spend a lot of their time talking about whether or not man is merely a machine, whether his brain is just a powerful computer that might one day be copied.

Richard P. Feynman

CONTENTS

SUMMARY

Recently great achievements have been obtained with Artificial Intelligence (AI) methods including human-level performance in such challenging areas as image processing, natural language processing, computational biology, and game playing. Arguably, one of the most societally important application fields of such methods is healthcare.

AI is a broad term, which in general refers to systems and methods (components of systems), capable of solving complex tasks and ultimately doing it autonomously, i.e., without human participation, or, if necessary (e.g., in healthcare) with some human supervision. Machine Learning (ML) is a subfield of AI that consists of diverse methods which utilize available data to extract meaningful and actionable knowledge. Three key factors have contributed to the recent success of ML methods: 1) Novel algorithms; 2) Highly efficient hardware, the computational capabilities of which are perfectly aligned with the currently most popular component of AI systems - deep neural networks (a computational abstraction that vaguely resembles a brain and can be efficient in solving different ML problems); 3) Huge amounts of digitally available data which can be used to train ML models. In this thesis, we mainly focus on the combination of algorithm development and data-related aspects.

Optimization and ML are strongly connected, simply because an essential part of the development of an ML model (its training) entails solving an optimization problem. The training procedure depends a lot on the type of ML model and the optimization algorithm used, but in general, it means adjusting learnable model parameters to fit the data as well as possible. Deep Learning (DL) is a subfield of ML that uses neural networks to extract features and learn dependencies from data. Gradient descent-based algorithms enabled a breakthrough in ML by making DL (using large neural networks with millions and even billions of learnable parameters) possible through effective training (optimization) of neural networks of such size. However, not all optimization problems have gradient information available. In such cases, derivative-free methods can be used. Of these, Evolutionary Algorithms (EAs) are of particular interest.

EAs are an AI method, particularly, a special type of optimization algorithm, forming the subfield of meta-heuristics inspired by the natural evolution process. In EAs, instead of gradually improving one solution (as, for instance, in gradient descent), a collection of solutions called a population is evolved simultaneously. The main working principle of all EAs mimics the natural evolution process: survival of the fittest individuals (with respect to the optimization objective, also called the fitness function) and combining good individuals in order to obtain offspring that are hopefully even better. In this thesis, we focus on so-called model-based EAs. In such EAs, the generation process of new solutions is governed by a configurable model that is aligned as well as possible with specific properties associated with the problem at hand. Such alignment may be done manually, or automatically, during optimization, leveraging previously evaluated solutions. This feature makes model-based EAs the most efficient type of EAs currently

existing for many types of problems. In particular, in this thesis, we focus on algorithms from the Optimal Mixing Evolutionary Algorithms family (OMEAs) which showed state-of-the-art performance in different optimization domains such as discrete and real-valued optimization, genetic programming, and multi-objective optimization. Moreover, one of the key research subjects in this thesis is the optimization of computationally expensive functions (expensive optimization). This means that good solutions must be found by using only a few evaluations, as these are computationally expensive. To use EAs for this purpose, we study and design the integration of surrogate modeling techniques: an approach to replace evaluating the true, expensive optimization function with a much cheaper to compute, yet imprecise, approximation. In this thesis, we focus on the discrete expensive optimization domain, for which the literature on the use of EAs is scarce.

While DL has enormous capabilities, there are almost always also many components that need to be configured properly to get the best possible performance on a particular dataset and task. These include data preprocessing, training hyperparameters (stochastic gradient descent configuration, regularization strength, etc.), and, importantly, neural network architecture choice. The number of possible design choices of neural networks has been rapidly growing recently, raising the question for practitioners what is the best architecture for a particular task and dataset. In the Neural Architecture Search (NAS) field, the aim is to automate the search for the best possible neural network architecture. Although various efforts on NAS have been published, a question that did not yet get much attention is how problematic the stochasticity of architecture quality evaluation is and whether this can be alleviated in order to allow for finding better-performing architectures. In this thesis, we study this in more detail and find that for finding better-performing architectures, it is beneficial to spend more computational resources on diminishing the noise of architecture quality estimates.

Just like is the case for natural image processing tasks, DL has been successfully applied to medical image analysis tasks. A common example of such a task is organ segmentation on a Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) scan. Performing such tasks automatically (followed by human approval or correction if necessary) can contribute to making diagnosis and treatment procedures faster and more accurate and reduce the workload of medical professionals. One of the largest challenges in this field is bringing automatic methods into actual clinical use. With regard to the task of image segmentation (also called contouring or delineation), it means that a method needs to automatically produce such segmentations (e.g., of specific organs) which are found acceptable by a clinician to be used for the task at hand. One of the reasons why this is challenging is the phenomenon called observer variation. It is noticed that different clinicians can have (slightly) different ways of performing segmentation. Moreover, even one clinician might end up with a different result when asked to perform the same task again with some time interval between the segmentation sessions. Therefore, it might be concluded that there is no single correct way of performing organ segmentation. Consequently, it stands to reason that an automatic segmentation method that can automatically produce multiple segmentation variants instead of just one can increase the chances of acceptance of one of the presented variants by a clinician. In contrast to the standard use of DL (which produces one segmentation variant), this approach to automatic segmentation is relatively under-explored. In this thesis, we aim at bridging this

gap between developing new segmentation methods and making them more acceptable in clinical practice.

The main contributions of this thesis are as follows:

1. The current state-of-the-art in model-based black-box discrete optimization is improved by presenting a new version of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) (*Chapter 2*). Besides an analysis of the impact of different design choices for GOMEA, a modification of the Gene-pool Optimal Mixing (GOM) variation operator (one of the key components of GOMEA) called CGOM is presented. The main idea behind CGOM is taking into account not only strong dependencies between variables but also weak ones using the notion of conditional dependence. In experiments on a set of common benchmark functions, the new version of GOMEA outperformed previously published ones and other prominent model-based EAs such as P3 and DSMGA-II.

2. We explore an extreme case of surrogate-assisted EA design: aiming at having a perfectly accurate surrogate model (*Chapter 3*). For a specific class of functions (decomposable functions with a linear number of subfunctions and a constant number of variables in them), it is shown that a Walsh decomposition can be used to obtain such a surrogate model with excellent scalability.

   We introduce a more generally applicable surrogate-assisted GOMEA for discrete optimization problems with expensive function evaluations (*Chapter 4*). Furthermore, a variant that is capable of solving discrete optimization problems with arbitrary alphabets (beyond binary problems) is proposed (*Chapter 5*). Noteworthy, our proposed algorithm has an automatic population size management scheme and discards the need for manual population size tuning, a commonly known difficulty in EAs. We compare the proposed algorithm to the most common choice for solving expensive optimization problems - Bayesian optimization algorithms. In our experiments, we use both synthetic benchmark functions and a real-world optimization problem from the ML field, namely, dataset partitioning for maximizing the performance of an ensemble of classifiers. The proposed surrogate-assisted GOMEA outperformed Bayesian optimization alternatives as well as non-surrogate-assisted GOMEA and simple search algorithms such as random and local search.

3. The problem of NAS is studied in regard to understanding whether noisy fitness functions form a significant problem for NAS algorithms (*Chapter 6*). It is shown that the noise originating from the stochastic nature of neural network initialization and the training and validation procedure can indeed significantly hinder the performance of NAS algorithms (in an independent test evaluation). It is proposed to use more robust network evaluation procedures involving cross-validation. Under the same computational budget, it is shown that a more robust evaluation leads to finding better-performing architectures. Experiments were conducted for the task of NAS for medical image segmentation.

4. A novel method for automatic medical image segmentation is introduced. Its main difference from standard DL approaches is that it is able to produce multiple segmentation variants. We base our method on the idea that instead of training one neural network on all available data and therefore getting the best "average" result, we can train multiple (sub)networks on more homogeneous data subsets and get multiple, but different, "specialized" results. To obtain such subsets, we propose to use a combinatorial optimization algorithm. In particular, we use the surrogate-assisted GOMEA algorithm presented earlier in this thesis (*Chapters 4, 5*). Our proposed segmentation method has been first shown to correctly identify data subsets pertaining to artificially introduced variations (*Chapter 7*). Then, we show that in the case of clinical data (CT scans used for brachytherapy treatment without any artificially added variations), our algorithm is as well capable of producing better segmentations than a standard DL approach (*Chapter 8*).

Summing up, in this thesis we improve and propose novel EAs continuing the research line of the OMEA family of algorithms, focusing on, but not limited to expensive combinatorial optimization problems. We also study the existing problem of the impact of optimization function stochasticity in NAS on the performance of optimization algorithms and propose a way to alleviate it. Finally, we demonstrate how efficient EAs can be combined with DL in order to design a novel medical image segmentation method that is potentially more applicable in clinical practice than standard DL methods.

We conclude that GOMEA and its surrogate-assisted version have strong potential in solving combinatorial optimization problems, including the ones where fitness function evaluations are computationally expensive. Combining EAs with DL can result in novel approaches, some of which can provide added value in such societally important domains as healthcare, in particular, medical image analysis.

SAMENVATTING

Onlangs zijn er grote prestaties behaald met methoden voor Kunstmatige Intelligentie (KI), waaronder prestaties op menselijk niveau op uitdagende gebieden zoals beeldverwerking, natuurlijke taalverwerking, computationele biologie en het spelen van games. Naar alle waarschijnlijkheid is één van de belangrijkste toepassingsgebieden van dergelijke methoden voor de samenleving de gezondheidszorg.

KI is een brede term die over het algemeen verwijst naar systemen en methoden (onderdelen van systemen) die in staat zijn om complexe taken op te lossen en uiteindelijk autonoom te handelen, dat wil zeggen zonder menselijke tussenkomst, of indien nodig (bijv. in de gezondheidszorg) met enige menselijke supervisie. Machine Learning (ML) is een subgebied van KI dat bestaat uit diverse methoden die gebruikmaken van beschikbare data om betekenisvolle en bruikbare kennis te extraheren. Drie belangrijke factoren hebben bijgedragen aan het recente succes van ML-methoden: 1) Nieuwe algoritmen; 2) Zeer efficiënte hardware, waarvan de rekenkracht perfect aansluit bij de momenteel meest populaire component van KI-systemen - diepe neurale netwerken (een computationele abstractie die vaag lijkt op een brein en efficiënt kan zijn bij het oplossen van verschillende ML-problemen); 3) Enorme hoeveelheden digitaal beschikbare data die kunnen worden gebruikt om ML-modellen te trainen. In dit proefschrift richten we ons voornamelijk op de combinatie van de ontwikkeling van algoritmen en aspecten die verband houden met data.

Optimalisatie en ML zijn sterk met elkaar verbonden, simpelweg omdat een essentieel onderdeel van de ontwikkeling van een ML-model (het trainen ervan) het oplossen van een optimalisatieprobleem omvat. Het trainingsproces is sterk afhankelijk van het type ML-model en het gebruikte optimalisatie-algoritme, maar het houdt over het algemeen in dat leer- of trainbare modelparameters worden aangepast om zo goed mogelijk bij de data te passen. Deep Learning (DL) is een subgebied van ML dat gebruikmaakt van neurale netwerken om kenmerken te extraheren en afhankelijkheden te leren uit data. Algoritmen op basis van gradient descent hebben een doorbraak mogelijk gemaakt in ML door DL (het gebruik van grote neurale netwerken met miljoenen en zelfs miljarden trainbare parameters) mogelijk te maken door effectieve training (optimalisatie) van neurale netwerken van dergelijke omvang. Echter is gradiëntinformatie niet voor alle optimalisatie problemen beschikbaar. In dergelijke gevallen kunnen methoden zonder afgeleiden worden gebruikt. Van deze methoden zijn Evolutionaire Algoritmen (EAs) van groot belang.

EAs zijn een KI-methode, in het bijzonder een speciaal type optimalisatie-algoritme, die het subveld van metaheuristieken vormt die geïnspireerd zijn door het natuurlijke evolutieproces. Een EA evolueert een verzameling oplossingen, die een populatie wordt genoemd, gelijktijdig in plaats van geleidelijk één oplossing te verbeteren (zoals bijvoorbeeld bij gradient descent). Het belangrijkste werkingsprincipe van alle EAs bootst het natuurlijke evolutieproces na: overleving van de fitste individuen (met betrekking tot

het optimalisatiedoel, ook wel de fitnessfunctie genoemd) en het combineren van goede individuen om hopelijk nog betere nakomelingen te verkrijgen. In dit proefschrift richten we ons op zogenaamde modelgebaseerde EAs. In dergelijke EAs wordt het generatieproces van nieuwe oplossingen gestuurd door een configureerbaar model dat zo goed mogelijk is afgestemd op specifieke eigenschappen van het probleem waarvoor geoptimaliseerd wordt. Deze afstemming kan handmatig worden gedaan of automatisch tijdens de optimalisatie, waarbij in dit laatste geval eerder geëvalueerde oplossingen worden gebruikt. Deze eigenschap maakt modelgebaseerde EAs de meest efficiënte vorm van EAs die momenteel bestaat voor veel soorten problemen. In het bijzonder richten we ons in dit proefschrift op algoritmen uit de familie genaamd Optimal Mixing Evolutionary Algorithms (OMEAs). Binnen verschillende optimalisatie domeinen, waaronder discrete en continue optimalisatie, alsook genetische programmering en problemen waarbij er meerdere doelen tegelijkertijd geoptimaliseerd moeten worden, zijn met algoritmen van het OMEA type state-of-the-art resultaten behaald. Bovendien is één van de belangrijkste onderwerpen van onderzoek in dit proefschrift de optimalisatie van rekenkundig dure functies (dure optimalisatie). Dit betekent dat goede oplossingen moeten worden gevonden met slechts een paar evaluaties, aangezien deze rekenkundig duur zijn. Om EAs hiervoor te gebruiken, bestuderen en ontwerpen we de integratie van technieken voor surrogaatmodellering: een aanpak om de werkelijke, dure optimalisatiefunctie te vervangen door een veel goedkopere, maar minder nauwkeurige benadering. In dit proefschrift richten we ons op het domein van discrete dure optimalisatie, waarbinnen er weinig literatuur is over het gebruik van EAs.

Hoewel DL enorme mogelijkheden biedt, moeten er bijna altijd ook veel componenten goed geconfigureerd worden om de best mogelijke prestaties te behalen op een specifieke dataset en taak. Deze omvatten het voorbewerken van de data, het trainen van hyperparameters (configuratie van stochastic gradient descent, regularization strength, enz.) en in het bijzonder de keuze van de architectuur van het neurale netwerk. Het aantal mogelijke ontwerpkeuzes van neurale netwerken is de laatste tijd snel toegenomen, wat voor vakmensen de vraag oproept: wat is de beste architectuur voor een specifieke taak en dataset? In het veld van Neural Architecture Search (NAS) is het doel om de zoektocht naar de best mogelijke architectuur van het neurale netwerk te automatiseren. Hoewel er verschillende inspanningen op het gebied van NAS zijn gepubliceerd, is er nog niet veel aandacht besteed aan de problematiek van de stochasticiteit van de evaluatie van de architectuurkwaliteit en of dit kan worden verminderd om beter presterende architecturen te vinden. In dit proefschrift bestuderen we dit in meer detail en vinden we dat het gunstig is om meer rekenkracht te besteden aan het verminderen van de ruis van de architectuurkwaliteitsschattingen om beter presterende architecturen te vinden.

Vergelijkbaar met het succes van het toepassen van DL op natuurlijke beeldverwerkingstaken, is DL ook met succes toegepast op taken binnen de medische beeldanalyse. Een veelvoorkomend voorbeeld van zo'n taak is orgaansegmentatie op een Magnetic Resonance Imaging (MRI) of Computed Tomography (CT) scan. Het automatisch uitvoeren van dergelijke taken (gevolgd door goedkeuring door mensen of correctie indien nodig) kan bijdragen aan het versnellen en nauwkeuriger maken van diagnose- en behandelingsprocedures en de werklast van medische professionals verminderen. Één van de grootste uitdagingen in dit vakgebied is om automatische methoden daadwerkelijk

in de klinische praktijk te gebruiken. Met betrekking tot de taak van beeldsegmentatie (ook wel contouring of delineatie genoemd) betekent dit dat een methode automatisch dergelijke segmentaties moet produceren (bijvoorbeeld van specifieke organen) die door een clinicus als acceptabel worden beschouwd voor de betreffende taak. Één van de redenen waarom dit uitdagend is, is het fenomeen dat bekend staat als variatie onder waarnemers. Het is bekend dat verschillende clinici (lichtelijk) verschillende manieren kunnen hebben om segmentatie uit te voeren. Bovendien kan zelfs één clinicus een ander resultaat krijgen wanneer hem/haar wordt gevraagd dezelfde taak opnieuw uit te voeren met een tijdsinterval tussen de segmentatiesessies. Daarom kan geconcludeerd worden dat er geen enkele juiste manier is om orgaansegmentatie uit te voeren. Het staat dan ook redelijkerwijs vast dat een automatische segmentatiemethode die automatisch meerdere segmentatievarianten kan produceren in plaats van slechts één, de kans op acceptatie van één van de gepresenteerde varianten door een clinicus kan vergroten. In tegenstelling tot het standaard gebruik van DL (dat één segmentatievariant produceert), is deze benadering van automatische segmentatie relatief onderbelicht. In dit proefschrift streven we ernaar om deze kloof te overbruggen tussen het ontwikkelen van nieuwe segmentatiemethoden en het acceptabel maken ervan in de klinische praktijk.

De belangrijkste bijdragen van dit proefschrift zijn als volgt:

1. Huidige toonaangevende modelgebaseerde black-box discrete optimalisatie wordt verbeterd door een nieuwe versie van het Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) te presenteren (*Hoofdstuk 2*). Naast een analyse van de impact van verschillende ontwerpkeuzes voor GOMEA, wordt er een wijziging van de Gene-pool Optimal Mixing (GOM) variatie-operator (één van de belangrijkste componenten van GOMEA) gepresenteerd, genaamd CGOM. Het voornaamste idee achter CGOM is om rekening te houden met zowel sterke afhankelijkheden tussen variabelen als zwakke afhankelijkheden met behulp van het concept van conditionele afhankelijkheid. In experimenten op een set van veelvoorkomende benchmarkfuncties presteerde de nieuwe versie van GOMEA beter dan eerder gepubliceerde versies en andere prominente modelgebaseerde EAs zoals P3 en DSMGA-II.

2. We onderzoeken een extreme situatie voor de integratie van technieken voor surrogaatmodellering met het EA, namelijk het creëren van een perfect nauwkeurig surrogaatmodel (*Hoofdstuk 3*). Voor een specifieke klasse van functies (ontbindbare functies bestaande uit een lineair aantal subfuncties en een constant aantal variabelen), wordt aangetoond dat een Walsh-decompositie kan worden gebruikt om zo'n surrogaatmodel te verkrijgen met uitstekende schaalbaarheid.

   We introduceren een meer algemeen toepasbare variant van GOMEA waarbij gebruik gemaakt wordt van een surrogaat model voor discrete optimalisatieproblemen met dure functie-evaluaties (*Hoofdstuk 4*). Bovendien wordt er een variant voorgesteld die in staat is om discrete optimalisatieproblemen met willekeurige alfabetten op te lossen (buiten binaire problemen) (*Hoofdstuk 5*). Bijzonder hierbij is dat ons voorgestelde algoritme automatisch de populatiegrootte beheert waardoor de handmatige afstemming van de populatiegrootte niet langer nodig is, wat een veelvoorkomend probleem bij EAs is. We vergelijken het voorgestelde algoritme

met de meest gebruikelijke keuze voor het oplossen van dure optimalisatieproblemen - Bayesiaanse optimalisatiealgoritmen. In onze experimenten gebruiken we zowel synthetische benchmarkfuncties als een optimalisatieprobleem in de praktijk uit het ML-veld, namelijk het partitioneren van datasets om de prestaties van een ensemble van classifiers te maximaliseren. De voorgestelde GOMEA, dat gebruikmaakt van een surrogaat model, presteerde beter dan Bayesiaanse optimalisatiealgoritmen, evenals klassieke GOMEA en eenvoudige zoekalgoritmen zoals random en local search.

3. Het probleem van NAS wordt bestudeerd om te begrijpen of fitnessfuncties met ruis een significante uitdaging vormen voor NAS-algoritmen (*Hoofdstuk 6*). Er wordt aangetoond dat de ruis die voortkomt uit de stochastische aard van de initialisatie van het neurale netwerk en het trainings- en validatieproces inderdaad de prestaties van NAS-algoritmen aanzienlijk kan belemmeren (in een onafhankelijke testevaluatie). Er wordt voorgesteld om robuustere netwerkevaluatieprocedures te gebruiken, zoals cross-validation. Bij gebruik van hetzelfde computationele budget wordt aangetoond dat een robuustere evaluatie leidt tot het vinden van beter presterende architecturen. De experimenten werden uitgevoerd op NAS voor medische beeldsegmentatie.

4. Een nieuwe methode voor automatische medische beeldsegmentatie wordt geïntroduceerd. Het belangrijkste verschil met standaard DL-aanpakken is dat het in staat is om meerdere segmentatievarianten te produceren. Onze methode is gebaseerd op het idee dat we in plaats van één neuraal netwerk te trainen op alle beschikbare data en daardoor het beste "gemiddelde" resultaat te krijgen, meerdere (sub)netwerken kunnen trainen op meer homogene subsets van data en meerdere, maar verschillende, "gespecialiseerde" resultaten kunnen verkrijgen. Om dergelijke subsets te verkrijgen, stellen we voor om een combinatorisch optimalisatiealgoritme te gebruiken. In het bijzonder maken we gebruik van het GOMEA algoritme met surrogaatmodellering dat eerder in dit proefschrift is gepresenteerd (*Hoofdstukken 4, 5*). Eerst wordt aangetoond dat onze voorgestelde segmentatiemethode op correcte wijze datasubsets kan identificeren die betrekking hebben op kunstmatig geïntroduceerde variaties (*Hoofdstuk 7*). Vervolgens laten we zien dat in het geval van klinische data (CT-scans die ingetekend waren door artsen ten behoeve van het opstellen van brachytherapie behandelplannen) ons algoritme in staat is om betere segmentaties te produceren dan een standaard DL-aanpak.(*Hoofdstuk 8*).

In dit proefschrift verbeteren we bestaande EAs en stellen we nieuwe EAs voor, voortbouwend op het onderzoek van de OMEA-familie van algoritmen. We richten ons op, maar beperken ons niet tot, kostbare combinatorische optimalisatieproblemen. We onderzoeken ook het bestaande probleem van de impact van de stochasticiteit van de optimalisatiefunctie in NAS op de prestaties van optimalisatiealgoritmen en stellen een manier voor om dit te verlichten. Tot slot laten we zien hoe efficiënte EAs gecombineerd kunnen worden met DL om een nieuwe methode voor medische beeldsegmentatie

te ontwerpen die mogelijk beter toepasbaar is in de klinische praktijk dan standaard DL-methoden.

We concluderen dat GOMEA en de variant die gebruikmaakt van een surrogaat model een sterk potentieel hebben om combinatorische optimalisatieproblemen op te lossen, inclusief de problemen waarbij de evaluatie van de fitnessfunctie rekenkundig duur is. Het combineren van EAs met DL kan leiden tot nieuwe aanpakken, waarvan sommige toegevoegde waarde kunnen bieden in maatschappelijk belangrijke domeinen zoals de gezondheidszorg, met name in de medische beeldanalyse.

1

INTRODUCTION

1.1. EFFICIENT COMBINATORIAL OPTIMIZATION ALGORITHMS

Optimization problems arise in various aspects of human life. From classical problems like the Travelling Salesman problem to recently emerged ones like tuning hyperparameters of deep learning models, all of them require efficient and effective optimization algorithms in order to solve them and have practical application value as well. Optimization problems vary in many aspects including search domain (real-valued, discrete, permutations, etc.), problem size (number of variables), and computational costs to compute the objective (fitness) function value. Therefore, optimization algorithms also vary, as it follows from the No Free Lunch Theorem [1] that no algorithm can be the most efficient at finding the optimal solution for all problems in the world. However, if we limit the scope of the problems we are interested in, then it is potentially possible to design an algorithm that outperforms available alternatives. This has resulted in the emergence of various algorithms tailored to different problem types. This thesis focuses on combinatorial optimization problems, explicitly also including problems with computationally expensive fitness evaluations. In other words, problems for which good, or ideally, optimal solutions should be found within a budget of a few objective function evaluations (typically a few hundred or thousand at best). In this thesis, novel Evolutionary Algorithms (a nature-inspired class of heuristic-based optimization algorithms) are proposed for this class of problems. The best performance on an optimization problem (i.e., finding better solutions within a given computational budget) with specific characteristics is usually achieved with an algorithm tailored to it. However, a highly-tailored algorithm may not generalize to other problems. In contrast, some of the ideas about designing Evolutionary Algorithms proposed in this thesis are more generic, aiming to generalize to other problem domains, while also including a basis for problem-specific tailoring.

1.1.1. COMBINATORIAL OPTIMIZATION

Combinatorial optimization problems form one of the most well-known and important classes of optimization problems. Famous examples of combinatorial optimization prob-

lems are Maximum cut (Max-Cut), and Maximum satisfiability (MAXSAT). The Max-Cut problem formulated as an optimization problem entails finding a partitioning of graph nodes into two sets such that the sum of weights of edges between pairs of nodes in opposite sets is maximized. The MAXSAT optimization problem is finding an assignment of Boolean values to variables such that the number of true clauses in a logic formula is maximized. Many well-known combinatorial optimization problems are NP-hard and, therefore, it is reasonable to use search heuristics to find a good solution (i.e., non-optimal, but high quality) within a reasonable amount of time as the optimal solution for an arbitrary problem instance cannot be expected to be found in polynomial time (unless $P = NP$). Such search problems as hyperparameter tuning (searching for the best combination of hyperparameters of an ML model in order to maximize its quality) and Neural Architecture Search (searching for the architecture of a neural network that performs the best on a given dataset), which have been recently attracting a lot of attention, can also often be formulated as combinatorial optimization problems.

Without loss of generality, an unconstrained (all solutions in the defined search space are considered feasible) global, single-objective, combinatorial optimization problem can be defined as follows:

$$\max_{x \in \mathscr{D}} f(x),$$

where

$f(x) : \mathscr{D} \to \mathbb{R}$ is the objective function,
$\mathscr{D}$ is the *search space*, $\mathscr{D} = \{(x_1, x_2, \ldots, x_\ell) \mid x_i \in \{1, 2, \ldots, \alpha_i\}\}$,
ℓ is the number of variables, and α_i is the alphabet size of variable i for $i \geq 1$.

In this thesis, we will focus on a particular type of optimization algorithms, evolutionary algorithms, where the objective function is often referred to as the fitness function. Moreover, we focus on the *black-box* formulation of the combinatorial optimization problem, which means that the function is not known beforehand. This includes any dependence relationships between variables (the case in which they are known is usually called *gray-box* optimization in the literature). Often in literature, solving a combinatorial optimization problem means finding its global optimum: a solution x^* that $f(x^*) \geq f(x)\ \forall x \in \mathscr{D}$. The number of fitness evaluations required to find the global optimum is a common performance metric of optimization algorithms. If the global optimum is not known (this might often be the case for real-world problems), then the best found solution up to a particular number of fitness evaluations might be alternatively used to measure algorithm performance. Generally, optimization algorithms can be classified into the following categories:

1. *Exact* optimization algorithms. The goal of this type of algorithms is to find an optimal solution to the optimization problem with a theoretical guarantee of the computational complexity.
2. *Approximation* algorithms. Such algorithms lack the ability to find the optimal solution but have theoretical guarantees of how close to the optimal solution the found solutions are.

3. *Heuristics.* Algorithms of this type can be used to find non-optimal, but good enough solutions when using approximation algorithms is computationally intractable or hard to design. However, usually, heuristics do not have theoretical guarantees in contrast to approximation algorithms.
4. *Metaheuristics.* This class of algorithms includes various types of algorithms, and they differ from heuristics by usually being more complex and general, i.e., potentially applicable to a wide range of problems instead of being designed to solve a specific problem.

1.1.2. FROM SIMPLE GENETIC ALGORITHMS TO STATE-OF-THE-ART MODEL-BASED EVOLUTIONARY ALGORITHMS

Evolutionary Algorithms (EAs) have been shown to be a powerful metaheuristic for solving optimization problems (e.g., [2, 3]). During the search, a *population* of individuals (solutions) is maintained and (gradually) evolved with the goal of improving their quality (fitness). EAs resemble the natural Darwinian evolution process by implementing two key elements of it: *selection* (the fittest individuals survive) and *variation* (producing offspring by combining genomes of the parents). A subclass of EAs which was one of the first to become popular for solving optimization problems is called Genetic Algorithms (GAs). In classical GAs, variation consists of *recombination* (*crossover*): fit individuals are combined to produce an offspring, and *mutation*: random changes are made in the offspring genotype. Over decades of research, it became clear that to make a GA efficient, these operators, especially the variation, should be aligned with the optimization problem structure. In one of the fundamental works in the field [4] it was hypothesized that GAs work by preserving and propagating through the population high-performing *building blocks*: *schemata* (i.e., an expression which represents multiple solutions by fixing the values of some of the variables while leaving others unspecified). Such building blocks contribute to the above-average fitness value of the defined solutions.

Example 1.1.1. Consider the maximization of a function $f(x)$ of four binary variables, $x = (x_1, x_2, x_3, x_4)$ such that $f(x) = x_1x_2 + x_3x_4$. The solutions that adhere to the following expression $(x_1 = 1, x_2 = 1, x_3 = *, x_4 = *)$, with arbitrary values of x_3 and x_4, will then have an above-average fitness and thus the expression represents a building block. Similarly, the expression $(x_1 = *, x_2 = *, x_3 = 1, x_4 = 1)$ is also a building block.

This hypothesis, also called Holland's Schema Theorem, created a foundation for so-called *model-based* EAs. In [5] it was shown that recombination without disruption of the building blocks is crucial for good scalability of GAs. The main idea of model-based EAs is to use information about building blocks during the variation in order to find better solutions more efficiently. If this information is a priori unknown, it can be learned during evolution. To identify building blocks, information about interactions between variables (also called *linkage*) is required. For example, it can be concluded that two variables x_1 and x_2 are dependent (linked) if the effect of x_1 on the solution fitness depends on the value of x_2, and vice versa. In such a case, it is important to take their linkage into account during evolution, i.e., preserve a building block $\{x_1^*, x_2^*\}$ with high fitness contribution (i.e., an instance of variables x_1 and x_2 that is good for the solution in general) as soon as such an instance is found. Disrupting a good, high-performing

building block is detrimental to the search performance, as the interaction between variables of such building block ensures the high fitness of solutions in which it is present. In the black-box optimization scenario, it is not known a priori how variables interact with each other and whether a fitness function can be decomposed into independent subfunctions. In Example 1.1.1 the knowledge that the function $f(x)$ consists of two independent subfunctions $f_1(x_1, x_2) = x_1 x_2$ and $f_2(x_3, x_4) = x_3 x_4$ is useful to find the correct building blocks. However, this information can be extracted (learned) from the evolving population. Though in most cases the learned problem structure is only an approximation of the true problem structure, it was shown that it can be very useful when used in variation operators for making the optimization process [6, 7, 8] more efficient.

One type of model-based EAs is Estimation-of-Distribution Algorithms (EDAs). Algorithms of this type use a probability distribution over the problem variables which is maintained during search. The distribution is estimated, typically once each generation, from a selection of the fittest individuals. Rather than doing crossover and/or mutation as in classical GAs, EDAs generate new offspring solutions by sampling from this probability distribution. During evolution, the distribution adapts such that better solutions (with better fitness) become more likely to sample. One of the most prominent EDAs for combinatorial problems is the Bayesian Optimization Algorithm (BOA) [6] and its follow-up development that is better suited for solving problems with hierarchical dependencies between variables, called hierarchical BOA [9] (hBOA). BOA and hBOA work by learning a Bayesian network (i.e., the concept of conditional dependencies between variables is used), and sampling new solutions from it. They outperformed classical GAs by orders of magnitude in terms of population size and the number of fitness evaluations required to find the global optimum of several standard benchmark functions [6, 9, 10].

Despite the strong search performance potential, BOA and hBOA have a large computational overhead due to the learning of the Bayesian network. To overcome this issue, a conceptually similar yet rather different idea was later proposed with the family of algorithms called Optimal Mixing Evolutionary Algorithm(s) (OMEA) and its subfamily Gene-pool Optimal Mixing Evolutionary Algorithms (GOMEAs). The OMEA family of algorithms started with the introduction of the Linkage Tree Genetic Algorithm (LTGA) in 2010 [7]. In GOMEA the most important ideas of LTGA are kept and further built upon.

One key feature contributing to the success of the GOMEA family of algorithms is the so-called linkage learning: the extraction of information about dependencies between variables during the evolution process. This information is stored in a *linkage model*. The linkage model in the OMEA family of algorithms is represented as the *Family of Subsets (FOS)*. The main idea of FOS is to store sets of variables that are (presumably) dependent on each other. Assuming that a problem has variables $x_1, x_2, \ldots, x_\ell$ (ℓ is the number of variables), and that the variable indices form a set $S = \{1, 2, \ldots, \ell\}$, an FOS is a set of subsets of S. In principle, different FOS variants can be used, the most simple of which is the univariate FOS: $\{\{1\}, \{2\}, \ldots \{\ell\}\}$. It implies that all variables are independent. The most commonly used FOS in the GOMEA family of algorithms and well-performing on a variety of problems [11] is called *the Linkage Tree (LT)*. An LT is a binary tree with $2\ell - 1$ nodes. LT leaves are singletons of problem variables. The root of an LT is the set of all problem variables, and all other nodes are subsets. Moreover, each node except the leaf nodes is a union of disjoint subsets of the children of that node. The first step of

linkage learning is measuring pairwise dependencies between variables, which, in the discrete case, can be done, for instance, using Mutual Information measured from the population. Mutual Information can capture dependencies between variables as after a population is evolved for several generations, more building blocks are expected to be present in it since selection is expected to make them more prominently featured, and recombination is meant to preserve and mix them. Then, hierarchical clustering is used to merge subsets of dependent variables together.

In contrast to EDAs, in GOMEA the aim is not to explicitly model the probability distribution of variable values corresponding to high fitness, but to use the learned FOS during variation. An elaborate variation operator called Gene-pool Optimal Mixing (GOM) is the second key feature of GOMEA. With GOM, dependencies between variables as stored in the FOS are explicitly exploited, following the principle derived from Holland's Schema Theorem that good building blocks should be preserved and exploited in the population. This is achieved by treating sets of linked variables together in GOM, and ensuring that already found high-performing building blocks are not disrupted. In GOMEA, GOM is applied to each population member. First, the solution is cloned. Next, in each iteration of GOM, all FOS elements are considered. For each FOS element $\{i_1, i_2, \ldots, i_K\}$, a random donor is selected from the current population. Its corresponding genes $\{x_{i_1}, x_{i_2}, \ldots, x_{i_K}\}$ are copied to the current solution. During variation, it is ensured that the changes made to a solution do not lead to fitness deterioration (otherwise, a change is simply reverted). Such a greedy approach also works as a replacement for the selection step in traditional GAs. Finally, the resulting solution is added to a pool of offspring and after each population member has undergone GOM, the offspring replace the population.

The GOMEA family of algorithms includes algorithms for different types of optimization problems: real-valued, multi-objective, permutations, etc. It has been demonstrated to show outstanding performance on both standard benchmarks and, notably, real-world problems. One such example is the usage of Real-Valued Multi-Objective GOMEA (RV-MO-GOMEA) [12] for brachytherapy treatment planning optimization that is currently used in clinical practice at the Amsterdam University Medical Centers for the treatment of prostate cancer patients [13]. Another example is GOMEA for Genetic Programming (GP-GOMEA) [14]. GP-GOMEA has been shown to be applicable to an important medical problem, 3D radiation dose reconstruction [15, 16]. Many cases of GOMEA outperforming alternative optimization algorithms and successful applications show the potential of the GOMEA family of algorithms and model-based EAs in general.

1.1.3. PARAMETERLESS EVOLUTIONARY ALGORITHMS

For practical usage of EAs, or any algorithm for that matter, it is important to reduce the number of hyperparameters (which determine the settings of an algorithm and control the (optimization) process) as much as possible, or, at least, have an algorithm that performs well with the default hyperparameters across a wide range of problems. An illustrative example of why a strong dependence of performance on chosen hyperparameters is problematic is the following. Suppose that a GA is used for hyperparameter tuning of a Machine Learning (ML) model, but the GA has a number of hyperparameters as well. This is an undesirable situation because it only shifted the tuning from one type of problem to another.

GOMEA does not have the typical tunable hyperparameters that most GAs have, such as mutation rate, crossover, and selection types. In principle, the linkage learning model type can be selected from a few possible options, but in practice, the Linkage Tree model was shown to perform well on a variety of different problems. This makes GOMEA very convenient to use for solving real-world optimization problems.

Arguably the most crucial hyperparameter for an EA is the population size. It is highly problem dependent to set it to the right size, and there is no universal solution to determine it. Setting the population size to a large value means slower convergence and potentially, a huge amount of unnecessary computation, while setting it to a lower value restricts the diversity of solutions, and might limit the search process too much to get good solutions. One of the first proposed approaches to automatic adjustment of the population size as well as other parameters is called the Interleaved Multistart Scheme (IMS) which is inspired by the parameterless GA [17]. It works by automatically creating and managing multiple populations of increasing sizes at the same time so that small populations might converge, but if the optimization problem is difficult to solve, larger populations will be used. This scheme was also used in some GOMEA versions, e.g., in [12] and Chapter 4. In this thesis, we also use the more recent population parameter-free scheme called Parameterless Population Pyramid (P3) [18]. It was shown to be more efficient in terms of the number of required fitness evaluations [18] which is crucial for expensive optimization applications. The idea behind P3 is to gradually expand the set of solutions by adding new solutions one-by-one. In detail, it is described, for instance, in [18]. This population management scheme removes the necessity of manually setting the population size hyperparameter, while not inducing any additional hyperparameters.

Lots of different versions and modifications of GOMEA have been proposed, but a study that analyzes them all and concludes what is the best-performing version of GOMEA for unconstrained combinatorial optimization problems is missing in the literature. This is an important question to be answered for both researchers of EAs and practitioners who apply GOMEA to real-world optimization problems.

RESEARCH QUESTION 1

What is the best configuration of parameterless GOMEA for solving unconstrained combinatorial optimization problems?

In Chapter 2 we address this question by studying what are the best design choices (linkage model, linkage measure, etc.), population management scheme, and, importantly, GOM variation for GOMEA. Moreover, we study what design choices have the most impact on performance. Experiments are done on a diverse set of commonly used combinatorial optimization benchmark problems. The ultimate goal of this study is to obtain a version of GOMEA that shows state-of-the-art performance on a set of diverse problems. The contents of this chapter are based on the following publication:

> **A. Dushatskiy**, M. Virgolin, A. Bouter, D. Thierens, and P. A. N. Bosman. “Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms”. In: *Evolutionary Computation* (June 2023), pp. 1–28.

1.1.4. EXPENSIVE COMBINATORIAL OPTIMIZATION

Expensive combinatorial optimization is a special case of combinatorial optimization. Though there is no standard formal definition of it, it is usually presumed that the fitness calculation is so computationally expensive that the use of common approaches to solving combinatorial optimization problems becomes infeasible. Important examples of expensive combinatorial optimization problems can be found in ML. For instance, tuning the hyperparameters of an ML model on a large dataset might be quite computationally costly. The problem of expensive fitness evaluations is even more aggravated in deep learning-related problems, such as hyperparameter tuning of a neural network training process, and searching for the best possible architecture of a deep neural network for a specific dataset (Neural Architecture Search or NAS). In a traditional NAS approach, a fitness evaluation entails full deep neural network training (e.g., [19]), and it might take hours, if not days, even on a modern system equipped with Graphics Processing Units (GPUs). This raises the question of the feasibility of adopting traditional optimization algorithms. In the expensive optimization scenario, it is usually acceptable that the global optimum is not reached (and it is often not known in real-world applications). Rather, it is important to find good solutions within a few fitness evaluations. The number of allowed fitness evaluations depends on the computational cost of the problem and the available computational resources, but for a typical expensive optimization problem it is between a hundred and a few thousand evaluations. It is radically different from standard combinatorial optimization where sometimes millions or even billions of function evaluations is considered to be acceptable to solve the problem, i.e., to find the global optimum. The quantitative performance of expensive optimization algorithms is usually evaluated as the best fitness value found after performing a particular number of function evaluations.

1.1.5. ALGORITHMS FOR EXPENSIVE COMBINATORIAL OPTIMIZATION

The challenge of creating an efficient combinatorial optimization algorithm is two-fold: first, effective navigation of the search space to find good solutions is important, and, secondly, some special mechanisms should be integrated in order to reduce the computation costs as much as possible.

The most common approach to reducing the number of performed fitness evaluations is integrating a so-called *surrogate model* into an optimization algorithm. The purpose of a surrogate model is to estimate the fitness of a solution without performing an actual fitness calculation. Such a fitness estimate is further referred to in this thesis as *surrogate fitness*, as opposed to *real fitness* which is the fitness value produced by the original fitness function. The surrogate model usage is beneficial if it can estimate fitness much faster than performing a real fitness evaluation would take. However, the challenging part of surrogate model integration is that usually it can only approximate fitness, i.e., it will exhibit some degree of error.

A surrogate model is, formally, a function that takes as input a solution x from the search space $\mathscr{D}$, and outputs its fitness estimate: $\hat{f} : x \in \mathscr{D} \mapsto \mathbb{R}$. The goal is to have a surrogate model which is as accurate as possible, thus, surrogate modeling can be seen as a standard regression task. Hence, a surrogate model can be any regression model used in ML, e.g., a linear regression, a random forest [20], or a neural network. It can be trained on a set of solutions for which the real fitness is known. Standard regression

loss and metrics (e.g., the mean squared error and the R^2 coefficient respectively) can be naturally used for training and model evaluation. Identical to regression in ML, evaluation should be done on a set of examples previously unseen in the training set. Common evaluation techniques known from ML, such as cross-validation, can also be used. There are no fundamental differences in using a surrogate model for real-valued and discrete search spaces. However, in the discrete case, solutions should be preprocessed similarly to the preprocessing of categorical variables in ML. For instance, the one-hot encoding technique might be utilized. For some types of surrogate models, such as neural networks, additional preprocessing techniques such as normalization might also be required.

Probably the most straightforward way of using a surrogate model divides the search process into two stages: 1) Collecting a number of solutions (for instance, sampling random solutions from the search space), performing real fitness evaluations for them, and training a surrogate model on them; 2) Using this surrogate model solely in the optimization algorithm which means that no real evaluations are further performed and only estimates made by the surrogate model are used as solution fitness. Hypothetically, if a surrogate model is perfectly accurate (its fitness estimation always precisely matches with the real fitness), there is no need to perform more real fitness evaluations: one can just use this surrogate model and always use its fitness prediction instead of real fitness as there is no difference between them. An efficient (in terms of required solutions with known fitness values) creation of such a model for an arbitrary optimization problem seems unrealistic. However, it is interesting to study whether it is possible at least in some specific cases, such as for binary variables. Moreover, the linkage learning problem is closely related to the problem of creating such a surrogate model. Indeed, if all dependencies between variables are known, one can use a so-called *Walsh decomposition* to represent any fitness function of binary variables [21]:

$$f(x) = \sum_{s \in \{0,1\}^\ell} w_s \psi_s(x),$$

where

w_s are so-called Walsh coefficients,

$\psi_s(x) = (-1)^{bc(s \wedge x)}$,

$bc(i)$ is a bit counting function that indicates the number of ones in a vector i,

and $\wedge$ is the bitwise binary AND operator.

Hence, the Walsh decomposition can be seen as a surrogate model (with learnable coefficients w_s) which approximates the true fitness function f. We address the question of whether a perfectly accurate surrogate model can be obtained within a reasonable computational budget.

RESEARCH QUESTION 2

Is it possible to efficiently create a perfectly accurate surrogate model in a black-box optimization scenario (and then use it to solve an optimization problem)?

We use the Walsh decomposition as a surrogate function and propose how to efficiently learn the linkage structure of a problem, and then the Walsh decomposition coefficients. After a perfect surrogate model is obtained, for instance, GOMEA can be used to solve the optimization problem without performing more real evaluations. The favorable computational complexity of the Walsh surrogate model creation can be potentially obtained and proved for a specific type of fitness function. This is studied and discussed in Chapter 3. The contents of this chapter are based on the following publication:

> **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "A novel approach to designing surrogate-assisted genetic algorithms by combining efficient learning of Walsh coefficients and dependencies". In: *ACM Transactions on Evolutionary Learning and Optimization* 1.2 (2021), pp. 1–23.

Though there are works using an approach of first training a surrogate model and then fully relying on it during the optimization for practical applications such as NAS [22, 23], the main drawback of it (assuming that the surrogate model is not perfectly accurate) is that the surrogate model does not adapt and focus on more promising parts of the search space as the search process proceeds. A natural modification of this approach includes the interleaved usage of surrogate fitness and real fitness evaluation during the second stage of an algorithm (when the actual search using the trained surrogate model is performed). Namely, the second stage of the above-mentioned approach can be modified as follows: 2.1) Run an optimization algorithm to find the best solutions according to the surrogate fitness; 2.2) Perform real evaluations for a selection of top solutions (selected using a specified criterion); 2.3) Update the surrogate model using these newly real-evaluated solutions and go to 2.1 again. Such an approach can demonstrate a good performance when the number of allowed fitness evaluations is very limited [24].

A wide class of algorithms for expensive optimization is called Bayesian Optimization (BO) algorithms. The key idea of BO algorithms is to have a surrogate model which can estimate not only the fitness value of a solution but also the variance of such an estimate. In step 2.1 of the above-described algorithm, the surrogate fitness can then be replaced with a so-called acquisition function. Acquisition functions are used to calculate the performance score of a solution, taking into account both its fitness estimate and its variance. The most promising solutions are selected according to their acquisition function value, rather than just their surrogate fitness estimate. As the variance of the fitness estimate should be higher for solutions in under-explored regions of the search space, such an approach can, potentially, automatically ensure a good balance between the search space exploration and exploitation of its most promising regions. While many BO algorithms focus on real-valued optimization problems [25], there are a few works on BO algorithms for combinatorial optimization. One such algorithm which is commonly used in practice is Sequential Model Algorithm Configuration (SMAC) [26]. Another example is the Tree Parzen Estimator (TPE) [27] of which the most commonly used implementation can be found in the Hyperopt optimization framework [28]. Both algorithms are capable of dealing with discrete, real-valued, and mixed search spaces and were developed specifically for the expensive optimization scenario. In this thesis, we use SMAC and Hyperopt as baselines in order to evaluate the ability of the developed algorithms for expensive optimization to attain state-of-the-art performance.

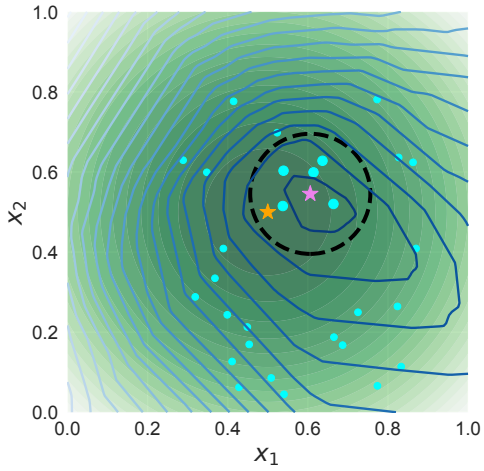


Figure 1.1.1. Illustration of the conceptual work principle of a surrogate-assisted EA. The objective function here is of two variables x_1 and x_2. The isolines of the true fitness function and its surrogate approximation are shown in green (contours on the background) and blue (lines), respectively. The true function maximum is shown with an orange star, while the maximum according to the surrogate model (i.e., the solution with the largest so far obtained surrogate fitness value) is shown with the pink star. The surrogate model is trained on a limited number of samples, therefore, it does not perfectly approximate the true fitness function. Cyan points show considered points during the search. For all points, a surrogate evaluation is performed first. For certain points (depicted as larger dots) also a real evaluation is performed. A real evaluation is performed for a solution if its surrogate fitness is close enough to the best obtained surrogate fitness (it is located inside the black dashed line).

An alternative approach is to integrate a surrogate model into an optimization algorithm, for instance, an EA, by replacing a part of the real fitness evaluations it performs by surrogate ones. In contrast to BO algorithms, real and surrogate evaluations can be mixed together during the search process (depending on a certain criterion, it is decided whether to use a real evaluation or a surrogate one). Potentially, such an approach can preserve the search capability of an EA, but at the same time reduce the number of performed real function evaluations as for some solutions only surrogate ones are used. EAs with an integrated surrogate model are usually called *surrogate-assisted EAs*.

The GOMEA family of algorithms has been demonstrated to achieve excellent performance in various search domains. We hypothesize that a modification of GOMEA which uses a surrogate model might also show state-of-the-art performance compared to existing approaches (including BO algorithms) on expensive combinatorial optimization problems. Therefore, in this thesis, we propose and study the approach of adapting EAs (specifically, GOMEA) to solving expensive optimization problems.

RESEARCH QUESTION 3

How can GOMEA be adapted to effectively find high-quality solutions for expensive combinatorial optimization problems and show state-of-the-art performance on various problems (including real-world ones) with up to hundreds of binary and higher cardinality discrete variables?

In the proposed approach, most of the time, the search relies on surrogate evaluations, however, for the most promising solutions, real evaluations are performed. In contrast to BO algorithms, the promising solutions for which real evaluations are performed are selected based on their surrogate fitness only (the variance of the prediction is not taken into account). For instance, one possible way to select such solutions is to check, whether a solution has a surrogate fitness that is close to the currently best obtained surrogate fitness value. The idea of the proposed approach is shown schematically in Figure 1.1.1. We study not only how a surrogate model can be integrated into GOMEA, but also what types of surrogate models show the best performance. We show how to integrate a surrogate model for computationally cheap fitness estimation into GOMEA. Performance comparisons are made against standard GOMEA and BO algorithms. In Chapter 4 we show the proof of principle of this idea on benchmark functions with binary variables. We call this developed algorithm Convolutional neural network Surrogate-assisted GOMEA (*CS-GOMEA*). This research is followed up in Chapter 5 by extending CS-GOMEA to non-binary real-world combinatorial optimization problems. This algorithm is further referred to as Surrogate-Assisted GOMEA (*SAGOMEA*).

The content of Chapters 4 and 5 are based on the following publications:

1. **A. Dushatskiy**, A. M. Mendrik, T. Alderliesten, and P. A. N. Bosman. "Convolutional neural network surrogate-assisted GOMEA ". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 753–761.
2. **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "A novel surrogate-assisted evolutionary algorithm applied to partition-based ensemble learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2021, pp. 583–591.

1

1.2. DEEP LEARNING FOR COMPUTER VISION

Neural networks were originally proposed as a computational model which approximately imitates the neurons, the connections between them, and the signal propagation mechanism in the human brain. Neural networks consist of artificial neurons which are computational units (functions) and produce an output for a given input. A neural network is typically composed of layers (which in turn are composed of neurons). This is schematically shown in Figure 1.2.1 for the case of a simple *feed-forward* network (i.e., the computations are performed sequentially from the first layer to the last one).

Deep learning is a more recent development of this field. In deep learning, *deep neural networks* are used which formally can be characterized by the presence of multiple layers, and, informally, denote more complex networks, with different types of neurons with which sophisticated operations can be performed. Arguably, the modern deep neural networks are not necessarily related to the structure of a human brain [29], but are nevertheless powerful computational models.

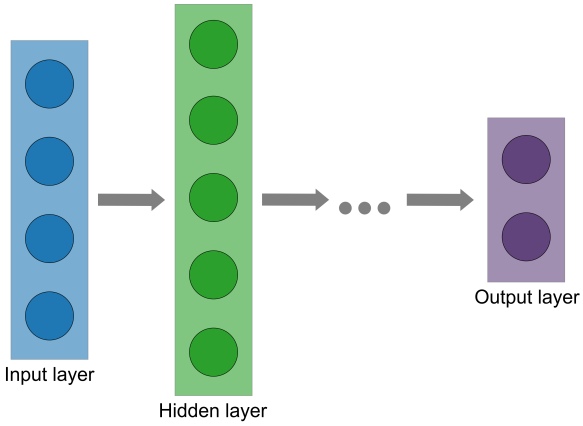


Figure 1.2.1. Schematic illustration of a simple feed-forward neural network with input/output layers and (multiple) hidden layers. Colored cirlces denote neurons and arrows depict the direction of the computation (the hidden and output layers use the outputs of the previous layer as the input).

Deep learning has significantly changed many challenging fields of computer science and expanded the limits of artificial intelligence capabilities, in many cases bringing them a level of performance equivalent to that of humans. Examples of such domains where deep learning has had much impact are natural language processing, reinforcement learning, and computer vision. There are multiple factors contributing to the success of deep learning (their order by impact is arguable and difficult to determine): 1) Progress in computational capabilities of GPUs, supported by the development of more efficient low-level (CUDA) and high-level (for instance, PyTorch) software for programming using GPUs; 2) Algorithmic developments in various aspects of deep learning: more capable architectures [30, 31], regularization techniques [32], and training procedure refinements, (e.g., for faster and better training [33]); 3) More available training data [34]. The synergy between these factors allowed the recent huge progress in deep learning, making it the de-facto approach to solving problems in various domains, including computer vision.

A key component of a neural network is its collection of *learnable* (adjustable) weights. Weights are parameters of the computational units of neural networks. In general, a neural network can be represented as a function f which has parameters (weights) θ and produces an output $\hat{y}$ for the input x: $y = f(x,\theta)$. The idea behind learning the weights (this process is called *network training*) is adapting them so that the function f better aligns with the data. Using the given data, the computational function of the neural network is changed in order to produce an output tailored to the task this network is supposed to solve, for instance, classifying the input into two categories (binary classification).

A deep learning model can be defined by a triplet $(\mathscr{A},\theta,\mathscr{H})$, where $\mathscr{A}$ is the architecture defining a computational graph which transforms the input and produces the output, θ represents the learnable weights, and $\mathscr{H}$ denotes the hyperparameters of the training procedure. In the case of traditional supervised learning, the training dataset consists of pairs of input data and associated labels (also called the training samples): $D_{train} = \{(x_i, y_i) \mid i = 1 \dots N\}$ (N is the dataset size). Having as input a tensor x (a multi-dimensional array, e.g., an RGB image), a deep neural network produces an output $\hat{y}$.

To learn the weights, firstly the loss function $\mathscr{L}(\hat{y}, y)$ needs to be defined. The learning goal is to minimize the loss function over the training samples. The loss function should be differentiable to allow efficient optimization using a gradient-based method (usually, the number of weights is extremely large which makes the usage of other optimization algorithms infeasible). For instance, a commonly used loss function for a regression task is the mean squared error: $\mathscr{L}_{D_{train}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$.

The most simple gradient-based optimization algorithm for training neural networks is Stochastic Gradient Descent (SGD) which is a modification of standard Gradient Descent (GD). In GD and SGD training is started with random initialization of the weights. The weights are then updated in multiple steps. At each step, the weights are updated in the opposite direction of the gradient, i.e., in the direction of the steepest descent of the function. This way, a local minimum of the function can be effectively found. However, calculating the true gradient (using the loss $\mathscr{L}_{D_{train}}$ over the whole training dataset) is in most cases computationally intractable. In contrast to GD, SGD does not use the true gradient and instead uses its approximate estimate on subsets of the data (called *batches*) $B \subseteq D_{train}$. Training is performed on randomly composed batches. Typically, batch size $|B| << N$. The update of the weights from step t to step $t+1$ in SGD is then defined as follows:

$$\theta_{t+1} = \theta_t - \alpha \nabla \mathscr{L}_B(\theta_t),$$

where the loss $\mathscr{L}_B$ is computed only using the samples from the current batch B. Update step size α (*learning rate*) is a hyperparameter of SGD. The loss gradient is therefore calculated with respect to the current batch, and updating the weights is performed accordingly. Weights in the hidden layers of a neural network are updated using the *backpropagation* technique, which, basically, uses the chain rule of differentiation to calculate the loss gradient with respect to the weights of a hidden layer. There exist numerous more advanced modifications of SGD, such as SGD with Momentum [35] and Adaptive Moment Estimation (ADAM) [36], but all of them share the general principle of SGD: at each training step (on one batch of data) weights are updated using the gradient information of the loss function. Hyperparameters of the optimizer (for instance,

the learning rate), as well as the loss function definition, are included in the model hyperparameters $\mathscr{H}$.

The computational graph of a neural network, also commonly called *an architecture*, defines the transformations applied to the input data. A typical deep learning model works by learning the weights associated with transformations that allow to extract and process features from the data, starting from more low-level (abstract) ones, progressively moving to consider more high-level features. While feature extraction operations may vary depending on the data type (image, sound, text, graph, tabular data), one of the key ideas which allowed rapid progress in computer vision, and that is widely used in this thesis is the *convolution* operation. The main principle of this operation is to apply a filter that acts as a feature extractor with the same weights to different locations of the image. Thus, a specific image feature is aimed to be captured independently of its location. A schematic example of the application of the convolution operation (one convolutional filter in one convolutional layer) is shown in Figure 1.2.2. Typically, many convolutional (tens or even hundreds) layers are stacked and applied sequentially.

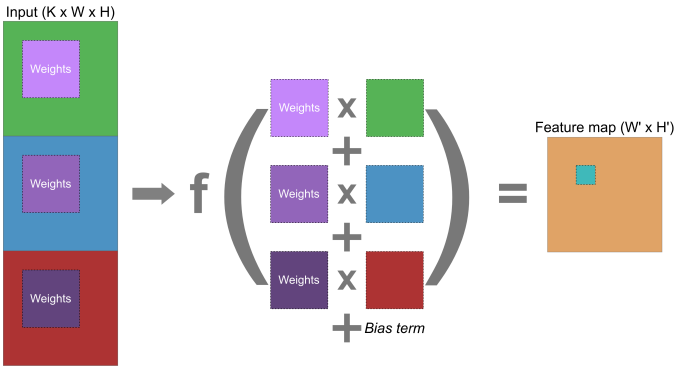


Figure 1.2.2. Illustration of a convolution operation scheme. A filter with learnable weights is slid over the input of the dimensionality $K \times W \times H$ (an RGB image, in this case, channel-wise flattened). At each location, the standard inner product of a corresponding sub-matrix of the input and the weights matrix is calculated. Usually, the summation of these products is followed by adding a bias term (learnable) and applying a non-linear transformation (activation function). The commonly used activation function in modern convolutional neural networks is the Rectified Linear Unit (ReLU): $f(x) = max(0, x)$. Thus, a single value in the corresponding position of the output (typically called the feature map) is obtained. In hidden layers of the neural network, the convolution operations use outputs from the previous layers as inputs. A single convolutional filter of size $F_W \times F_H$ over a single-channel input has $F_W F_H + 1$ learnable weights (including the bias term). If an input has more than one channel (K channels), then a separate weights matrix is used for each channel, and their multiplication results over corresponding channels are summed up. Then, the number of learnable weights becomes $KF_W F_H + 1$ (bias terms are typically identical for all input channels). Usually, more than one filter is applied, each having its own weights. Thus, a convolution operation with D filters has $DKF_W F_H + D$ weights. The output dimensionality after applying a single filter is $W' \times H'$ ($D \times W' \times H'$ when multiple filters are applied) and it is not necessarily identical to the input dimensionality.

1.2.1. NEURAL ARCHITECTURE SEARCH

Computer vision tasks vary a lot: even the subclass of image classification might mean classification of natural images (photographs) or classification of medical images (for instance, for the purpose of deciding if an image contains a tumor). Using a deep neural network tailored for a specific combination of an image dataset and task (classification, object detection, segmentation, etc.) might, potentially, provide better performance than using a general-purpose network. However, the number of options for architecture design is immensely large, making it virtually impossible to optimize the architecture by hand. This raises the question of whether neural network architecture design can be done automatically and what gains are possible by doing so. The study of this type of automation is called *Neural Architecture Search*. Its general overview is shown in Figure 1.2.3.

It should be noted that while an architecture of a neural network might have a significant effect on its performance, it was also shown in [30, 32, 37], that there are other crucial components, such as data preprocessing, the hyperparameters of the training procedure, and data augmentations, that might have a significant impact on the performance. Moreover, the amount and the quality of the collected data have arguably, the largest influence. Nevertheless, in a real-world scenario, it is often impossible to increase the dataset size, leaving NAS (as well as training of the hyperparameters) to be considered as a possible option for improving the performance of a deep learning model.

NAS can be considered as an expensive combinatorial optimization problem. What makes it particularly interesting are the noisy fitness evaluations. One fundamental reason for fitness evaluations being noisy is that neural network training is a stochastic process. Multiple factors contribute to this: 1) Different random initial weights of a network usually result in slightly different final performance; 2) The training algorithm is SGD which means that batches of training data are sampled randomly; 3) Data augmentations (a commonly used regularization technique) are usually applied with some probability (e.g., an image flip can be applied with probability 0.5). Therefore, the same architecture initialized and trained with different random seeds usually has a (slightly) different performance per random seed. The second reason for noisy fitness evaluations in NAS comes from the way performance is measured. Usually, a fixed validation set is used, however, it might happen that a network shows different performance on different validation sets, making the performance estimate on just one validation set unreliable. This noise problem is naturally more aggravated in the case of relatively small and heterogeneous datasets, a situation that often happens in NAS for medical image analysis tasks, e.g., segmentation.

To the best of our knowledge, it has not been studied in detail to what extent noise in the network performance evaluation (i.e., NAS fitness function) has a negative impact on the performance of optimization algorithms for NAS. We believe however that this question is important to take into consideration when new NAS algorithms are designed and then applied in practice. A better understanding of the trade-off between a more computationally expensive evaluation of an architecture (for instance, by averaging over multiple random seeds used for network training) and the amount of noise it contains might potentially allow for finding better-performing architectures using NAS.

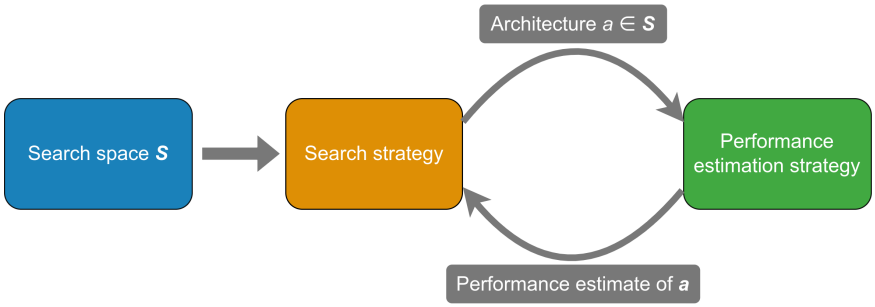


Figure 1.2.3. A general overview of NAS as formulated in [38]. After defining a search space of possible architectures, an optimization algorithm can be used to find the best-performing architecture. During optimization, iteratively a new architecture is probed and a performance score is determined by the performance estimation strategy (e.g., a score on the validation set). Optimization is continued until the computational budget is exhausted.

RESEARCH QUESTION 4

Does the innate stochasticity of fitness evaluations in Neural Architecture Search represent a significant hurdle for optimization algorithms, and, if yes, can it be alleviated?

This research question is discussed in Chapter 6. We focus on NAS for medical image segmentation and use two open-source datasets (tasks of multi-class segmentation of the prostate and heart areas) for experiments. Different optimization algorithms are considered, including those which are designed specifically for problems with expensive fitness evaluations. The contents of this chapter are based on the following publication:

> **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "Heed the noise in performance evaluations in neural architecture search". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM. 2022, pp. 2104–2112.

1.3. MEDICAL IMAGE SEGMENTATION

Similar to other domains mentioned in Section 1.2, in medical image segmentation great progress has been recently achieved with the use of deep learning. It has been shown that deep learning models are capable of achieving human (expert) level performance in some important applications: organs at risk segmentation for head and neck radiotherapy [39] on computed tomography scans and lungs and heart segmentation on chest radiographs [40]. However, even with these advances, challenges remain to achieve wider adoption of deep learning-based segmentation methods in clinical practice [41]. We study some of these challenges in more detail in this thesis.

1.3.1. THE TASK OF MEDICAL IMAGE SEGMENTATION

Medical image segmentation is one of the most practically important tasks in computer vision. The goal of it is to automatically segment, or, in other words, contour, areas of interest on medical images (scans). In this thesis, we call a segmentation method *automatic* even though it is assumed that in clinical practice a machine-performed segmentation might be followed by a manual correction if needed. We also note that our focus is on developing the segmentation algorithms which do not use the paradigm of human-in-the-loop learning [42] (e.g., receiving feedback from clinicians during training to improve the segmentation model).

The type of medical scan can be Magnetic Resonance Imaging (MRI), Computed Tomography (CT), X-ray, or, in general, any type of medical image used in clinical practice. What particularly should be segmented, depends on the medical goal. It can, for example, be an organ of interest, a tumor, or an individual bone. Generally, the input can be a combination of images of different modalities (for instance, MRI and CT). The output can also be a joint segmentation of different targets, in most cases, they are mutually exclusive. Formally, the segmentation task can be defined as follows:

$$\Phi : T_{M \times X \times Y \times Z} \mapsto S_{C \times X \times Y \times Z}$$

where

Φ is a segmentation model, for instance, a deep neural network,

T and S are tensors representing the scan (*input*) and the segmentation (*output*),

$X \times Y \times Z$ is the spatial dimensionality of the scan and its segmentation,

M is the number of input modalities,

C is the number of segmentation classes,

the scan and model output are, in general, real-valued tensors.

The output of a deep learning model, usually, represents the probabilities of segmentation classes in a particular voxel. The value p in position (c, x, y, z), (i.e., S_{cxyz}) means that the voxel with coordinates (x, y, z) belongs (according to the model) to class c with probability p. For practical usage, the class with the largest predicted probability can be selected, and the segmentation prediction can be binarized. Then, the value one in position (c, x, y, z), means that the voxel with coordinates (x, y, z) belongs to class c. In general, the segmentation task can be considered to be a voxel-wise classification.

While the above-given formulation of the medical image segmentation task is general, there are potential pragmatic choices in practice that change the original input before executing the task. For instance, for 3D medical images, the segmentation is sometimes performed in 2D, which means that instead of getting a 3D volumetric scan as an input, the model gets 2D slices obtained from the scan. In general, 3D segmentation has the potential for better performance [37] as the model can capture dependencies between parts of the scan in all dimensions. However, in practice, 2D segmentation is often used due to much larger computational resources needed to train a deep neural network to perform a 3D segmentation.

1.3.2. SEGMENTATION QUALITY METRICS

For measuring performance and achieved progress in medical image segmentation, just as in any other ML subfield, it is critically important to have commonly used performance metrics. Likely, the most commonly used metric in medical segmentation is the Dice coefficient [43]. For multi-class segmentation (C classes) it is defined by the formula $\frac{1}{C}\sum_{c=1}^{C}\frac{2|R_c \cap P_c|}{|R_c|+|P_c|}$, where R_c and P_c are, correspondingly, the reference and the predicted segmentation masks for class c. Usually, the background class (which does not represent any particular organ) is omitted from the calculation. Dice coefficient values range between 0 (completely wrong segmentation) and 1 (perfectly accurate segmentation). This metric is easy to calculate and it provides a general notion of how good the predicted segmentation is. The main drawback of it is that it does not reflect all aspects of the practical usage of automatically produced segmentations. Specifically, an automatically produced segmentation that is very similar to the reference (meaning a high Dice coefficient, close to 1), but with a small area of the image segmented in a very wrong way, still requires a manual correction from a clinician. At the same time, it might be that an algorithm produces a segmentation that is slightly less accurate in general (resulting in a smaller Dice coefficient value) but does not require any manual correction, because there are no critical (according to the clinician) inaccuracies in the segmentation. Another negative aspect of the Dice coefficient is that it is easier to obtain a large value for regions of interest that are large, i.e., missing one voxel (or pixel in the 2D case) in a small object has the same loss in Dice as missing quite a large part in a very large object. For these reasons, it is useful to also consider other metrics, in particular, the Surface Dice coefficient [39].

The Surface Dice value denotes the fraction of the segmentation contour which deviates from the reference contour by no more than τ mm (needs to be set as a parameter). Similar to the Dice coefficient, the Surface Dice coefficient has values ranging between 0 (segmentation needs major manual adjustment, i.e., the whole predicted contour deviates from the reference contour by more than τ mm) and 1 (segmentation can be used without manual adjustment, i.e., the whole predicted contour does not deviate from the reference contour by more than τ mm). In this thesis, both Dice and Surface Dice (with a varying threshold value) coefficients are used to evaluate the performance of segmentation methods.

1.3.3. NEURAL NETWORK ARCHITECTURES FOR MEDICAL IMAGE SEGMENTATION

Probably the most commonly used neural network architecture for segmentation is the U-Net [44]. The main idea behind it is to use an encoder-decoder structure. The encoder works like a typical deep neural network for classification: it gradually reduces the spatial dimensionality of the input while extracting more features from it. While the purpose of the encoder is to extract and compress features from the input scan (or a slice in the 2D case), the decoder translates these features into the segmentation. It works by gradually upsampling the compressed features until their spatial dimensionality becomes equal to the input. Importantly, in the decoder, the feature maps of the corresponding spatial dimensionality from the encoding path are used, which allows using more information in the upsampling path and therefore contributes to better learning of the connection between the scan and its segmentation. A standard U-Net architecture is shown in Figure 1.3.1.

The U-Net can be naturally generalized beyond the originally proposed U-Net architecture, and there exists a large family of various architectures with similar encoder-decoder structures. The original U-Net relies on a rather simple (compared, for instance, to state-of-the-art deep learning models for classification) structure consisting of blocks similar to the ones used in the Visual Geometry Group (commonly known as VGG) architecture [45]. Both the encoder and decoder can be modified in order to obtain better performance. A common and efficient U-Net design strategy is to modify only the encoder by using state-of-the-art architectures developed for image classification (usually, natural image classification). The benefit of such an approach is that one can use an open-sourced pre-trained encoder on large classification datasets such as Imagenet [46]. The usage of such pre-trained encoders was shown to be, in most cases, beneficial for performance on the medical image segmentation task, especially when the amount of medical data is limited [47, 48]. The encoder architecture does not have to be limited to a convolutional network only. For instance, the recently introduced transformer architecture can also be used in an encoder, as shown in [49, 50].

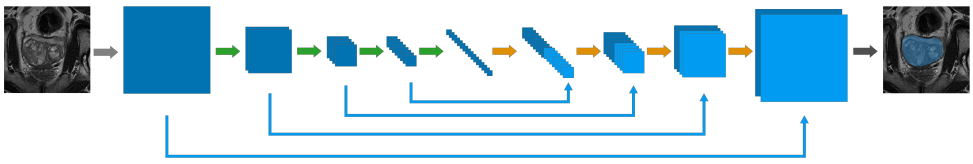


Figure 1.3.1. Generalized scheme of a standard U-Net architecture for medical image segmentation. Feature maps are depicted with blue squares. Green arrows denote downsampling operations which decrease the spatial dimensionality of feature maps but increase the number of channels (for instance, a convolutional layer with stride 1 followed by a pooling layer with stride 2). Orange arrows denote upsampling operations which increase the spatial dimensionality of feature maps but decrease the number of channels (for instance, a transposed convolutional layer with stride 2). Blue arrows denote the concatenation of feature maps. Gray arrows denote input and output convolution operations.

Medical image segmentation is known for the variety of datasets and tasks it can be applied to. Datasets vary in size, modality (the most common ones are MRI, CT, ultrasound, X-ray), image resolution, and scanning devices used for their creation. The segmentation tasks themselves also vary a lot, from segmenting single or multiple organs, to a pathology (such as tumor). Therefore, using NAS for creating a specialized network architecture for a specific medical image segmentation task, might, potentially, have a larger impact than, for instance, in the image classification domain.

1.3.4. DATA VARIATION

Heterogeneity of image segmentation data often occurs when the data is collected in a real-world clinical environment. In general, data variation can be classified into *image*- and *observer*- variation.

Image variation means that a dataset might contain scans that visually look different: different brightness, contrast, or slightly different patient positioning when the scan was taken. This type of variation originates from the variability of scanning devices, their settings, and clinical scanning protocols.

The observer variation can be divided into two different subtypes of variation and has a different origin. It was shown in, for instance, [51, 52] that given the same scan, different clinicians might segment it in slightly different ways (*inter-observer* variation). The second subtype of observer variation is *intra-observer* variation which means that there might be inconsistencies between segmentations made by the same observer if they are given the same scan multiple times. While both types of observer variation should be taken into account, inter-observer variation is naturally larger [53] and therefore might be more problematic. It turns out that the Dice similarity coefficient between Prostate segmentation on MRI scans made by different observers is about 0.9 [54]. Interestingly, in [54] there is no strong connection between years of experience in a group of clinicians and their inter-observer variation value. This might indicate that the inter-observer variation (or, at least, a substantial part of it) comes not from the fact that an observer makes a mistake while segmenting a scan, but from the fundamental ambiguity of the segmentation task. Given that the images do not perfectly and unambiguously visualize the boundaries of regions of interest, there is a lack of a golden standard. In other words, *more than one segmentation is correct*, at least according to different observers.

1.3.5. OBSERVER VARIATION-AWARE MEDICAL IMAGE SEGMENTATION

The most common approach to applying deep learning to medical image segmentation is done under the assumption that there is only one correct way of segmenting a scan, and therefore, a standard neural network architecture for segmentation (such as the U-Net) produces one segmentation prediction for an input scan. However, for a better acceptance of deep learning-based segmentation methods (in general, any automatic segmentation methods) and their wider integration in clinical practice, it might be useful to have an algorithm that imitates a group of humans segmenting a scan, namely, producing multiple possible segmentations which represents variation in the segmentations made by different clinicians. More generally, in this thesis, we call different plausible ways of segmenting a scan *different segmentation styles* (the same concept is used in [55]), and one segmentation style refers to not necessarily one clinician, but a group of clinicians

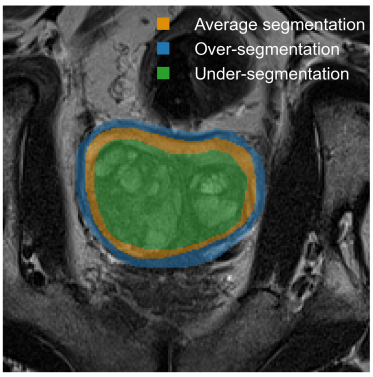


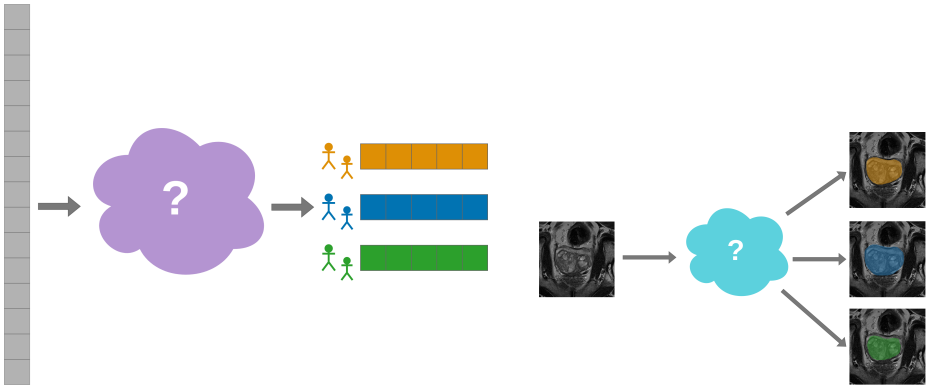
Figure 1.3.2. An example of three different, hypothetically possible styles of prostate segmentation on an MRI scan slice. The image data and the average segmentation variant are taken from the prostate segmentation task of the Medical Segmentation Decathlon [56], over- and under-segmentation variants are simulated. In this example, the under-segmentation variant is fully contained in the average one, which, in turn, is fully contained in the over-segmentation variant.

which perform the segmentation in the same way. The *segmentation style* concept used in this thesis can be seen as a more informal way of referring to inter-observer variation. A simple example of different styles of segmentation is to over- and under-segment an organ. This is visually demonstrated for prostate segmentation in Figure 1.3.2. Some clinicians might have a preference for one of these specific styles rather than having one style of segmenting that would represent an average segmentation [53]. In this example, if a deep learning model can produce three segmentation predictions corresponding to these segmentation styles, potentially, there is a higher chance that a clinician finds at least one of the automatically produced segmentations acceptable for clinical usage because it resembles his or her way of segmenting the scan, reducing the need for manual re-annotations.

In this thesis, we mainly consider the following data availability scenario. A dataset likely contains scans segmented by different observers and thereby represents different segmentation styles. However, the information about the observers, or the segmentation styles, is not necessarily readily available. We see such situations appearing in clinical practice, for instance, at the Amsterdam University Medical Centers (location AMC). Visually, the considered scenario is shown in Figure 1.3.3a.

1.3.6. DATA VARIATION TAKEN INTO ACCOUNT IN DEEP LEARNING

Some works acknowledged the problem of data variation and attempted to take it into account. In, e.g., [57, 58] datasets with multiple segmentations per scan are considered, but the assumption is that some observers or particular segmentations are better than others. The goal is then to find erroneous reference segmentations or fuse multiple segmentations together, diminishing the negative effects of individual mistakes. While such techniques might be useful for data quality improvement, this problem is fundamentally different from our main focus.



(a) The considered data availability scenario. The dataset contains multiple scans (depicted by squares), segmented by six different observers, but only by one observer per scan. Presumably, among the observers there are some specific styles of segmenting, however, the mapping between scans and styles is not necessarily known. In this example, the dataset has three distinct segmentation styles (shown by different colors). For simplicity, it is not shown here that the observers might agree on how to segment one part of the region of interest while disagreeing on how to segment the remainder of it.

(b) An envisioned usage of an automatic segmentation approach. Given a medical scan as input, the approach is supposed to produce multiple segmentation variants, each corresponding to a specific segmentation style.

Figure 1.3.3. Considered data availability scenario (*a*) and automatic segmentation model employment (*b*).

A different direction of research related to this topic is usually called *uncertainty quantification* [59, 60]. The goal is to perform segmentation in a probabilistic manner, i.e., in addition to performing segmentation, producing information about its uncertainty. Knowing the uncertainty of segmentation can be used in practical medical applications, for instance, by defining an acceptable level of certainty and using only those segmentations/parts of the segmentation that the model is sure enough about. Proposed methods of obtaining prediction uncertainty include dropout [60], the ensemble of multiple models [61], and using an auxiliary network to estimate predictive uncertainty of the main segmentation model [62]. While a significant part of uncertainty is attributed to inter- and intra-observer variation, the uncertainty estimation problem is different from the task of explicitly capturing and exploiting observer variation.

One of the first works, which aims at capturing and taking into account observer variation for making segmentation predictions, is the probabilistic U-Net [63]. Similar to variational autoencoders [64], given an input scan, its component called a prior network produces the parameters of a multidimensional Gaussian distribution, samples from which are then used by a U-Net to produce a segmentation. Different samples from the distribution lead to different produced segmentations. This way, the probabilistic U-Net approach can produce a wide range of potentially feasible segmentations with respect to the probability of each segmentation variant. A similar but more advanced and better-performing approach was later proposed in [55]. However, such methods do not map a

segmentation to a particular segmentation style. Moreover, in the original probabilistic U-Net publication [63], the used dataset contained multiple (four) segmentations for each input image. We believe that the creation of such a dataset might take a lot of additional effort from clinical experts outside routine medical practice. Being able to directly work with clinical data is much more practically relevant in most real-world cases.

1.3.7. EXPLICITLY CAPTURING AND EXPLOITING OBSERVER VARIATION FOR SEGMENTATION

We hypothesized that for wider adoption of deep learning methods in clinical practice, not only a spectrum of possible segmentations needs to be produced, but it should be possible to connect the automatically created segmentations to the observers or group of observers in the dataset (in other words, a segmentation style). We believe such an approach has large potential value, yet it is missing in the current literature. Therefore, we study whether it is possible to develop such a method (which might include not only deep learning models for image segmentation but also additional optimization components if necessary).

RESEARCH QUESTION 5

Can deep learning and optimization techniques be combined to establish a novel medical image segmentation approach that is capable of explicitly capturing and exploiting observer variation in the dataset?

Potentially, if a deep neural network is trained on a subset of data segmented in one segmentation style, it can produce segmentations resembling that particular style. Thereafter, a collection of such models, each trained on its specific subset, can produce various segmentations, each corresponding to a particular style of segmenting. The challenging part of implementing such an approach is that the observers are not always listed in real clinical datasets. Moreover, there might be many observers, but only several groups of them may have distinctive styles of segmenting. To overcome both of these problems, we propose to consider automatically partitioning the dataset into subsets. We formulate then an optimization problem of which the goal is to obtain a dataset partitioning such that networks trained on the data subsets can produce diverse and high-quality segmentations. During the inference stage (ultimately, using trained models in practice), we can obtain multiple segmentations by utilizing each of the trained models. The envisioned usage is shown in Figure 1.3.3b. To evaluate the quality of a produced set of segmentations, we simulate the situation where a clinician sees multiple segmentations and selects the preferred one. In other words, we select the best segmentation among the produced segmentation variants for each scan according to the reference segmentation mask and calculate one or multiple quality metrics as described in Section 1.3.2. To avoid overfitting, this should be done on a different set of scans than the training set.

This dataset partitioning problem is naturally an expensive combinatorial optimization problem. In our studies, we use our own developed surrogate-assisted EA (CS-SAGOMEA) to solve this optimization problem, as it was shown to exhibit state-of-the-art performance in Chapters 4 and 5.

We note that the proposed method is, in general, capable of capturing and explicitly exploiting variation of any source, e.g., scans acquired by different scanning systems. In this thesis, we focus on the inter-observer variation. First, we use artificially created variations in order to demonstrate that the proposed approach is in principle capable of capturing different segmentation styles and producing multiple segmentation accordingly. Simulated variations are similar to the various segmentation styles as presented in Figure 1.3.2. For these experiments, we use the Prostate Segmentation Dataset from Medical Segmentation Decathlon [56]. This is studied and discussed in Chapter 7. The contents of this chapter are based on the following publication:

> **A. Dushatskiy**, A. M. Mendrik, P. A. N. Bosman, and T. Alderliesten. "Observer variation-aware medical image segmentation by combining deep learning and surrogate-assisted genetic algorithms". In: *Medical Imaging 2020: Image Processing*. Vol. 11313. SPIE. 2020, pp. 296–306.

One important application of automatic segmentation methods is their integration into automatic treatment planning systems for radiation oncology. One such system is BRachytherapy via artificially Intelligent GOMEA-Heuristic based Treatment planning (BRIGHT) [13]: a system for automatically finding treatment plans currently used for prostate brachytherapy. Brachytherapy is a form of internal radiation treatment that entails the insertion of catheters (implants) in the prostate for dose delivery purposes via a radioactive source that is guided through these catheters. The problem of treatment planning is formulated in BRIGHT as a bi-objective optimization problem. A state-of-the-art multi-objective optimization algorithm is used to solve the problem: MO-RV-GOMEA [65]. It was shown that this automatic approach is capable of finding better plans than manual plan adjustment performed by experienced clinicians [66]. An important feature of the used optimization algorithm is its speed-up using GPUs [12] which allows finding treatment plans fast, a crucial requirement for clinical usage. This algorithm has been used in clinical practice in the Amsterdam UMC, location AMC since 2020 [13]. To calculate and optimize treatment planning objectives, BRIGHT needs organ segmentations. Currently, these segmentations are made manually. However, potentially it might be possible to automate this part of the treatment planning pipeline as well by segmenting organs automatically. Moreover, multiple automatically produced segmentations might be used to ensure the robustness of the resulting treatment plans to the known observer variation in the organ contours, similar to already conducted studies of incorporating robustness into BRIGHT [67]. Verifying and potentially further improving the robustness of automatic treatment planning might be important for even wider clinical adoption.

While the study conducted in Chapter 7 is a necessary proof-of-principle of the proposed segmentation method, ultimately we are interested in integrating automatic segmentation in the treatment planning pipeline. This requires addressing the question of how our proposed algorithm performs on the prostate segmentation task with a real clinical dataset used for brachytherapy treatment.

RESEARCH QUESTION 6

Is the proposed observer variation-aware medical image segmentation method capable of producing high-quality and diverse segmentation variants on a real clinical dataset?

In Chapter 8, we use a real clinical dataset (CT scans with inserted catheters) in order to demonstrate that the proposed approach is capable of capturing different segmentation styles and producing segmentations accordingly in a real-world scenario. A dataset of CT scans that were acquired and used for brachytherapy treatment in the Mount Vernon Cancer Centre is used in this study. The contents of this chapter are based on the following publication:

> **A. Dushatskiy**, G. Lowe, P. A. N. Bosman, and T. Alderliesten. "Data variation-aware medical image segmentation". In: *Medical Imaging 2022: Image Processing*. Vol. 12032. SPIE. 2022, pp. 759–765.

REFERENCES

[1] D. H. Wolpert and W. G. Macready. “No free lunch theorems for optimization”. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82.

[2] A. Yang, Y. Shan, and L. T. Bui. *Success in evolutionary computation*. Vol. 92. Springer, 2008.

[3] K. Kannappan et al. “Analyzing a decade of human-competitive (“humie”) winners: what can we learn?” In: *Genetic Programming Theory and Practice XII* (2015), pp. 149–166.

[4] J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[5] D. E. Goldberg, K. Deb, and D. Thierens. “Toward a better understanding of mixing in genetic algorithms”. In: *Journal of the Society of Instrument and Control Engineers* 32.1 (1993), pp. 10–16.

[6] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, et al. “BOA: the Bayesian optimization algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vol. 1. Citeseer. 1999, pp. 525–532.

[7] D. Thierens. “The Linkage Tree Genetic Algorithm”. In: *International Conference on Parallel Problem Solving from Nature*. Ed. by R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph. 2010, pp. 264–273.

[8] P. A. N. Bosman and D. Thierens. “Linkage neighbors, optimal mixing and forced improvements in genetic algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 585–592.

[9] M. Pelikan and D. E. Goldberg. “Hierarchical Bayesian optimization algorithm”. In: *Scalable optimization via probabilistic modeling*. Springer, 2006, pp. 63–90.

[10] M. Pelikan and T.-K. Lin. “Parameter-less hierarchical BOA”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Springer. 2004, pp. 24–35.

[11] D. Thierens and P. A. N. Bosman. “Optimal mixing evolutionary algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2011, pp. 617–624.

[12] A. Bouter et al. “GPU-accelerated bi-objective treatment planning for prostate high-dose-rate brachytherapy”. In: *Medical Physics* 46.9 (2019), pp. 3776–3787.

[13] D. Barten et al. “PD-0821 Artificial Intelligence based planning of HDR prostate brachytherapy: first clinical experience.” In: *Radiotherapy and Oncology* 161 (2021), S653–S655.

[14] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. N. Bosman. “Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 1041–1048.

[15] M. Virgolin, T. Alderliesten, A. Bel, C. Witteveen, and P. A. N. Bosman. "Symbolic regression and feature construction with GP-GOMEA applied to radiotherapy dose reconstruction of childhood cancer survivors". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 1395–1402.

[16] M. Virgolin, Z. Wang, T. Alderliesten, and P. A. N. Bosman. "Machine learning for the prediction of pseudorealistic pediatric abdominal phantoms for radiation dose reconstruction". In: *Journal of Medical Imaging* 7.4 (2020), p. 046501.

[17] G. R. Harik and F. G. Lobo. "A parameter-less genetic algorithm." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vol. 99. 1999, pp. 258–267.

[18] B. W. Goldman and W. F. Punch. "Fast and efficient black-box optimization using the parameter-less population pyramid". In: *Evolutionary Computation* 23.3 (2015), pp. 451–479.

[19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. "Regularized evolution for image classifier architecture search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4780–4789.

[20] L. Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[21] R. B. Heckendorn and A. H. Wright. "Efficient linkage discovery by limited probing". In: *Evolutionary Computation* 12.4 (2004), pp. 517–545.

[22] W. Wen et al. "Neural predictor for neural architecture search". In: *European Conference on Computer Vision*. Springer. 2020, pp. 660–676.

[23] Y. Sun et al. "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor". In: *IEEE Transactions on Evolutionary Computation* 24.2 (2019), pp. 350–364.

[24] C. Wei et al. "NPENAS: neural predictor guided evolution for neural architecture search". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[25] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. "Taking the human out of the loop: a review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[26] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 507–523.

[27] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[28] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. "Hyperopt: a Python library for model selection and hyperparameter optimization". In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[29] R. Schaeffer, M. Khona, and I. Fiete. "No free lunch from deep learning in neuroscience: a case study through models of the entorhinal-hippocampal circuit". In: *bioRxiv* (2022), pp. 2022–08.

[30] Z. Liu et al. “A ConvNet for the 2020s”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11976–11986.

[31] H. Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2021, pp. 10347–10357.

[32] T. He et al. “Bag of tricks for image classification with convolutional neural networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 558–567.

[33] I. Loshchilov and F. Hutter. “Stochastic gradient descent with warm restarts”. In: *Proceedings of the 5th International Conference on Learning Representations*. 2017, pp. 1–16.

[34] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. “Revisiting unreasonable effectiveness of data in deep learning era”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 843–852.

[35] N. Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.

[36] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *Proceedings of the 3th International Conference on Learning Representations*. Ed. by Y. Bengio and Y. LeCun. 2015.

[37] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. In: *Nature methods* 18.2 (2021), pp. 203–211.

[38] T. Elsken, J. H. Metzen, and F. Hutter. “Neural architecture search: a survey”. In: *Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[39] S. Nikolov et al. “Clinically applicable segmentation of head and neck anatomy for radiotherapy: deep learning algorithm development and validation study”. In: *Journal of Medical Internet Research* 23.7 (2021), e26151.

[40] M. Frid-Adar, A. Ben-Cohen, R. Amer, and H. Greenspan. “Improving the segmentation of anatomical structures in chest radiographs using U-Net with an ImageNet pre-trained encoder”. In: *Image Analysis for Moving Organ, Breast, and Thoracic Images*. Springer, 2018, pp. 159–168.

[41] C. E. Cardenas, J. Yang, B. M. Anderson, L. E. Court, and K. B. Brock. “Advances in auto-segmentation”. In: *Seminars in radiation oncology*. Vol. 29. 3. Elsevier. 2019, pp. 185–197.

[42] S. Budd, E. C. Robinson, and B. Kainz. “A survey on active learning and human-in-the-loop deep learning for medical image analysis”. In: *Medical Image Analysis* 71 (2021), p. 102062.

[43] K. H. Zou et al. “Statistical validation of image segmentation quality based on a spatial overlap index: scientific reports”. In: *Academic Radiology* 11.2 (2004), pp. 178–189.

[44] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[45] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *Proceedings of the 3th International Conference on Learning Representations*. 2015.

[46] J. Deng et al. "ImageNet: a large-scale hierarchical image database". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 248–255.

[47] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio. "Transfusion: understanding transfer learning for medical imaging". In: *Advances in Neural Information Processing Systems* 32 (2019).

[48] A. A. Kalinin, V. I. Iglovikov, A. Rakhlin, and A. A. Shvets. "Medical image segmentation using deep neural networks with pre-trained encoders". In: *Deep learning applications*. Springer, 2020, pp. 39–52.

[49] J. Chen et al. "TransUNet: transformers make strong encoders for medical image segmentation". In: *arXiv preprint arXiv:2102.04306* (2021).

[50] A. Hatamizadeh et al. "UNETR: transformers for 3D medical image segmentation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, pp. 574–584.

[51] G. M. Villeirs et al. "Interobserver delineation variation using CT versus combined CT + MRI in intensity–modulated radiotherapy for prostate cancer". In: *Strahlentherapie und Onkologie* 181.7 (July 2005), pp. 424–430.

[52] W. Qiu et al. "User-guided segmentation of preterm neonate ventricular system from 3-D ultrasound images using convex optimization". In: *Ultrasound in Medicine & Biology* 41.2 (2015), pp. 542–556.

[53] C. Fiorino, M. Reni, A. Bolognesi, G. M. Cattaneo, and R. Calandrino. "Intra-and inter-observer variability in contouring prostate and seminal vesicles: implications for conformal treatment planning". In: *Radiotherapy and Oncology* 47.3 (1998), pp. 285–292.

[54] S. Montagne et al. "Challenge of prostate MRI segmentation on T2-weighted images: inter-observer variability and impact of prostate morphology". In: *Insights into Imaging* 12.1 (2021), p. 71.

[55] C. F. Baumgartner et al. "PHiSeg: capturing uncertainty in medical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 119–127.

[56] A. L. Simpson et al. "A large annotated medical image dataset for the development and evaluation of segmentation algorithms". In: *arXiv preprint arXiv:1902.09063* (2019).

[57] S. K. Warfield, K. H. Zou, and W. M. Wells. "Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation". In: *IEEE Transactions on Medical Imaging* 23.7 (2004), pp. 903–921.

[58] L. Zhang et al. “Disentangling human error from ground truth in segmentation of medical images”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 15750–15762.

[59] Y. Yang et al. “Uncertainty quantification in medical image segmentation with multi-decoder U-Net”. In: *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 7th International Workshop, BrainLes 2021, Held in Conjunction with MICCAI 2021, Virtual Event, September 27, 2021, Revised Selected Papers, Part II*. Springer. 2022, pp. 570–577.

[60] A. Kendall, V. Badrinarayanan, and R. Cipolla. “Bayesian SegNet: model uncertainty in deep convolutional encoder-decoder architectures for scene understanding”. In: *arXiv preprint arXiv:1511.02680* (2015).

[61] F. Wenzel, J. Snoek, D. Tran, and R. Jenatton. “Hyperparameter ensembles for robustness and uncertainty quantification”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6514–6527.

[62] R. Robinson et al. “Real-time prediction of segmentation quality”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 578–585.

[63] S. Kohl et al. “A probabilistic U-Net for segmentation of ambiguous images”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6965–6975.

[64] D. P. Kingma and M. Welling. “Auto-encoding variational Bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[65] A. Bouter, N. H. Luong, C. Witteveen, T. Alderliesten, and P. A. N. Bosman. “The multi-objective real-valued gene-pool optimal mixing evolutionary algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 537–544.

[66] S. C. Maree et al. “Evaluation of bi-objective treatment planning for high-dose-rate prostate brachytherapy—A retrospective observer study”. In: *Brachytherapy* 18.3 (2019), pp. 396–403.

[67] M. C. van der Meer et al. “Robust optimization for HDR prostate brachytherapy applied to organ reconstruction uncertainty”. In: *Physics in Medicine & Biology* 66.5 (2021), p. 055001.

2

PARAMETERLESS GENE-POOL OPTIMAL MIXING EVOLUTIONARY ALGORITHMS

When it comes to solving optimization problems with evolutionary algorithms (EAs) in a reliable and scalable manner, detecting and exploiting linkage information, i.e., dependencies between variables, can be key. In this chapter, we present the latest version of, and propose substantial enhancements to, the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA): an EA explicitly designed to estimate and exploit linkage information. We begin by performing a large-scale search over GOMEA design choices, to understand what matters most and obtain a generally best-performing version of the algorithm. Next, we introduce a novel version of GOMEA, called CGOMEA, where linkage-based variation is further improved by filtering solution mating based on conditional dependencies. We compare our latest version of GOMEA, the newly introduced CGOMEA, and another contending linkage-aware EA DSMGA-II in an extensive experimental evaluation, involving a benchmark set of 9 black-box problems that can only be solved efficiently if their inherent dependency structure is unveiled and exploited. Finally, in an attempt to make EAs more usable and resilient to parameter choices, we investigate the performance of different automatic population management schemes for GOMEA and CGOMEA, de facto *making the EAs parameterless. Our results show that a new version of GOMEA and CGOMEA significantly outperform the original GOMEA and DSMGA-II on most problems, setting a new state of the art for the field.*

The contents of this chapter are based on the following publication: **A. Dushatskiy**, M. Virgolin, A. Bouter, D. Thierens, and P. A. N. Bosman. "Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms". In: *Evolutionary Computation* (June 2023), pp. 1–28.

2.1. INTRODUCTION

Key to the success of any optimization algorithm in terms of search effectiveness and efficiency, is the ability to exploit structural features of the problem being solved. To this, Evolutionary Algorithms (EAs) are no exception. For EAs, it is predominantly the variation operators that need to be favorably configured to exploit structural features. One structural feature that is of particular importance is variable dependence. Not only does variable dependence have a direct influence on the inherent difficulty of a problem, not being able to exploit such dependency information may lead to very inefficient optimization performance. If two variables are completely independent in a problem, this problem can be solved by considering the variables separately. Conversely, if two variables are strongly dependent, joint settings of these variables need to be considered in order to find the optimal solution. In EAs, such dependencies (between the variables that are directly manipulated by the EA, i.e., the genes), are also known as *linkages*. It has been long known that groups of variables that exhibit such strong linkages need to be treated, with high probability, in a joint fashion by the variation operator in order for an EA to be an efficient solver [1, 2]. Especially in the domain of discrete variables that constitute a Cartesian search space, which is also the domain that this chapter pertains to, many EAs employ a mixing operator that exchanges parts of solutions. Ensuring this mixing operator is linkage-friendly, i.e., has a high probability of exchanging groups of genes that are highly dependent, can make the difference between obtaining efficient, i.e., low-polynomial, and inefficient, i.e., exponential, scale-up of the required running time to solve the problem [1, 2].

The relevance and importance of linkage processing are even more prominent when taking a black-box perspective on optimization. In Black-Box Optimization (BBO), there is very little to no information available on the problem being solved. Metaheuristics, among which EAs, are commonly formulated and studied in this context, with the notion of designing a powerful general problem solver in mind. Certainly, the no-free-lunch theorem assures us, that considering all possible optimization problems, no such solver exists [3]. However, a generally valid assumption can be made that the types of optimization problems we are interested in, are not completely random, but have some sort of exploitable structure. It is the exploitation of this structure that governs whether or not optimization will proceed effectively and efficiently. This then brings us back to the linkage problem, for it is assumed that the structure of the typical optimization problems we are interested in is nontrivial, i.e., its variables are not all fully independent. For this reason, we have no guarantee that a simple genetic algorithm with uniform crossover, or any static crossover operator for that matter, will effectively exploit the structure of the problem. Thus, their use comes with the risk of exponential scale-up of the required runtime on problems that are polynomial-time solvable [1]. To avoid this, linkage information needs to be exploited properly. In a BBO setting, however, such information is not readily available, and thus must be determined otherwise, using previously performed solution-quality, i.e., fitness, evaluations. This process is commonly known as *linkage learning*, which is a key concept in this chapter.

An argument can be made at this point that the added complexity and effort of performing linkage learning is superfluous because a true BBO scenario is not frequently encountered when solving real-world problems. A need for BBO may still very well

surface, however, even when efficient local search heuristics are available for a particular problem. It is well-known that combining EAs and local search is highly effective for many problems [4]. The reason for this is that by applying local search to every solution, a second search problem can be seen to exist in the space of local optima of the optimization problem. Running a local search heuristic multiple times, i.e., a random restart heuristic, then can effectively be seen as random search in the space of local optima. This space may be searched more efficiently using an EA, which can be obtained by applying local search to every solution that the EA generates. Even when we understand very well the problem is solved to the point where we can design efficient local search algorithms, the nature of the search space composed of the local optima may still be extremely hard to analyze. In that space, then, there is again a need for powerful BBO algorithms.

The linkage problem has already been identified a long time ago, and much work on tackling this problem has previously been done, often presented simultaneously with a new EA, see e.g., [5, 6]. Much of this work has been toward building more complex models that are capable of capturing problem structure in more intricate detail, up to the relatively complex task of estimating entire (factorized) probability distributions, as is done in Estimation-of-Distribution Algorithms (EDAs) [7]. Although ultimately capable of exploiting problem structure properly, the overhead involved with estimating actual probabilities surpasses the need to determine the linkage information that needs to be effectively exploited. Importantly, such overhead becomes more significant on large-scale problems causing scalability issues. This chapter focuses specifically on the linkage hurdle on the road to powerful BBO algorithms. In particular, we introduce the new version of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) that seamlessly integrates the traditionally separate operators of selection and variation in EAs in order to get the most out of available linkage information. Moreover, a generalized model of linkage information allows linkage information to be processed at more than one level, e.g. processing a hierarchy of weak and strong dependencies.

The main contribution of this chapter can be summarized as a presentation of the parameterless EA showing state-of-the-art performance in the field of discrete BBO. This chapter joins all algorithmic information from our previous work that is needed to make this chapter self-contained and represent the current state-of-the-art in the GOMEA research line.

We extend our previously published work on GOMEA by a more extensive experimental analysis on more optimization problems and larger problem sizes, testing the impact of various possible design choices such as local search operators on GOMEA, and, importantly, we propose a novel variation operator which exploits conditional dependencies between sets of variables and is called Conditional GOM (CGOM). Finally, we demonstrate the practical applicability of GOMEA by designing parameterless modifications of it. The performance of the new and the old versions of GOMEA is shown in comparison with other EAs, including, the recent version of DSMGA-II [8], and the Parameter-less Population Pyramid (P3) [9]. The obtained GOMEA modification demonstrates better performance than previously published versions of it. Moreover, CGOM further improves GOMEA performance on most considered problems.

The remainder of this chapter is organized as follows. In Section 2.2 we discuss related work. In Section 2.3 we outline the general working scheme of GOMEA and present design

options for its most important components in more detail. Also, we present GOMEA instances without the population size parameter and describe schemes to run GOMEA in a population size-free fashion. Then, we present our benchmark problems and the design of experiments in Section 2.4, followed by the results in Section 2.5. This chapter ends with a discussion in Section 2.6 and conclusions in Section 2.7.

2.2. FROM GENETIC ALGORITHMS TO EDAS AND BACK AGAIN

It was already hypothesized by John Holland himself that the simple Genetic Algorithm succeeds at optimization if it can proliferate important building blocks [10], i.e., partially defined solutions for which it holds that, when averaging over all the solutions that it is part of in a population, the fitness is better than the average fitness of the population.

The key is the proper mixing of these partial solutions, which means disrupting them as little as possible (i.e., copying them entirely from one solution to the next) and not copying other parts of the solution that they are (semi)independent of. If these important partial solutions have a large probability of being destroyed during variation, for instance by using uniform crossover, the population size that is required to find the optimum may grow exponentially with the problem size. Conversely, polynomial population size growth can be achieved if the partial solutions are properly mixed. A well-known example of this is represented by the sum of additively decomposable, non-overlapping, deceptive trap functions [11].

Since then, there has been a dedicated research line in the field of evolutionary computation to design variation operators that are capable of automatically detecting the presence of important building blocks, and of reconfiguring the way in which variation proceeds to ensure that building blocks are mixed well and disrupted as little as possible. The first family of EAs along this line was the messy genetic algorithm family [12]. Algorithms in this family allowed genes to be re-ordered by explicitly encoding their location. Although eventual algorithms were able to avoid exponential scale-up, the overhead of re-ordering genes was still substantial and lacked explanatory statistical underpinning.

For this reason, researchers started looking into probabilistic approaches that were capable of explicitly computing dependencies between problem variables by estimating probability distributions over them. The population can be seen as a database that represents the type of solutions that are desired, and, over time, through selection, gets pushed toward the optimum. Selecting the better solutions makes dependencies stand out, since on average, the solutions that contain important building blocks will have a better fitness than those solutions that do not. By estimating a probability distribution from the population, these dependencies can be explicitly modeled in a probabilistic fashion. Moreover, by sampling new solutions from the estimated distribution, these dependencies are respected. Because the process of sampling generates a new database that has the same statistical properties as the original database (to the extent to which these properties were modeled in the probability distribution), this approach can be considered as mixing solutions at a population level rather than at the two-parent level as was before typically reminiscent of genetic algorithms. This type of algorithm is known as the Estimation-of-Distribution Algorithm (EDA) [7, 13]. Effectively and efficiently estimating probability distributions that capture higher-order dependencies was still key however to avoid exponential scale-up on problems with non-trivial dependency

structure. Initial attempts that used either univariately factorized probability distributions (e.g., PBIL [14], and cGA [15]) that modeled every variable to be independent of every other variable, or bivariately factorized distributions that considered variable dependencies of at most order two (e.g., MIMIC [16], COMIT [17], and BMDA [18]) still fail to obtain polynomial scale-up on the additively decomposable deceptive trap functions. Low-order polynomial scale-up is only obtained by EDAs that model higher-order dependencies (e.g., ECGA [19], LFDA [20], and BOA [2]).

The most advanced EDA in this line is commonly accepted to be the hierarchical Bayesian Optimization Algorithm (hBOA) [5]. Both its predecessor BOA and hBOA itself estimate a Bayesian network every generation using a greedy learning procedure. hBOA, however, has the ability to store the parameters in this network more efficiently by storing only those combinations of values for dependent variables that actually appear in the population, thereby preventing the need to generate huge probability tables that require the explicit enumeration of *all* possible value combinations of a set of dependent variables. This, combined with a mechanism (restricted tournament selection) that promotes population diversity, allowed hBOA to be the only EDA capable of solving problems with hierarchical dependency structures while requiring only low-order polynomial-time scale-up of the population size and number of function evaluations. Although to a large extent now satisfactorily solving the linkage problem and providing a solid, statistically sound basis for doing so, the overhead required by hBOA is still substantial, requiring asymptotically $\mathcal{O}(n\ell^3)$ time per generation where ℓ is the number of problem variables and n the population size. Moreover, the number of generations required to solve a problem is typically in the same order as a properly configured GA requires, which is typically in the order of $\Theta(\sqrt{\ell})$ [21], the proofs for different EDAs, e.g., UMDA [22] are provided in [23].

Although a solid approach to tackling the linkage problem, estimating entire probability distributions comes with the necessity to estimate not only a dependency structure but also to estimate parameters (e.g., actual probabilities). Moreover, in order to decide what underlying dependency structure is a good one, i.e., not missing key dependencies and not overly complex, quality-of-fit measures need to be computed that decide when to stop the greedy learning approach that iteratively increases the complexity of the underlying dependency structure. These aspects are not necessarily important for tackling the linkage problem, because for that it would suffice to know which variables are (strongly) dependent on which other variables. The joint probabilities of entire building blocks do not need to be computed explicitly, as they are stored implicitly in the population. Mixing the information stored in the population therefore automatically follows these probabilities. These foundations form the basis of the GOMEA framework. GOMEA was first introduced in 2011 [6], posed as a broadened scope of the idea behind the original Linkage Tree Genetic Algorithm (LTGA) introduced in 2010 [24]. LTGA was one of the first algorithms to depart from the EDA principle of estimating entire probability distributions, and thus essentially going back to the notion of genetic algorithm, but still using similar statistical concepts as used in EDAs to detect dependencies. Ultimately this lead to a model-building complexity of an order of magnitude faster ($\mathcal{O}(n\ell^2)$) than hBOA, while being able to capture and exploit both low-order dependencies as well as high-order dependencies at the same time. Moreover, LTGA requires only a handful of generations to

find the optimal solution due to much more extensive model exploitation during the variation, further reducing the overall required model-building complexity. As later versions of LTGA, including the one presented in this chapter, are seen as instances of the GOMEA framework, details will be described in subsequent sections. Besides LTGA, which we will from now refer to as LT-GOMEA, other non-EDA algorithms that build models to model and exploit linkage information have been proposed lately [25, 8, 9]. These algorithms that can more generally be described as model-based EAs have also been successful at outperforming hBOA.

2.3. THE GOMEA FAMILY OF EVOLUTIONARY ALGORITHMS

The family of Gene-pool Optimal Mixing Evolutionary Algorithms (GOMEAs) has been proven to show impressive performance on benchmarks and, importantly, real-world problems. For instance, Real-Valued Multi-Objective GOMEA (RV-MO-GOMEA) [26] is now used for brachytherapy treatment planning optimization. This application received a Silver Humies award [27] which highlights its practical value and outstanding, better-than-human performance. Another example is the adaptation of GOMEA for Genetic Programming (GP-GOMEA) [28]. Beside showing better performance than alternative GP algorithms on classical machine learning benchmarks, GP-GOMEA has been also successfully applied to a real-world medical problem, namely, a radiotherapy dose reconstruction [29]. This application was noted with a Silver Humies award in 2021. These two examples show the potential of the GOMEA family of algorithms.

The family of GOMEAs is actually a subset of the OMEA family [6]. Another subset is the Recombinative OMEA (ROMEA) [24] family, whereby the mixing of solutions occurs only between 2 parent solutions rather than between all solutions in the population as is the case for GOMEA. When tested on various problems, however, GOMEA was found to have the best performance as long as the models capturing linkage information were adequate [6]. For this reason, we focus particularly on GOMEA here. The graphical overview of the GOMEA family of algorithms is shown in Figure 2.3.1.

The main idea behind the OMEA framework is that linkages are identified using sets of variable indices (see Section 2.3.1), which we shall call linkage sets. These individual linkage sets are then explicitly exploited, in contrast to classical GAs and EDAs. In the latter, entire solutions are generated and subsequently evaluated. The main idea of OMEA however is to take values only for a linkage subset from a donor solution and try these values out in another solution to see if it improves. It is this direct notion of acceptance that makes the success of the mixing operation independent of the effect of all other mixing events that may happen when constructing an entire new solution first. Because this makes each mixing event an optimal decision, unhampered by potential collateral noise, and because when all linkage sets are correctly identified, mixing essentially does not make any mistakes this way (unless unhelpful donor solutions are selected), this approach to variation was called Optimal Mixing (OM).

GOMEAs are a subclass of EAs and as such are a form of population-based search. The most traditional approach to population management is to have a population of a fixed size. We will discuss what GOMEA looks like with this approach, as well as with different approaches to population management that no longer require the specification of a value

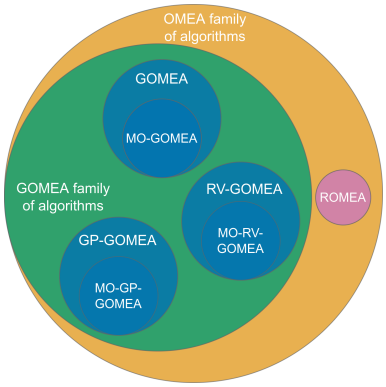


Figure 2.3.1. The set diagram of the key algorithms in the OMEA family of algorithms. ROMEA refers to Recombinative OMEA; (MO-)GOMEA refers to (Multi-Objective) discrete GOMEA; (MO-)RV-GOMEA refers to (Multi-Objective) Real-Valued GOMEA; (MO-)GP-GOMEA refers to (Multi-Objective) GOMEA for Genetic Programming.

for the population size parameter. The latter is especially of high practical value. In the remainder of this section, we provide more details on the various components of GOMEA.

2.3.1. FAMILY OF SUBSETS (FOS) AS A LINKAGE MODEL

The GOMEA class of EAs focuses on modeling linkage by explicitly identifying sets of variables to be treated jointly in the variation process. Moreover, such linkage sets are allowed to overlap. Specifically, *any* subset of the set of all variables may be identified within the linkage model. This may be defined as follows. Let $\mathscr{L} = \{0, 1, \ldots, \ell - 1\}$ be the set of ℓ unique identifiers of variables that the EA processes, then the linkage model in GOMEA is a subset of the powerset of $\mathscr{L}$. Such a set is commonly called a family-of-subsets in mathematics. We therefore call the linkage model in GOMEA the family-of-subsets, or FOS, model, and denote it by $\mathscr{F}$, i.e.,:

$$\mathscr{F} \subseteq \wp(\mathscr{L}).$$

LINKAGE TREE (LT) MODEL

Though different ways to configure a FOS model by learning linkage from the the population were introduced, we focus here on a so-called Linkage Tree (LT) model which demonstrated efficiency in solving various combinatorial optimization problems [24]. An LT is a binary tree with $2\ell - 1$ vertices. LT leaves are singletons of problem variables, the root of an LT is the set of all problem variables L, and all other vertices are variables subsets F^i which are unions of disjoint subsets of children k, j of vertex i: $F^i = F^j \cup F^k, F^j \cap F^k = \emptyset$.

SIMILARITY MEASURES

An LT can be built in a bottom-up fashion using hierarchical clustering [30]: starting from singletons, the most similar subsets of variables are merged until a subset containing all variables is obtained (a tree root). A similarity between two subsets of variables F^i,

F^j is defined as the average similarity measure of all pairs of variables (X, Y) where $X \in F^i, Y \in F^j$. Different similarity measures can be used [31]. Here, we consider two of them which are most commonly used (e.g., in [32, 33, 9]), namely, standard Mutual Information (MI) and Normalized Mutual Information (NMI). For two variables, X, Y MI and NMI are defined as:

$$\mathrm{MI}(X,Y) = \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y).$$

$$\mathrm{NMI}(X,Y) = \mathrm{MI}(X,Y)/\mathrm{H}(X,Y).$$

where $H(X)$ is information entropy, defined as

$$H(X) = \sum_{x \in \Omega_X} -P(X=x)\log(P(X=x)),$$

where Ω_X is the set of X values.

LINKAGE TREE FILTERING

It was shown in [34] that a full LT model (with $2\ell - 1$ vertices) may have redundant subsets which can be filtered out to increase mixing efficiency. Here we consider one particular case of filtering which was successfully applied in [9]. When two subsets F^j and F^k are merged into a subset F^i, it may happen, that the similarity between them is maximal (one in case of MI or NMI), which means that in a population, values of variables from one subset can perfectly predict values of variables from another subset. We suppose that there is no merit in using these subsets in mixing separately, as it may disrupt this pattern and use additional unnecessary evaluations. Thus, keeping subsets F^j and F^k in a FOS is not reasonable, and it is sufficient to keep only the parent subset F^i. In practice, to deal with possible numerical errors in similarity measure calculation, the filtering rule is invoked if the similarity measure value is above $1-\varepsilon$ threshold (we use $\varepsilon = 10^{-6}$). Let $S(X,Y)$ be the similarity measure. After the filtering rule is applied, the subsets of an LT model satisfy the description:

$$\begin{aligned} &\forall F^i, F^j, F^k \in \mathscr{F} \text{ s.t. } F^i = F^j \cup F^k, \\ &S(F^j, F^k) \leq 1 - \varepsilon. \end{aligned} \tag{2.1}$$

2.3.2. GENE-POOL OPTIMAL MIXING (GOM)

Variation in GOMEA is guided by the contents of the FOS model, in order to prevent disrupting the linkage information it represents. To do so, an operator called *Gene-pool Optimal Mixing (GOM)* is used that integrates selection and variation and has many similarities with greedy search algorithms. The GOM operator is described in pseudo-code in Algorithm 2.3.1.

GOM is applied to a single solution and outputs a single solution that is never worse than the input solution. To improve a solution, GOM loops over the contents of the FOS model. We consider two ways of iterating over FOS elements: in random order [24] and ascending order of subsets size ($|F^i|$) [9]. For each linkage subset F^i, GOM attempts to overwrite the values of the variables in F^i of the solution in consideration, with values from a donor solution that is chosen at random from the population. If this overwriting

action does not cause the fitness of the solution to become worse, the copy action is accepted. Otherwise, the donor material is rejected and the action is undone. To allow traversing of fitness plateaus, changes that lead to the same fitness are also accepted if the solution is not the elitist one (having the best fitness among all so far evaluated solutions). Note that a FOS subset containing all variables, i.e., the root of the LT model, is not used in GOM, as it implies replacing an entire solution rather than changing only a part of it.

EXHAUSTIVE DONOR SEARCH (EDS)

When population diversity becomes low, it is likely that a randomly selected donor has the same genes F^i as the current solution, therefore, no new genotype is obtained. To deal with this situation, we can continue trying different donors until one is found in which genes F^i are different from the current solution is found. This modification is called *Exhaustive Donor Search (EDS)*, following [9].

FORCED IMPROVEMENTS (FI)

If no subset F^i leads to changes in the solution undergoing GOM, the so-called *forced improvements* (FI) phase can (optionally) start. Originally, the FI was proposed in [6] to deal with convergence issues in MAXCUT. Namely, it can happen that the population starts to drift in fitness plateaus, i.e., solutions keep changing without improving. This lack of convergence makes it unlikely for further improvements to be discovered. Therefore, FI is specifically designed to steer the search towards converging to the elitist solution. Note that for simplicity only one elitist (i.e., best) solution is stored if there are multiple solutions with equally good fitness values. The FI phase works like the normal GOM phase, except for the fact that the donor solution is always set to be the elitist solution. Moreover, to further ensure convergence, changes that lead to equal fitness are now rejected (one can no longer drift in fitness plateaus). Only if the solution strictly improves in fitness, the overwrite action is accepted. To prevent the FI phase to reduce diversity too fast, the FI phase is stopped as soon as an improvement happens. Finally, if a solution could not be improved in the FI phase, it is overwritten by the elitist solution. This action decreases diversity in the population, but on the other hand might improve convergence.

2.3.3. CONDITIONAL GENE-POOL OPTIMAL MIXING (CGOM)

By design, the GOM operator copies genes from a donor solution independently for each FOS element. Therefore, dependencies between FOS elements are not taken into account, i.e., when GOM is applied to a FOS element, any (weak) dependencies of variables inside the FOS element to variables outside the FOS element are not considered which might lead to suboptimal linkage usage because it may well be that although interactions between variables are of low order, they may still not be defined in terms of mutually exclusive subsets. I.e., consider the NK-landscapes [35] with random subsets of variables for the subfunctions To alleviate this limitation, we consider a new gene-pool optimal mixing operator - the *Conditional GOM (CGOM)*. CGOM is closely related and inspired by recently introduced conditional linkage models for the real-valued GOMEA (RV-GOMEA) [36]. However, in RV-GOMEA conditional dependencies were not considered together with a hierarchical model like the LT which we do have for the first time.

CGOM works similarly to GOM but takes into account what gene values are being processed to choose suitable donor solutions. If the variables contained in a FOS subset

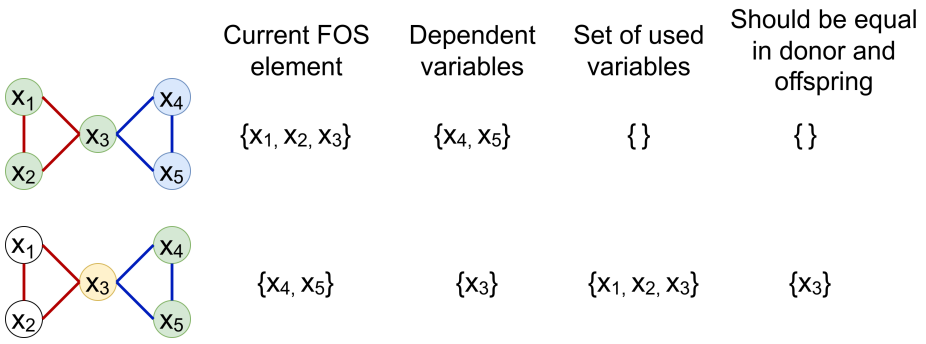


Figure 2.3.2. A minimal CGOM working example. Fitness function f is a function of 5 variables and can be decomposed into two subfunctions: $f(x_1,x_2,x_3,x_4,x_5) = f_1(x_1,x_2,x_3) + f_2(x_3,x_4,x_5)$. This is shown by the different colors of the edges. Suppose that after filtering, the LT FOS contains two elements: $\{x_1,x_2,x_3\}$ and $\{x_4,x_5\}$. Variables from the current FOS element are colored in green; variables that are dependent on it are colored in blue; variables that are dependent on it and are already used (i.e., are taken into account by CGOM) are colored in yellow. Green-colored genes are pooled to offspring from only those donors which have yellow-colored genes equal to what is found in the current offspring.

F^i are weakly dependent on variables not in F^i, CGOM takes this into account during mixing. Specifically, each FOS subset can be made conditionally dependent on a group of other variables under the condition that they were already used in the current iteration of GOM.

Suppose some genes have already been considered during mixing, i.e., for the current application of GOM to a given solution, these variables have been subjected to GOM before (they were in a FOS element considered earlier). We store these genes in a set U. When a new FOS element F^i is considered, we compute (explained below) the set of variables V s.t. 1) $V \cap F^i = \emptyset$, 2) all variables in V depend on the variables in F^i (we refer to such set of variables as G_i), and 3) $V \subseteq U$ (i.e., they were considered before). Since variables from V and F^i (weakly) depend on each other, we enforce that selecting which gene configuration for F^i is considered should be conditioned on V. This is achieved by considering as donor solutions only those which have the same genes for variables in V as the current solution undergoing CGOM has.

A minimal CGOM working example is shown in Figure 2.3.2. The CGOM differences as compared to GOM in terms of pseudocode are highlighted in Algorithm 2.3.1.

In the BBO paradigm, we have no a priori information on the dependence structure between variables. However, similar to FOS learning, we can estimate a notion of variable dependence based on the state of the population and the similarity measure (e.g., MI or NMI). Broadly speaking, we say that a FOS element F^i is dependent on a variable X ($X \notin F^i$) if the average pairwise similarity measure $S(x,y)$ between the variables in F^i and X ($\frac{1}{|F^i|}\sum_{y \in F^i} S(X,y)$) is relatively large compared to the average similarity measure

between the variables only in F^i on the one hand and all the variables that do not belong to F^i on the other hand, i.e. all measures $\frac{1}{|F^i|}\sum_{y\in F^i} S(x,y)$ for $x \notin F^i$ [1].

Particularly, we use a threshold to detect such dependencies: a FOS element F^i is considered to have a dependency with variable j if the average pairwise similarity measure between j and variable in F^i is greater than λM where M is the largest average pairwise similarity score between variables from F^i and variables not belonging to F^i. This dependencies learning procedure is described in pseudocode in the function *learnDependencies* of Algorithm 2.3.3. The Hyperparameter λ is tunable, its range of possible values is $[0,1]$. The smaller the value of λ, the larger the number of estimated dependencies. In other words, small λ values will result in high recall (we are unlikely to miss dependencies but might have many false positives), while large λ values will improve precision (we might miss many dependencies but will have few small positives).

2.3.4. GOMEA WITH A TRADITIONAL, SINGLE POPULATION

The initial population of n solutions is initialized randomly. After random solutions are generated, a local search algorithm can be applied to efficiently move them to a local optimum.

LOCAL SEARCH

We consider two local search algorithms here: simple *Single-Iteration Hill Climber* (*SIHC*) and *Exhaustive Hill Climber* (*EHC*). SIHC (also called *first-improvement local search* in literature [37] works by flipping bits of a solution in random order and greedily accepting improving changes. EHC (also called *best-improvement local search* [37]) is SIHC repeated multiple times until no improvements are found in a single bit flipping iteration over all variables. Both hill climber variants were shown to be efficient components of advanced EAs, for instance, SIHC was used in [25], and EHC was used in [9]. The pseudocode for considered Hill Climber algorithms is listed in Algorithm 2.3.2.

TOURNAMENT SELECTION

Each iteration of the main GOMEA loop starts with linkage model learning. GOMEA does not have a traditional selection phase because GOM already induces selection, by discarding changes that are detrimental to a solution's fitness. However, we consider the option of using tournament selection to select good solutions upon which to learn the linkage model, as done in [25, 8]. We remark that with this option, the selection is disregarded after the linkage model is learned, i.e., it is not used to override the population.

After the linkage model is learned, the GOM variation operator is applied to every solution in the population to generate n offspring solutions. The population is then completely replaced by the offspring solutions. This main loop runs until the termination criterion is satisfied, which is naturally triggered when the population converges (i.e., all solutions have equal genotypes), but can also include other termination conditions

[1] Minimal/Maximal pairwise similarity metrics between variables in F^i and a variable x not belonging to it (e.g., $\max_{y\in F^i} S(x,y)$) are very unstable (e.g., prone to just one outlier value), and in many cases would be just 1 (in case of taking maximum) or 0 in case of taking minimum). Also, then it becomes difficult to rank different variables as many would have the same value of such metric. The average value is intuitive, easy to use and less sensitive to outliers.

Algorithm 2.3.1: GOM and the proposed CGOM operators. Lines in green font color are used in CGOM only.

Function (C)GOM($s, \mathscr{P}, \mathscr{F}, \mathscr{G}$, *useEDS*, *useFI*)**:**
- **input :** current solution s, population $\mathscr{P}$ (of size n), FOS $\mathscr{F}$, dependency graph $\mathscr{G}$, hyperparameters *useEDS*, *useFI*
- **output:** evolved offspring o
- $b \leftarrow o \leftarrow s$ // b is a backup solution
- *changed* $\leftarrow$ false
- $U \leftarrow \emptyset$
- $\mathscr{F} \leftarrow$ orderFOS($\mathscr{F}$) // e.g., random permutation of FOS elements F^i
- **for** $i \in \{0,1,\ldots,|\mathscr{F}|-1\}$ **do**
  - *donorsList* = randomPermutation($\{\mathscr{P}_0, \mathscr{P}_1, \ldots, \mathscr{P}_{n-1}\}$) // P_i is the i-th individual
  - **for** $j \in \{0,1,\ldots,n-1\}$ **do**
    - $U \leftarrow U \cup F^i$
    - $d \leftarrow donorsList_j$
    - **if** ¬checkDonor($o, d, \mathscr{G}, U$) **then**
      - **continue**
    - $o_{F^i} \leftarrow d_{F^i}$
    - **if** $o_{F^i} \neq d_{F^i}$ **then**
      - evaluateAndUpdateElitist(o) // get fitness of o, update elitist if necessary
      - **if** ($o \neq elitist$ ***and*** $o.fitness \geq b.fitness$) ***or*** ($o = elitist$ ***and*** $o.fitness > b.fitness$) **then**
        - $b_{F^i} \leftarrow o_{F^i}$
        - *changed* $\leftarrow$ true
      - **else**
        - $o_{F^i} \leftarrow b_{F^i}$
      - **break**
    - **if** ¬*useEDS* **then**
      - **break**
- **if** *useFI* ***and*** (¬*changed* ***or*** $o.NIS > 1 + \log_{10}(n)$) **then**
  - FI (s, $\mathscr{F}, \mathscr{G}$, *useEDS*)
- // $o.NIS$ is a counter of non-improving GOM iterations for this solution ("No Improvement Stretches")
- **if** $o.fitness \leq s.fitness$ **then**
  - $o.NIS \leftarrow o.NIS + 1$
- **else**
  - $o.NIS \leftarrow o$
- **return** o

Function `FI`$(s, \mathscr{F}, \mathscr{G}, useEDS)$**:**
- **input :** current solution s, FOS $\mathscr{F}$, dependency graph $\mathscr{G}$, hyperparameter $useEDS$
- **output:** evolved offspring o
- $b \leftarrow o \leftarrow s$ `// b is a backup solution`
- $changed \leftarrow$ false
- $U \leftarrow \emptyset$
- $\mathscr{F} \leftarrow$ `orderFOS`$(\mathscr{F})$ `// e.g., random permutation of FOS elements` F^i
- **for** $i \in \{0, 1, \ldots, |\mathscr{F}| - 1\}$ **do**
  - $U \leftarrow U \cup F^i$
  - **if** ¬`checkDonor`$(o, elitist, \mathscr{G}, U)$ **then**
    - **continue**
  - **if** $o_{F^i} \neq elitist_{F^i}$ **then**
    - $o_{F^i} \leftarrow elitist_{F^i}$
    - `evaluateAndUpdateElitist` (o) `// get fitness of` o`, update elitist if necessary`
    - **if** $o.fitness > b.fitness$ **then**
      - $changed \leftarrow$ true
      - **break**
    - **else**
      - $o_{F^i} \leftarrow b_{F^i}$
  - **if** ¬$useEDS$ **then**
    - **break**
- **if** ¬$changed$ **then**
  - $o \leftarrow elitist$
- **return** o

Function `checkDonor`$(o, d, \mathscr{G}, U)$**:**
- **input :** offspring o, donor d, dependency graph $\mathscr{G}$, currently used varaibles set U
- **output:** decision (false/true) if the donor can be used
- `//` $\mathscr{G}(v)$ `are variables linked with` F^i
- **for** $p \in \mathscr{G}(v) \cap U$ **do**
  - **if** $o_p \neq d_p$ `//` p `in` o `and` d **then**
    - **return** false
- **return** true

Algorithm 2.3.2: Single-Iteration and Exhaustive Hill Climber algorithms.

```
Function exhaustiveHillClimber(s):
    input  : solution s
    output : solution s after applied multiple-iterations hill-climber
    do
        s, improved ← singleIterationHillClimber(s)
    while improved
    return s
Function singleIterationHillClimber(s):
    input  : solution s
    output : solution s after applied hill-climber, indicator improved (false/true) showing whether the
             solution was improved or not
    improved ← false
    f_s ← evaluate(s)
    for i ∈ randomPermutation({0,...,ℓ−1}) do
        s' ← copy(s)
        s'_i ← s'_i ⊕ 1 // flipping the i-th gene
        f_s' ← evaluate(s')
        if f_s' > f_s then
            s_i ← s'_i
            f_s ← f_s'
            improved ← true
    return s, improved
```

such as a maximum allowed runtime, a maximum number of function evaluations, or a maximum number of generations. The pseudocode of single-population GOMEA is provided in Algorithm 2.3.3.

2.3.5. GOING PARAMETERLESS: REMOVING THE NEED TO SET THE POPULATION SIZE

The population size is a crucial parameter for the success of EAs. With model-based EAs like GOMEA, this is arguably even more so because the linkage model needs sufficient samples to be learned to achieve a sufficient level of accuracy for the linkage to be reliable. However, choosing the right population size is problem-dependent, and highly non-trivial. Methods to scale the population size automatically over time are therefore extremely useful and convenient in practice. In this chapter, we consider two well-known population size-free schemes.

First, we consider the Interleaved Multistart Scheme (IMS), which was heavily inspired by the work by Harik and Lobo on parameterless GAs [38]. The IMS has been shown to be easy-to-use and can be naturally applied to almost any EA in various optimization domains ([32, 39, 28], Chapter 4).

The IMS consists of evolving multiple populations simultaneously, in an interleaved fashion. Its pseudocode with recursive implementation is listed in Algorithm 2.3.4. In the beginning, a single population is initialized, typically of a very small size (e.g., 2). After $\mathscr{M}_{IMS}$ generations, a new population is initialized that is larger, and it is advanced by one generation. This larger population will execute its next generation only after the smaller population performs $\mathscr{M}_{IMS}$ generations more. When the larger population

Algorithm 2.3.3: Single population GOMEA. Necessary modifications to use CGOM instead of GOM are shown in green font color.

Function `singlePopulationGOMEA(`*n, useHC, useEDS, useFI*`)`**:**
- **input :** population size n, hyperparameters *useHC*, *useEDS*, *useFI*
- **output:** evolved population $\mathscr{P}$
- $\mathscr{P} \leftarrow$ `createPopulation(`*n, useHC*`)`
- **while** $\neg$*terminationCriterionSatisfied* **do**
  - $\mathscr{P} \leftarrow$ `doOneGeneration(`$\mathscr{P}$`)`
- **return** $\mathscr{P}$

Function `createPopulation(`*n, useHC*`)`**:**
- **input :** population size n, hyperparameter *useHC*
- **output:** population $\mathscr{P}$
- **for** $i \in \{0,1,\ldots,n-1\}$ **do**
  - $\mathscr{P}_i \leftarrow$ `createRandomSolution()`
  - **if** *useHC* **then**
    - $\mathscr{P}_i \leftarrow$ `hillClimber(`$\mathscr{P}_i$`)` `// applying either SIHC or EHC`
  - `evaluateAndUpdateElitist(`$\mathscr{P}_i$`)`
- **return** $\mathscr{P}$

Function `doOneGeneration(`$\mathscr{P}$`)`**:**
- **input :** current population $\mathscr{P}$
- **output:** evolved population $\mathscr{P}$
- $\mathscr{F} \leftarrow$ `learnModel(`$\mathscr{P}$`)` $\setminus \{\{0,1,\ldots,\ell-1\}\}$ `// creating linkage model (FOS), the set containing all variables is omitted`
- $\mathscr{G} \leftarrow$ `learnDependencies(`$\mathscr{F},\mathscr{P}$`)`
- **for** $i \in \{0,1,\ldots,n-1\}$ **do**
  - $\mathscr{O}_i \leftarrow$ `GOM(`$\mathscr{P}_i,\mathscr{P},\mathscr{F}$, *useEDS*, *useFI*`)` `// GOM usage can be changed to CGOM`
  - $\mathscr{O}_i \leftarrow$ `CGOM(`$\mathscr{P}_i,\mathscr{P},\mathscr{F},\mathscr{G}$, *useEDS*, *useFI*`)`
- $\mathscr{P} \leftarrow \mathscr{O}$
- **return** $\mathscr{P}$

Function `learnDependencies(`$\mathscr{F},\mathscr{P}$`)`**:**
- **input :** FOS $\mathscr{F}$, population $\mathscr{P}$
- **output:** dependency graph $\mathscr{G}$
- $\mathscr{G} \leftarrow \{\emptyset\}_{i=0}^{|\mathscr{F}|-1}$
- $\mathscr{S} \leftarrow$ `calculateSimilarityMatrix(`$\mathscr{P}$`)` `// e.g., Mutual Information (pairwise)`
- **for** $i \in \{0,1,\ldots,\mathscr{F}-1\}$ **do**
  - $R \leftarrow \{0\}_{j=0}^{\ell-1}$
  - **for** $j \in \{0,1,\ldots,\ell-1\} \setminus F^i$ **do**
    - $R_j \leftarrow \frac{1}{|F^i|}\sum_{k=0}^{|F^i|-1} S_{j,F^i_k}$
  - $M \leftarrow \max(R)$
  - **for** $j \in \{0,1,\ldots,\ell-1\} \setminus F^i$ **do**
    - **if** $R_j > \lambda M > 0$ **then**
      - $\mathscr{G}^i \leftarrow \mathscr{G}^i \cup \{j\}$
- **return** $\mathscr{G}$

has performed $\mathscr{M}_{IMS}$ generations, an even larger population is initialized, and so on. Our implementation of the IMS uses an initial population of size 2, exponential growth whereby each new population is twice the size of the previous, and $\mathscr{M}_{IMS} = 4$, as in [38]. Smaller populations are terminated if they have converged, or their average fitness has become smaller than the average fitness of a larger population. This is because when a larger population has caught up with the smaller population, the latter will likely converge sooner, and can therefore be considered obsolete. Additionally, another convergence criterion such as a maximum allowed number of generations per population can be implemented.

Algorithm 2.3.4: Interleaved Multistart Scheme (IMS).

Function `IMS(`*useHC, useEDS, useFI*`)`**:**
- **input :** hyperparameters *useHC*, *useEDS*, *useFI*
- **output:** multiple evolved populations *Populations*
- *Populations* ← []
- **while** ¬*terminationCriterionSatisfied* **do**
  - $\mathscr{P}$ ← `createPopulation(`$2^{|Populations|+1}$`)`
  - *Populations*.append($\mathscr{P}$)
  - `generationalStep(`*Populations*`)`
- **return** *Populations*

Function `checkTermination(`*Populations, i*`)`**:**
- **input :** populations *Populations*, indices *first*, *last* of populations subject to generational step
- **output:** decision (false/true) of whether the i-th population ($Populations^i$) should be terminated
- **if** `converged(`$Populations^i$`)` **then**
  - **return** true
- **for** $j \in \{i+1, \ldots, |Populations| - 1\}$ **do**
  - **if** `averageFitness(`$Populations^j$`)` > `averageFitness(`$Populations^i$`)` **then**
    - **return** true;
- **return** false;

Function `generationalStep(`*Populations, first, last*`)`**:**
- **input :** populations *Populations*, index i of population subject to termination check
- **output:** decision (false/true) of whether the i-th population should be terminated
- $\mathscr{M}_{IMS} \leftarrow 4$ `// number of generations to do, 4 is the default value though in principle it is a hyperparameter`
- **for** *iter* ← $\{0, 1, \ldots, \mathscr{M}_{IMS} - 1\}$ **do**
  - **for** $i \in \{first, \ldots, last\}$ **do**
    - **if** ¬$Populations^i.terminated$ **then**
      - $Populations^i.terminated$ ← `checkTermination(`*Populations, i*`)`
    - **if** ¬$Populations^i.terminated$ **then**
      - `doOneGeneration(`$Populations^i$`)`
  - **for** $i \in \{first, \ldots, last - 1\}$ **do**
    - `generationalStep(`*Populations, first, i*`)`

The other schemes that we consider are the Parameter-less Population Pyramid (P3) [9], and its further modification Multiple Insertion Pyramid (P3-MI) [33]. The difference between the two is explained below and pseudocode is provided in Algorithm 2.3.5.

P3-MI arranges the population into a pyramidal structure, whereby each level of the pyramid is a set of solutions (duplicates are not stored). When a new population is created and, optionally, a local search is applied, all solutions are added to the bottom level of the pyramid. Then, by using solutions from the current pyramid level as donors, offspring of the current population are generated. Solutions that are improved by variation are promoted and entered into one level higher in the pyramid. If there is no next level in the pyramid, a new one is created. This process continues until the pyramid's top level is reached or no solutions are improved during a generation. In every generation, Linkage models are learned for each pyramid layer independently. Sizes of populations are determined by a population *growth function*. The growth function takes the iteration number as input and produces the population size (i.e., the number of solutions added to the bottom level of the pyramid). In [9] different population growth functions were studied. In this chapter, we use a quadratic function (t^2, where t is the iteration, starting from 1) as a trade-off between speed and number of function evaluations.

The P3 scheme is a special case of P3-MI with a constant growth function with a value 1, in other words, one new solution is created and evolved in each iteration.

Algorithm 2.3.5: P3-MI population management scheme. P3 is a special of case P3-MI when population growth function is constant and has value 1 for all iterations.

```
Function P3MI(useHC, useEDS, useFI):
    input  : hyperparameters useHC, useEDS, useFI
    output : evolved solutions stored in Pyramid (multi-level structure containing sets of solutions)
    iter ← 0
    Pyramid ← [∅]
    while ¬terminationCriterionSatisfied do
        n ← growthFunction(iter) // how many solutions added at this iteration
            (always one for P3)
        P ← createPopulation(n, useHC)
        Pyramid^0 ← Pyramid^0 ∪ P // add new solutions to level zero
        solutionsAdded ← true
        currentTopLevel ← |Pyramid| − 1
        L ← 0
        while L ≤ currentTopLevel and solutionsAdded do
            F ← learnModel(Pyramid^L) // learn FOS
            F ← F \ {{0, 1, ..., ℓ − 1}}
            for i ∈ {0, 1, ..., n − 1} do
                O_i ← (C)GOM(P_i, Pyramid^L)
                if O_i.fitness > P_i.fitness then
                    if L = currentTopLevel then
                        Pyramid^(L+1).append(∅) // initialize new level if necessary
                    Pyramid^(L+1) ← Pyramid^(L+1) ∪ {O_i} // add solution to the next level
                    solutionsAdded ← true
            L ← L + 1
        iter ← iter + 1
    return Pyramid
```

2.4. EXPERIMENTS

2.4.1. BENCHMARK PROBLEMS

We consider various combinatorial optimization problems that are commonly considered to be particularly interesting for benchmarking GAs.

CONCATENATED DECEPTIVE TRAPS

The concatenated deceptive trap is a well-known benchmark problem that was introduced to show that with disrupting building blocks, it takes exponentially growing resources to solve this problem. The fitness function of this problem is defined as:

$$f_{Trap_K^s}(x) = \sum_{i \in \{0,s,2s,\ldots\}, i<\ell} f^{sub}_{TrapK}\left(\sum_{j=0}^{k-1} x_{(i+j)\%\ell}\right),$$

$$f^{sub}_{TrapK}(u) = \begin{cases} k & \text{if } u = k, \\ k-1-u & \text{otherwise.} \end{cases}$$

Particularly, we consider trap functions with subfunctions size $k = 5$ and two different values of subfunctions overlap: separable traps with $s = 5$ (further referred to as $Trap_5^5$) and overlapping traps with $s = 4$ ($Trap_5^4$).

BIMODAL SEPARABLE DECEPTIVE TRAP

The bimodal symmetric concatenated trap functions [40] are interesting because, in contrast to the standard concatenated trap described above, each subfunction has two modes. We consider bimodal symmetric traps of size 6, such that the non-overlapping subfunctions are given by

$$f^{sub}_{BimodalTrapK}(u) = \begin{cases} 6 & \text{if } u = 0 \text{ or } u = 6, \\ 0 & \text{if } u = 1 \text{ or } u = 5, \\ 2 & \text{if } u = 2 \text{ or } u = 4, \\ 5 & \text{otherwise.} \end{cases}$$

NK-LANDSCAPES

The NK-landscapes with maximum overlap (also called NK-S1 landscapes) [35] with subfunctions of size $k = 5$ are interesting because of overlapping subfunctions which are different depending on the position in genotype:

$$f_{NK}(x) = \sum_{i=0}^{l-k} f^{sub}_{NK}(x_{(i,i+1\ldots,i+k)}),$$

where the values of f^{sub}_{NK} are tabular values, sampled from the uniform distribution in $[0;1]$ interval independently for different subfunctions positions.

HIERARCHIAL-IF-AND-ONLY-IF (HIFF)

The Hierarchical If-And-Only-If (HIFF) function is interesting because it includes hierarchically ordered dependencies of exponentially growing sizes that overlap:

$$f_{Hiff}(x) = \sum_{k \in \{1,2,4,\ldots,\frac{l}{2},l\}} \sum_{i=0}^{l/k-1} f_{Hiff}^{sub}(x_{ik\ldots(i+1)k-1}),$$

$$f_{Hiff}^{sub}(u) = \begin{cases} 1 & \text{if } \sum_{j=0}^{k-1} u_j = k \text{ or } \sum_{j=0}^{k-1} u_j = 0. \\ 0 & \text{otherwise} \end{cases}$$

MAXCUT

We consider MAXCUT as a well-known combinatorial optimization problem. Given a weighted undirected graph (V, E) the goal is to find a partition of the vertices in two sets such that the sum of weights of edges running between vertices in different partitions is maximized. The fitness function is therefore defined as:

$$f_{MAXCUT}(x) = \sum_{(i,j) \in E : x_i \neq x_j} w_{ij},$$

where w_{ij} is the weight of edge (i, j), x_i, x_j are solution values in the corresponding positions, i.e., each x_i is associated with one node in the graph and set to either 0 or 1 depending on which set it is assigned to.

We consider two types of MAXCUT instances. The first type is 3D square torus graphs. Each vertex is connected to 4 neighbors, forming a torus. Edges weights are integer values from $[1,5]$ sampled uniformly. This type of instance is further referred to as MAXCUT Sparse. The second type of instance is dense graphs with randomly selected $\sqrt{\ell}$ neighbors for each vertex. This type of MAXCUT instances further referred to as MAXCUT Dense, and they are known to be NP-hard problem. For MAXCUT Dense, we use edge weights values in $[0, 1000]$ sampled uniformly.

ISING SPIN-GLASS

2D Ising spin-glass problems have often been considered in the benchmarking of EDAs and other model-based EAs. The spin-glass problem fitness function is defined as:

$$f_{spinglass}(x) = \sum_{i=0}^{\ell-1} \sum_{j=0}^{\ell-1} x_i x_j J_{ij},$$

where J_{ij} defines an interaction value between two variables, $J_{ij} \in \{-1, 1\}$. In the used spin-glass instances each variable interacts with up to 4 neighbors in a 2D grid.

MAXSAT

Finally, we consider MAXSAT problem. Particularly, we consider unweighted MAX-3SAT uniform random instances [8]. MAX-3SAT is NP-hard:

$$f_{MAXSAT}(x) = \sum_{i=0}^{m-1} (\vee_{j=0}^{p_i-1} \Gamma_{ij} x_{f_{ij}}),$$

where m is the number of subfunctions (clauses), p_i is subfunction size (in the used instances $\forall i\ p_i = 3$), f_i determines which variables are contained by the subfunction with index i, and Γ can be either an unary negation operator (turning a binary x to an opposite value) or an identity operator keeping the value of x intact. The number of clauses m in the considered instances is $\approx 4.3\ell$.

2.4.2. SIZES OF PROBLEMS

We frame our experiments in terms of scalability, i.e., we record what the effort is (in terms of time and function evaluations) for an EA to find the optimum, for growing problem dimensionality. For experiments where we traditionally adopt a single population, the maximum dimensionality we consider is set to 640 for Trap_5^5, Trap_5^4, and NK-S1, to 636 for Bimodal Trap, to 1600 for MAXCUT Sparse, to 784 for Spin-glass, to 1024 for HIFF, and to 100 for NP-hard MAXCUT Dense and MAXSAT. In experiments with automatic population sizing schemes, the maximum problem sizes are doubled for all problems except for MAXCUT Sparse and Spin-glass. For the experiments with a single population, we need to use smaller maximal dimensionalities because we included bisection to discover what the optimal population size is, but bisection quickly becomes computationally prohibitive to run for large problems.

2.4.3. FINDING THE BEST SETTINGS FOR SINGLE-POPULATION GOMEA

We summarize different possible choices of single-population GOMEA components in Table 2.4.1. Since we are interested in eliminating the need to choose parameters, we attempt to define what the best GOMEA variant is across the different benchmark problems.

Hyperparameter	Options
Forced Improvements	**on** / off
Exhaustive Donor Search	**on** / off
Hill Climber	**SIHC** / EHC / off
Linkage Tree and similarity measure	unfiltered, MI / **filtered, NMI**
FOS ordering	random / **ascending subsets size**
Tournament Selection (size 2)	**on** / off

Table 2.4.1. Considered hyperparameters of single-population GOMEA. In bold, the best settings found by the experiment are described in Section 2.4.3.

In total, there are 96 ($2^5 \cdot 3$) combinations of hyperparameters. We perform an exhaustive hyperparameter search by running all 96 GOMEA variants on a set of benchmark problems. Here, our goal is to fairly compare all GOMEA variants. In order to do so, for the largest considered size of each problem, we carry out the comparisons among configurations that all have a respective optimal population size. We estimate the optimal population size using the bisection method. The success condition in bisection is solving (i.e., achieving a global optimum) a problem instance in every of 50 consecutive runs. We do not put any hard constraints on runtime. Instead, we bound it by limiting the total number of function evaluations by 10^8 and, additionally, the total number of generations

of each population by 200 (the same value as used in [8]) to prevent convergence problems. As it might happen that the smallest population size which allows to solve a problem instance does not require the fewest function evaluations, during the bisection procedure we keep track of the population size which allows to solve a problem instance with the fewest function evaluations. If the population size reaches 10^5 solutions and a problem instance is still not solved, the optimal population search procedure is terminated.

We rank the variants of GOMEA based on the minimal number of evaluations taken to find the optimal solution for each problem. The final ranking of a variant is the average of the rankings across the problems. If a variant is not able to solve one or more problems, it is dropped from the comparison. The best obtained GOMEA version is further referred to as $GOMEA^{best}$.

2.4.4. ADDING THE CGOM OPERATOR

Once $GOMEA^{best}$ is found, we look into the effect of replacing GOM with the new CGOM operator. Since CGOM requires a detection threshold λ to be set, we run comparisons with $\lambda \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. We determine the best-performing value of λ using the same approach as in Section 2.4.3. This best-performing CGOMEA version is further referred to as $CGOMEA^{best}$.

2.4.5. BENCHMARKING ALGORITHMS USING THE OPTIMAL POPULATION SIZE

$CGOMEA^{best}$ and $GOMEA^{best}$ are compared against each other, against the best previously published version of GOMEA [41], and against the most recent single-population DSMGA-II version [8]. The optimal population sizes for all algorithms are determined using bisection.

2.4.6. FINDING THE BEST SETTINGS FOR PARAMETERLESS ALGORITHMS

Next, we find the best-performing parameterless version of GOMEA. The considered options of a parameterless scheme are IMS, P3-MI with quadratic population growth function, and P3. The scheme is seen as another tunable hyperparameter. We combine it with 96 hyperparameter combinations described in Section 2.4.3 and perform a large-scale hyperparameter search, consisting of $96 \cdot 3 = 288$ possible algorithm configurations. This best-performing parameterless GOMEA version is further referred to as $GOMEA\text{-}P3^{best}$.

Once $GOMEA\text{-}P3^{best}$ is found, we replace GOM with the new CGOM operator (with λ value which was chosen for $CGOMEA^{best}$, i.e., 0.8). This CGOMEA version is further referred to as $CGOMEA\text{-}P3^{best}$.

Additionally, we add to the experiments the original P3 algorithm (as implemented in [9]) and DSMGA-II with IMS [39]. Note that we do not test other population management schemes for DSMGA-II since, to the best of our knowledge, their integration with DSMGA-II have not been studied.

To study the practical applicability of the algorithms, we remove the limit on the number of function evaluations. Instead, in all experiments with parameterless algorithms, we set a time limit of 24 hours. This is needed to make experiments computationally

Hyperparameter	Options
Forced Improvements	on / **off**
Exhaustive Donor Search	**on** / off
Hill Climber	**SIHC** / EHC / off
Linkage Tree and similarity measure	unfiltered, MI / **filtered, NMI**
FOS ordering	**random** / ascending subsets size
Tournament Selection (size 2)	on / **off**
Population scheme	**P3** / P3-MI / IMS

Table 2.4.2. Considered hyperparameters of parameterless GOMEA. In bold, the best settings found by the experiment are described in Section 2.4.6.

feasible, as some of the considered algorithms (especially some configurations which use the P3 scheme and DSMGA-II with IMS) perform in a way that the number of function evaluations is increasing very slowly.

2.4.7. STATISTICAL TESTING

To test the statistical significance of performance differences between algorithms we use the two-step approach following [42]: first, we use Friedman test (testing that performance of multiple algorithms is different), then a post-hoc multiple-hypothesis Holm procedure (testing pairs of hypotheses that one algorithm performs better than another). The significance level is set to 0.05.

2.4.8. IMPLEMENTATION DETAILS

All GOMEA variants and the P3 algorithm are implemented in C++[2]. P3[3] and DSMGA-II[4] implementations are the ones used in their corresponding original articles, with modified fitness functions to make them identical for all conducted experiments. Compiler settings for all considered algorithms are also identical.

2.5. RESULTS

2.5.1. GOMEA DESIGN CHOICES SEARCH RESULTS

We found that the best performance of single-population GOMEA is achieved when using *Single-Iteration Hill Climber*, *Forced Improvements*, *Exhaustive Donor Search*, *Filtered Linkage Tree* which is built based on *Normalized Mutual Information*, *FOS sorted in ascending elements size order*, and *Tournament Selection with tournament size 2* applied before linkage model learning, as highlighted in Table 2.4.1.

With the best hyperparameter settings, $GOMEA^{best}$ has better performance than the previously published GOMEA version on 7 out of 9 considered problems. These results are shown in Table 2.5.2, and scalability plots are presented in Figure 2.5.2. On the MAXSAT and HIFF problems, the improvement is approximately of an order of magnitude. Two

[2] Source code is available in the following repository: https://github.com/ArkadiyD/BinaryGOMEA
[3] https://github.com/brianwgoldman/FastEfficientP3/
[4] https://github.com/tianliyu/DSMGA-II-TwoEdge

problems on which performance became worse are Bimodal Concatenated Trap and MAXCUT Dense. We notice that for the Bimodal Trap problem, all algorithms of the GOMEA family perform worse than DSMGA-II. We believe that this is due to pairwise mutual information-based dependency learning failing, not optimal mixing itself (i.e., with the right FOS, scalability is excellent). Improving performance for this type of Deceptive Traps is an interesting question for future research.

Importantly, the $GOMEA^{best}$ algorithm was able to solve all considered problems with the given constraints while DSMGA-II failed to solve the HIFF, and MAXCUT Dense problems. Therefore, we can say that $GOMEA^{best}$ is an algorithm that can tackle a larger class of non-trivial problems efficiently and it is less likely to fail to solve a problem. However, we see that on 4 out of 9 problems, the performance of DSMGA-II is better.

Statistical testing of performance differences showed that there is a difference between GOMEA, DSMGA-II, $GOMEA^{best}$ and $CGOMEA^{best}$ (p-value = 0.02). Post-hoc statistical analysis showed that $CGOMEA^{best}$ performs better than GOMEA (p-value = 0.03). P-values for all comparisons are provided in Table 2.5.3.

Though the ultimate goal of the conducted hyperparameter search is to find the best-performing combination of design choices for GOMEA, it is also interesting to analyze how these choices affect the performance individually. To do so, for each design choice, we study the aggregated performance of all algorithms which use this design choice regardless of all other options they use. These results are shown in Figure 2.5.1. The most impactful design choices are Hill Climber and Exhaustive Donor Search. Results show that for most problems, Exhaustive Donor search is beneficial and substantially improves performance.

Algorithms with Single-Iteration Hill Climber on most problems outperform the ones without it, but Exhaustive Hill Climber is, apparently, too greedy and therefore is inferior to both a more simple Hill Climber and no Hill Climber at all. This is in line with earlier reported results [43]. Using the Filtered Linkage Tree built with the Normalized Mutual Information measure slightly improves the performance on some problems from the benchmark set, though it worsens the performance on the remaining ones. We see that Forced Improvements, FOS ordering, and Tournament Selection do not have a strong effect on the performance. It is noteworthy that the effects of different design choices on the performance of GOMEA on the NK-landscapes are the opposite to their effect on the majority of other problems (e.g., Exhaustive Donor Search, Hill Climber, and Filtered LT have worse performance), which suggests that the NK-landscapes problem has some unique properties compared to the other problems in the benchmark set.

2.5.2. CGOMEA PERFORMANCE

We take the found best-performing GOMEA version ($GOMEA^{best}$) and replace GOM with CGOM. First, we analyze how the performance of CGOM-based GOMEA depends on the threshold parameter λ. Results for single-population CGOMEA with different λ values are provided in Table 2.5.1. We see that λ values between 0.6 and 0.9 provide similar performance on most problems though there are some outliers in performance (as on MAXCUT Dense problem with $\lambda = 0.7$) which are caused by the stochastic nature of the bisection procedure. Nevertheless, using the same approach as for selecting the best GOMEA version, we select $\lambda = 0.8$ as the value which provides the best average

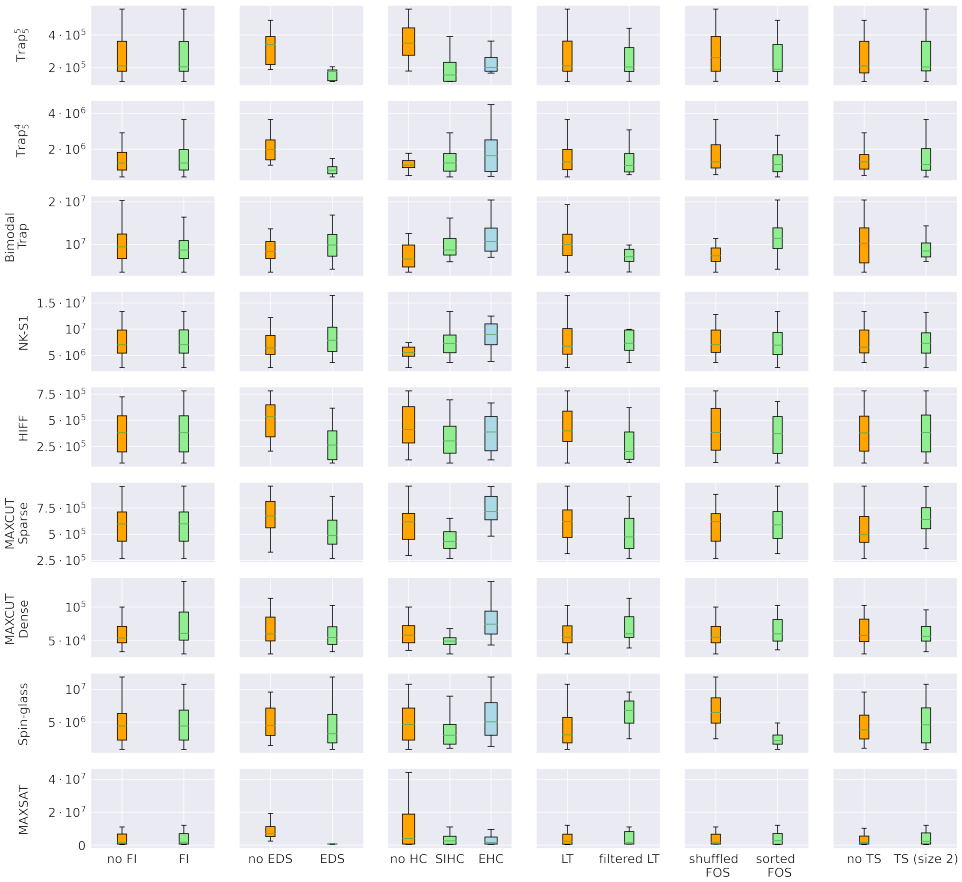


Figure 2.5.1. Effects of different design choices in GOMEA. The number of evaluations for each design choice for each problem is aggregated values of all possible GOMEA modifications with this design choice. Population sizes are found with bisection.

performance. CGOMEA with tuned λ value is further referred to as $CGOMEA^{best}$. We see that with $\lambda = 0.5$ performance deteriorates as detecting too many spurious dependencies slows down the mixing procedure. Hence, trying smaller values for λ is not necessary.

As shown in Table 2.5.2 and in Figure 2.5.2, $CGOMEA^{best}$ outperforms $GOMEA^{best}$ on 7 out of 9 considered problems. On Trap_5^5 $CGOMEA^{best}$ performs on par with $GOMEA^{best}$. Only on the HIFF problem CGOM performs slightly worse. Furthermore, $CGOMEA^{best}$ performs better than DSMGA-II on 5 problems, and there are two problems (HIFF and MAXCUT Dense) which CGOMEA managed to solve but DSMGA-II did not. Importantly, CGOMEA is still able to reliably solve all considered problems. CGOMEA's slightly inferior performance on the HIFF problem can be explained by the structure of HIFF: dependencies exist between all pairs of variables, and CGOM tends to include many variables

		λ				
Problem	ℓ	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
***Trap*$_5^5$**	640	1.16	1.16	1.16	1.16	1.16
***Trap*$_5^4$**	640	5.89	3.24	5.37	3.41	3.50
Bimodal Trap	636	120.73	106.39	100.11	99.69	104.63
NK-S1	640	31.96	28.77	34.18	37.03	46.86
HIFF	1024	1.35	1.10	1.15	1.20	0.91
MAXCUT Sparse	1600	3.84	3.90	3.27	3.35	3.46
MAXCUT Dense	100	0.62	0.53	0.62	0.37	0.58
Spin-glass	784	16.49	11.98	12.45	8.26	10.78
MAXSAT	100	4.08	4.89	3.44	2.95	3.25

Table 2.5.1. Results of single-population CGOMEA with different threshold values λ. Best population sizes are found with bisection. Ranking per problem is shown through color gradient from bright green (best, i.e., the fewest median number of function evaluations) to red (worst, i.e., the largest median number of function evaluations or problem instance not solved in all 50 runs). All results are divided by 10^5.

as dependent ones, leading to less efficient variation as the pool of appropriate donors becomes more limited.

The scalability of single-population algorithms in terms of wall-clock time required to find an optimum is shown in Figure 2.5.4. CGOMEA and GOMEA scale similarly on all problems which is better than DSMGA-II, especially on Trap$_5^5$, Bimodal Trap, NK-S1, HIFF, and MAXCUT Sparse. Only on Bimodal Trap CGOMEA is substantially slower than GOMEA, but it requires fewer function evaluations. This can be explained by the more careful donor selection done in CGOMEA. On the NP-hard MAXSAT problem scalability deviates from polynomial as expected, though on the NP-hard MAXCUT Dense problem, it is not seen for the considered problem sizes.

Problem	ℓ	**G**	**D−II**	**G^B**	**CG^B**
***Trap*$_5^5$**	640	4.02	1.30	1.16	1.16
***Trap*$_5^4$**	640	13.23	2.55	4.60	3.41
Bimodal Trap	636	34.87	3.60	114.36	99.69
NK-S1	640	62.65	66.72	51.36	37.03
HIFF	1024	7.81	N/A	0.92	1.20
MAXCUT Sparse	1600	7.31	80.03	3.87	3.35
MAXCUT Dense	100	0.49	N/A	0.51	0.37
Spin-glass	784	88.47	11.92	13.74	8.26
MAXSAT	100	69.63	3.97	5.20	2.95

ℓ	**P3**	**D−II**	**GP3^B**	**CGP3^B**
1280	2.72	31.10	1.71	1.61
1280	6.29	14.08	5.01	3.76
1278	829.32	65.03	379.01	311.83
1280	99.10	N/A	105.58	71.21
2048	5.92	N/A	4.67	6.55
1600	6.47	93.12	3.50	3.52
200	1.24	15.97	0.80	0.80
784	5.84	18.15	7.69	5.65
200	32.43	113.84	37.57	28.74

Table 2.5.2. Results of single-population (left table) and parameterless (right table) EAs. For single-population EAs, the best population sizes are found with bisection. Ranking per problem is shown through a color gradient from bright green (best, i.e., the fewest median number of function evaluations) to red (worst, i.e., the largest median number of function evaluations or problem instance not solved in all 50 runs). For MAXSAT, $\ell = 200$ results are shown for 48 instances that were solved by all algorithms. All results are divided by 10^5.
Legend: G = GOMEA; D-II = DSMGA-II; G^B = *GOMEA*^{best}; CG^B = *CGOMEA*^{best}; P3 = P3; GP3^B = *GOMEA-P3*^{best}; *CGP3*^{B} = *CGOMEA-P3*^{best}.

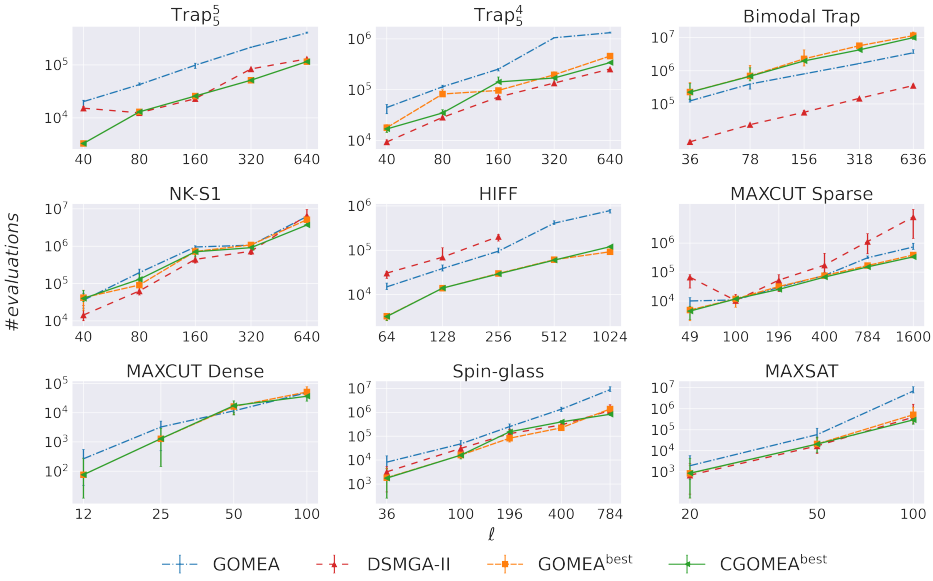


Figure 2.5.2. Scalability of single-population EAs in terms of function evaluations required to find an optimum. Points show median values of 50 runs. Bars show 3rd and 48th order statistics (92% confidence interval). If an algorithm fails to find the optimum of a problem instance in all 50 runs, the corresponding point is not shown. GOMEA refers to the previously published version [6].

2.5.3. PARAMETERLESS EAS

Results of experiments with parameterless EAs are presented in Table 2.5.2 and in scalability plots in Figure 2.5.3. We found that the best performance of a parameterless GOMEA is achieved when GOMEA uses *Single-Iteration Hill Climber, Exhaustive Donor Search, Filtered Linkage Tree* which is built based on *Normalized Mutual Information, randomly shuffled FOS*, and *P3* scheme, as highlighted in Table 2.4.2. The obtained parameterless GOMEA version is further referred to as *GOMEA-P3*^{best}. Note, that when P3 and P3-MI schemes do not use tournament selection, it makes them much more time efficient, as population statistics needed for Linkage Learning can be efficiently updated instead of re-calculated from scratch [9]. Noteworthy, crucial design choices, such as Hill Climber, Exhaustive Donor Search, and Linkage Tree type and information measure are the same in *GOMEA*^{best} and *GOMEA-P3*^{best}. Less important design choices (Forced Improvements, FOS ordering) differ, which is most likely due to the results' stochastic nature. The *GOMEA-P3*^{best} version, but with GOM replaced by CGOM ($\lambda = 0.8$ corresponding to the best value found in Section 2.4.3) is further referred to as *CGOMEA-P3*^{best}.

First, we see that DSMGA-II with IMS scheme was not capable to solve all problems in the experimental setup due to its issues with time efficiency. *CGOMEA-P3*^{best} performs better than *GOMEA-P3*^{best} on 5 problems out of 9. The most substantial differences are on Bimodal Trap, NK-S1,and Trap$_5^4$. Similar to results for single-population algorithms, CGOMEA performs worse than GOMEA on the HIFF problem, and differences between

CGOMEA and GOMEA on Trap_5^5, Trap_5^4, and MAXCUT Sparseare subtle. Compared to the P3 algorithm, [9] *CGOMEA-P3*^{best} performs better on all problems except HIFF . *GOMEA-P3*^{best} performs better than P3 on 6 problems.

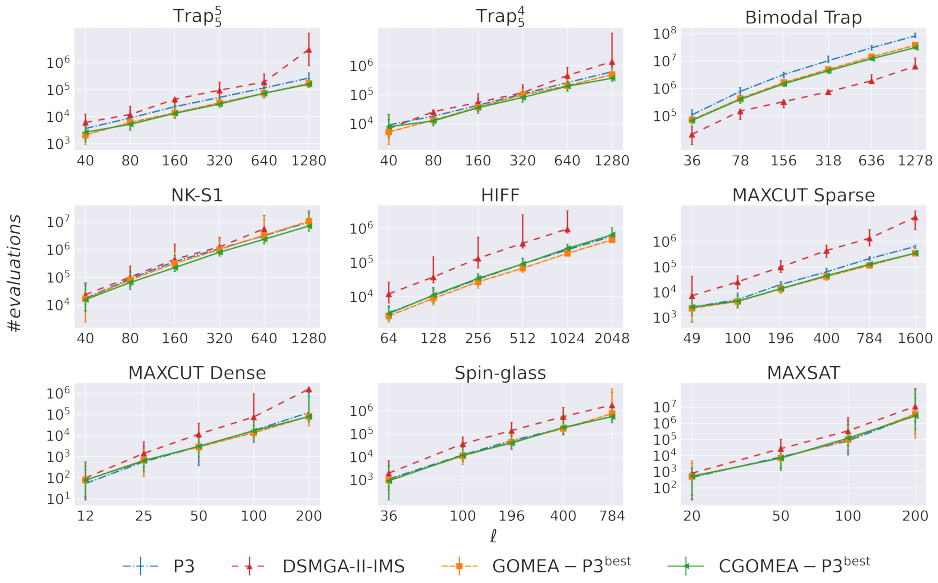


Figure 2.5.3. Scalability of parameterless EAs in terms of function evaluations required to find an optimum. Points show median values of 50 runs (48 runs for MAXSAT problem with $\ell = 200$). Bars show 3rd and 48th (46th for MAXSAT problem) order statistics (92% confidence interval). If an algorithm fails to find the optimum of a problem instance in all 50 runs (48 runs for MAXSAT problem), the corresponding point is not shown.

We note that only DSMGA-II with IMS was capable of solving all 50 instances of the MAXSAT problem of size 200 within the given time limit. *CGOMEA-P3*^{best} and P3 solved 49 problem instances, while *GOMEA-P3*^{best} solved 48. As problem instances significantly vary in complexity, we show the results for the 48 instances that were solved by all algorithms in order to provide a fair comparison.

Statistical testing of performance differences showed that there is a difference between GOMEA, DSMGA-II, *GOMEA-P3*^{best} and *CGOMEA-P3*^{best} (p-value = 0.0005). Post-hoc statistical analysis showed that *CGOMEA-P3*^{best} performs better than DSMGA-II (p-value = 0.002). All p-values are provided in Table 2.5.4.

As shown in Figure 2.5.5, P3 versions of GOMEA and CGOMEA scale similarly to P3, though they are slower. Scalability in terms of the required time to find an optimum is almost identical for *CGOMEA-P3*^{best} and *GOMEA-P3*^{best} . Both *CGOMEA-P3*^{best} and *GOMEA-P3*^{best} scale better than DSMGA-II IMS on most problems.

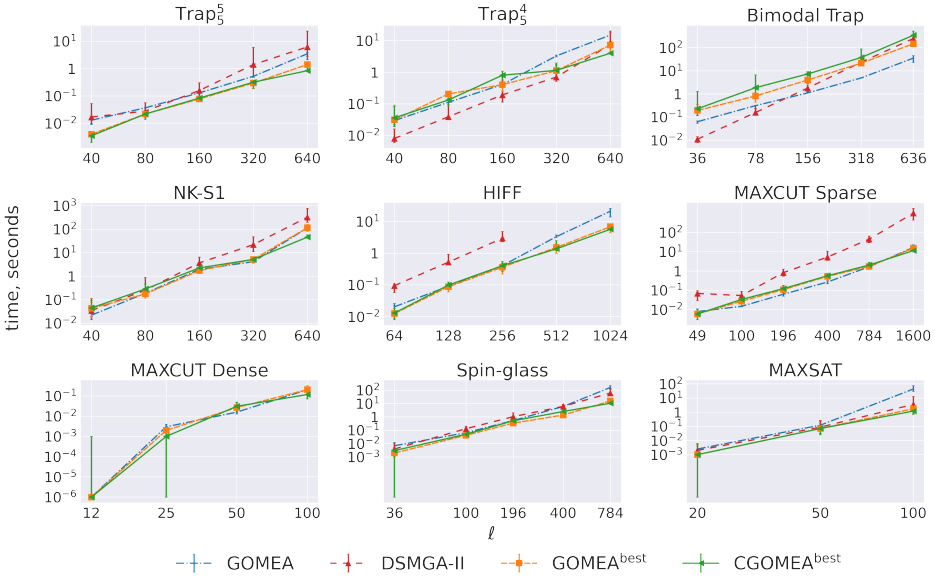


Figure 2.5.4. Scalability of single-population EAs in terms of wall clock time required to find an optimum. Points show median values of 50 runs. Bars show 3rd and 48th-order statistics (92% confidence interval). If an algorithm fails to find the optimum of a problem instance in all 50 runs, the corresponding point is not shown.

Algorithm 1	Algorithm 2	p-value
GOMEA	**CGOMEA$^{\mathbf{BEST}}$**	0.028
DSMGA−II	**CGOMEA$^{\mathbf{BEST}}$**	0.179
CGOMEA$^{\mathbf{BEST}}$	**GOMEA$^{\mathbf{BEST}}$**	0.401
GOMEA	**GOMEA$^{\mathbf{BEST}}$**	0.706
DSMGA−II	**GOMEA**	0.930
DSMGA−II	**GOMEA$^{\mathbf{BEST}}$**	0.930

Table 2.5.3. Statistical significance testing of a performance difference in pairs of single-population EAs. Reported p−values are obtained by post-hoc multiple-hypothesis Holm procedure after performing the Friedman test.

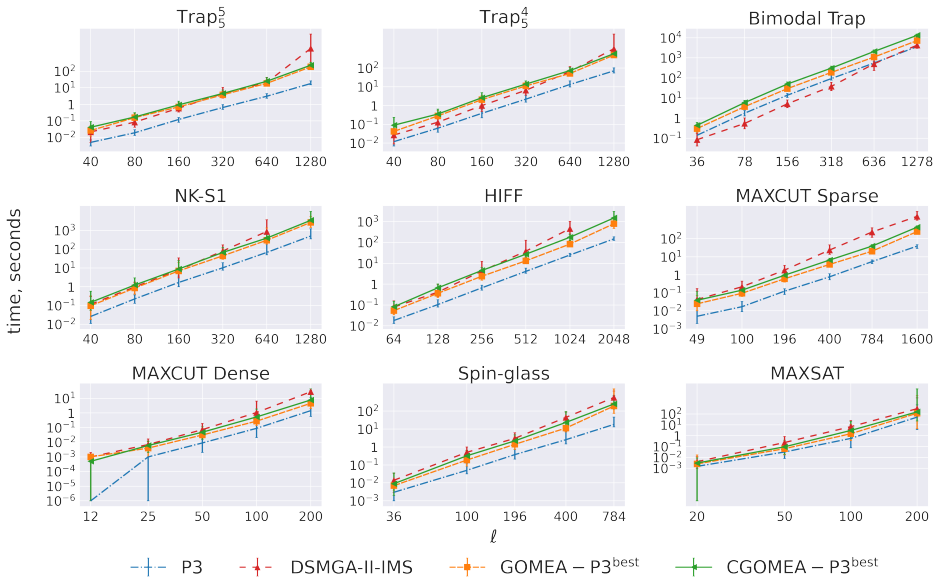


Figure 2.5.5. Scalability of parameterless EAs in terms of wall clock time required to find an optimum. Points show median values of 50 runs (48 runs for MAXSAT problem). Bars show 3rd and 48th (46th for MAXSAT problem) order statistics (92% confidence interval). If an algorithm fails to find the optimum of a problem instance in all 50 runs (48 runs for MAXSAT problem), the corresponding point is not shown.

Algorithm 1	Algorithm 2	p-value
DSMGA−II	**CGOMEA−P3^{BEST}**	0.002
DSMGA−II	**GOMEA−P3^{BEST}**	0.088
P3	**CGOMEA−P3^{BEST}**	0.178
DSMGA−II	**P3**	0.301
CGOMEA−P3^{BEST}	**GOMEA−P3^{BEST}**	0.402
P3	**GOMEA−P3^{BEST}**	0.465

Table 2.5.4. Statistical significance testing of a performance difference, the best-performing parameterless EAs. Reported $p-$values are obtained by post-hoc multiple-hypothesis Holm procedure after performing the Friedman test.

2.6. DISCUSSION

We implemented the conditional GOM operator using traditional, entropy-based similarity measures to predict dependencies between variables. Especially in early generations, this approach to detecting dependencies can be inaccurate, determining dependencies between variables that are actually independent, and missing some truly existing ones. Potentially, a more accurate approach to learning dependencies can further improve CGOM performance. Moreover, it can be interesting to apply CGOM operator in a Gray-Box Optimization (GBO) scenario when the true dependencies are known. Then, as was done for RV-GOMEA, a Bayesian network could be used rather than the conditional variant of the LT. The latter is advantageous when learning linkage in a BBO setting, but not as accurate and potentially more complex compared to direct and concise modeling of conditional dependencies. This analogy of the original concept of CGOM is to be studied in future research. However, in that case, it should be compared to different forms of EAs such as [44] which were designed specifically for GBO.

The considered model-based EAs relied on entropy-based information measures to learn dependencies between variables. We notice that the performance on Bimodal Trapcan be potentially improved if alternative linkage learning methods are used, such as fitness-based ones as it is known that alternative methods that use comparisons can find the right structure [45] (also discussed in Chapter 3). In general, the current state-of-the-art results are achieved by entropy-based linkage learning techniques, though, replacing them or combining with other methods is a promising question for future research.

In this chapter, to determine the best design choices (hyperparameters) for GOMEA and CGOMEA, we assessed performance on a standard benchmark set, and ranked algorithms based on average performance. Though this benchmark set includes well-known combinatorial optimization problems, problems arising in practical tasks may have properties (such as fitness landscape and dependencies structure) that are very different from all common benchmark functions. Though practitioners are interested in having the best-performing algorithm for their specific task, we do not have a priori knowledge of those tasks properties. Defining a good and comprehensive benchmark set is an open problem and active field of research [46]. We hypothesize however that the state-of-the-art benchmark problems in the field of EAs for binary optimization that we used is a decent compromise, in that we expect that obtaining good average performance on these problems is a good predictor of performance on many a priori unknown tasks. Moreover, in a BBO scenario matching a real-world problem with a problem from a benchmark set is a hard, if even solvable, task itself. Therefore, we did not try to specify the best possible GOMEA and CGOMEA versions for each benchmark problem but kept the focus on the best average performance.

By nature, GOM is a sequential variation procedure. However, for increasing the efficiency of (C)GOMEA it would be beneficial to use parallelization techniques to perform variation. Parallelization capability which utilizes Graphical Processing Units (GPUs) has been added to the real-valued GOMEA for the GBO case [47]. We believe that parallelizing (C)GOMEA for discrete BBO optimization is an important future work direction and can allow solving high-dimensional problems much faster.

2.7. CONCLUSION

In this chapter, we have continued the research line on the GOMEA family of algorithms with important innovations and comparisons of various ideas that have been proposed separately in the last decade since the introduction of GOMEA. First, we did an extensive hyperparameter search and obtained a version of GOMEA that showed significantly better performance than ever published before for GOMEA. Next, we introduced a new variation operator called Conditional Gene-pool Optimal Mixing (CGOM) which utilizes conditional dependencies of linkage model subsets on other variables to generate offspring solutions. GOMEA with CGOM (CGOMEA) outperformed GOMEA and DSMGA-II on most of the 9 considered diverse and non-trivial benchmark problems in a single-population EA experimental setup where we assess the scalability of required resources by the algorithms to obtain the optimum. Finally, we searched for the best-performing version of GOMEA integrated with various population size-free schemes. We found that CGOMEA with P3 scheme is a robust scalable algorithm that outperforms the competitors in terms of the number of function evaluations required to find the global optimum on almost all problems setting a new state-of-the-art performance for most of the benchmark problems and a new GOMEA variant that can serve as a new baseline in model-based evolutionary algorithms for binary search spaces for the next decade.

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO). We thank the Maurits en Anna de Kock Foundation for financing a high-performance computing system.

REFERENCES

[1] D. Thierens. “Scalability problems of simple genetic algorithms”. In: *Evolutionary Computation* 7.4 (1999), pp. 331–352.

[2] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, et al. “BOA: the Bayesian optimization algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vol. 1. Citeseer. 1999, pp. 525–532.

[3] D. H. Wolpert and W. G. Macready. “No free lunch theorems for optimization”. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82.

[4] W. E. Hart, N. Krasnogor, and J. E. Smith. “Memetic evolutionary algorithms”. In: *Recent advances in memetic algorithms*. Springer, 2005, pp. 3–27.

[5] M. Pelikan and D. E. Goldberg. “Hierarchical Bayesian optimization algorithm”. In: *Scalable optimization via probabilistic modeling*. Springer, 2006, pp. 63–90.

[6] P. A. N. Bosman and D. Thierens. “Linkage neighbors, optimal mixing and forced improvements in genetic algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 585–592.

[7] P. Larrañaga and J. A. Lozano. *Estimation of distribution algorithms: a new tool for evolutionary computation*. Vol. 2. Springer Science & Business Media, 2001.

[8] P.-L. Chen, C.-J. Peng, C.-Y. Lu, and T.-L. Yu. “Two-edge graphical linkage model for DSMGA-II”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 745–752.

[9] B. W. Goldman and W. F. Punch. “Fast and efficient black-box optimization using the parameter-less population pyramid”. In: *Evolutionary Computation* 23.3 (2015), pp. 451–479.

[10] J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[11] K. Deb and D. E. Goldberg. “Analyzing deception in trap functions”. In: *Foundations of genetic algorithms*. Vol. 2. Elsevier, 1993, pp. 93–108.

[12] H. Kargupta. “The gene expression messy genetic algorithm”. In: *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE. 1996, pp. 814–819.

[13] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea. *Towards a new evolutionary computation: advances on estimation of distribution algorithms*. Vol. 192. Springer, 2006.

[14] S. Baluja and R. Caruana. “Removing the genetics from the standard genetic algorithm”. In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 38–46.

[15] G. R. Harik, F. G. Lobo, and D. E. Goldberg. “The compact genetic algorithm”. In: *IEEE Transactions on Evolutionary Computation* 3.4 (1999), pp. 287–297.

[16] J. S. De Bonet, C. L. Isbell Jr., and P. A. Viola. “MIMIC: finding optima by estimating probability densities”. In: *Advances in Neural Information Processing Systems*. 1997, pp. 424–430.

[17] S. Baluja and S. Davies. *Combining multiple optimization runs with optimal dependency trees*. Tech. rep. Carnegie-Mellon University, Pittsburgh PA, Department of Computer Science, 1997.

[18] M. Pelikan and H. Mühlenbein. "The bivariate marginal distribution algorithm". In: *Advances in Soft Computing*. Springer, 1999, pp. 521–535.

[19] G. R. Harik, F. G. Lobo, and K. Sastry. "Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA)". In: *Scalable optimization via probabilistic modeling*. Springer, 2006, pp. 39–61.

[20] H. Mühlenbein and T. Mahnig. "FDA-A scalable evolutionary algorithm for the optimization of additively decomposed functions". In: *Evolutionary Computation* 7.4 (1999), pp. 353–376.

[21] M. Pelikan and D. E. Goldberg. "Hierarchical Bayesian optimization algorithm". In: *Scalable optimization via probabilistic modeling*. Springer, 2006, pp. 63–90.

[22] H. Mühlenbein and G. Paass. "From recombination of genes to the estimation of distributions I. Binary parameters". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1996, pp. 178–187.

[23] B. Doerr and F. Neumann. *Theory of evolutionary computation: recent developments in discrete optimization*. Springer Nature, 2019.

[24] D. Thierens and P. A. N. Bosman. "Optimal mixing evolutionary algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2011, pp. 617–624.

[25] S.-H. Hsu and T.-L. Yu. "Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: DSMGA-II". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2015, pp. 519–526.

[26] A. Bouter et al. "GPU-accelerated bi-objective treatment planning for prostate high-dose-rate brachytherapy". In: *Medical Physics* 46.9 (2019), pp. 3776–3787.

[27] E. Goodman. "Human-Competitive Results Awards - "Humies" 2019 - Announces Winners at GECCO". In: *SIGEVOlution* 12.3 (Jan. 2020), pp. 3–5.

[28] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. N. Bosman. "Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 1041–1048.

[29] M. Virgolin, Z. Wang, T. Alderliesten, and P. A. N. Bosman. "Machine learning for the prediction of pseudorealistic pediatric abdominal phantoms for radiation dose reconstruction". In: *Journal of Medical Imaging* 7.4 (2020), p. 046501.

[30] A. Kraskov and P. Grassberger. "MIC: Mutual Information based hierarchical Clustering". In: *Information theory and statistical learning*. Springer, 2009, pp. 101–123.

[31] P. A. N. Bosman and D. Thierens. "On measures to build linkage trees in LTGA". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 276–285.

[32] N. H. Luong, H. La Poutré, and P. A. N. Bosman. "Multi-objective Gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start scheme". In: *Swarm and Evolutionary Computation* 40 (2018), pp. 238–254.

[33] W. den Besten, D. Thierens, and P. A. Bosman. "The multiple insertion pyramid: a fast parameter-less population scheme". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2016, pp. 48–58.

[34] P. A. N. Bosman and D. Thierens. "More concise and robust linkage learning by filtering and combining linkage hierarchies". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2013, pp. 359–366.

[35] M. Pelikan, K. Sastry, D. E. Goldberg, M. V. Butz, and M. Hauschild. "Performance of evolutionary algorithms on NK-landscapes with nearest neighbor interactions and tunable overlap". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2009, pp. 851–858.

[36] A. Bouter, S. C. Maree, T. Alderliesten, and P. A. N. Bosman. "Leveraging conditional linkage models in gray-box optimization with the real-valued gene-pool optimal mixing evolutionary algorithm". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2020, pp. 603–611.

[37] G. Ochoa, S. Verel, and M. Tomassini. "First-improvement vs. best-improvement local optima networks of NK-landscapes". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2010, pp. 104–113.

[38] G. R. Harik and F. G. Lobo. "A parameter-less genetic algorithm." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vol. 99. 1999, pp. 258–267.

[39] Y.-J. Lin and T.-L. Yu. "Investigation of the exponential population scheme for genetic algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 975–982.

[40] K. Deb, J. Horn, and D. E. Goldberg. "Multimodal deceptive functions". In: *Complex Systems* 7.2 (1993), pp. 131–154.

[41] D. Thierens. "The linkage tree genetic algorithm". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2010, pp. 264–273.

[42] J. Derrac, S. Garcıéa, D. Molina, and F. Herrera. "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms". In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 3–18.

[43] P. A. N. Bosman and D. Thierens. "The roles of local search, model building and optimal mixing in evolutionary algorithms from a BBO perspective". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2011, pp. 663–670.

[44] F. Chicano, D. Whitley, G. Ochoa, and R. Tinós. "Optimizing one million variable NK landscapes by hybridizing deterministic recombination and local search". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 753–760.

[45] M. W. Przewozniczek and M. M. Komarnicki. "Empirical linkage learning". In: *IEEE Transactions on Evolutionary Computation* 24.6 (2020), pp. 1097–1111.

[46] K. van der Blom et al. “Towards realistic optimization benchmarks: a questionnaire on the properties of real-world problems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2020, pp. 293–294.

[47] A. Bouter and P. A. N. Bosman. “GPU-accelerated parallel gene-pool optimal mixing in a gray-box optimization setting”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 675–683.

3

A NOVEL APPROACH TO DESIGNING SURROGATE-ASSISTED GENETIC ALGORITHMS BY COMBINING EFFICIENT LEARNING OF WALSH COEFFICIENTS AND DEPENDENCIES

Surrogate-assisted evolutionary algorithms are potentially valuable for real-world optimization problems with expensive fitness evaluations. In this chapter, we focus on pseudo-Boolean functions in a black-box setting. Instead of using a surrogate model as a fitness function approximator, we propose to precisely learn the coefficients of the Walsh decomposition of a fitness function and use it as a surrogate. If the coefficients are learned correctly, the Walsh decomposition values perfectly match the fitness function, allowing us to find the optimal solution by optimizing the surrogate without additional fitness evaluations. Efficient Walsh coefficients learning is possible for pseudo-Boolean functions with $k-$bounded epistasis and known problem structure. We propose to learn dependencies between variables first to substantially reduce the number of Walsh coefficients to be calculated. Once an accurate Walsh decomposition is obtained, the surrogate model is optimized using GOMEA. We compare the proposed approach with standard GOMEA and two other Walsh decomposition-based algorithms. Our approach outperforms the alternatives and demonstrates scalability with a complexity of $\mathcal{O}(\ell \log \ell)$ function evaluations when the number of subfunctions is $\mathcal{O}(\ell)$ and all subfunctions are $k-$bounded.

The contents of this chapter are based on the following publication: **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "A novel approach to designing surrogate-assisted genetic algorithms by combining efficient learning of Walsh coefficients and dependencies". In: *ACM Transactions on Evolutionary Learning and Optimization* 1.2 (2021), pp. 1–23.

3.1. INTRODUCTION

A principled way to improve the efficiency and effectiveness of Evolutionary Algorithms (EAs) is to explicitly use (learnable) models. Such models can represent characteristics of the problem being solved, which can then be leveraged during optimization. If the model class allows capturing and efficiently learning the right characteristics for a certain problem, search can be very efficient on that problem. Different types of models exist, from those that help guide variation operators to those that model the fitness landscape. The latter models are often called the surrogate models [1], which we predominantly focus on here. We furthermore consider use of linkage models.

Linkage learning-based Genetic Algorithms (GAs) aim at solving problems with blocks of dependent (linked) variables more efficiently by exploiting linkage information in the variation operator. Improved efficiency in terms of the number of evaluations to find the optimum is achieved by alleviating a fundamental problem of simple GAs [2]: preventing the disruption of blocks of dependent variables. In BBO the linkage structure is not known a priori, and, therefore, needs to be learned. Moreover, as shown by [3], it is more efficient to learn the linkage structure from a population of solutions than using a static predefined one. There are several generations of such GAs, with different approaches to linkage learning. These include the Estimation of Distribution Algorithms (EDAs) that estimate a probability distribution from selected solutions and then sample new solutions, based on, e.g., marginal distribution calculation [4, 5] or Bayesian networks [6]. More generally, such algorithms are said to build a model to guide variation. In the case of EDAs, this is a probabilistic model. More recently, models have come into focus that do not necessarily describe a probability distribution directly, but rather focus on expressing linkages between variables [7]. A state-of-the-art model-based binary GA is the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) [8]. GOMEA uses linkage learning from the population. This can be done in several ways, but the most generally applied approach builds a so-called linkage tree, by performing hierarchical clustering based on Mutual Information between variables. An overview of GOMEA is provided in Section 3.2.2. A more detailed description can be found in [8].

In the context of optimization, a surrogate model is a function that approximates an original fitness function. Combining a surrogate model with a GA results in a class of algorithms called surrogate-assisted GAs. This is a common approach to reduce the number of evaluations for solving expensive optimization problems. Replacing some real function evaluations with evaluations of the surrogate model, which are considered to consume much less time, is a natural approach to reduce the overall runtime. For real-valued problems, common types of surrogate models integrated into GAs are polynomials, Kriging, Radial Basis Function Network (RBFN), and Support Vector Regression (SVR) (see, e.g., [1]). There is not much literature on surrogate-assisted GAs for binary problems. One of the examples is the recently introduced convolutional neural network surrogate combined with GOMEA (Chapter 4).

One of the key issues of using surrogate models in GAs is balancing between real function evaluations and surrogate ones to find the optimum. A conventional surrogate model is an approximation of a fitness function, and, therefore, it is not perfectly accurate. As shown, for instance, in [9], finding an optimal solution using only surrogate evaluations is in many cases impossible. However, as demonstrated in Chapter 4, carefully

selected combinations of real and surrogate evaluations can allow to solve a problem to optimality, but only if the surrogate model is sufficiently accurate. These issues of using surrogate models in combination with GAs can be alleviated if the surrogate model perfectly approximates the fitness function. The efficient and scalable way of learning such a surrogate and using it in combination with GAs to solve optimization problems (finding the optimum) is the main research question studied in this chapter.

We consider optimization of pseudo-Boolean functions with k-bounded epistasis meaning that a function f maps binary vectors of length ℓ to a real value: $\{0,1\}^\ell \rightarrow \mathbb{R}$ and can be represented in a form:

$$f(x) = \sum g_i(x),$$

where each $g_i(x)$ depends on at most k variables. Such a definition describes a broad class of functions, including well-known benchmark functions, such as deceptive traps, NK landscapes, MaxCut, and MAXkSAT problems. Walsh decomposition entails [10] that any pseudo-Boolean function can be represented as a linear combination of Walsh functions:

$$f(x) = \sum_{s \in \{0,1\}^\ell} w_s \psi_s(x),$$

where

$\psi_s(x) = (-1)^{bc(s \wedge x)}$,
$bc(i)$ is a bit counting function that indicates the number of ones in a vector i,
and $\wedge$ is the bitwise binary *AND* operator .

The terms w_s are the Walsh coefficients of f. Each binary vector $s \in \{0,1\}^\ell$ has a corresponding Walsh coefficient, thus, there are 2^ℓ Walsh coefficients in total. The order of a Walsh coefficient is defined as the number of ones in the corresponding binary vector s. Alternatively, the binary vector s can be written as a set of variables that correspond to the positions that have value 1. Further, in this chapter, we refer to binary vectors that determine Walsh coefficients as masks. The Walsh coefficient w_s is the contribution to the fitness function of the subfunction g_s which depends on all the variables in s. Therefore, a zero Walsh coefficient means that such a contribution is zero.

In this chapter, we take the extreme view on surrogate-assisted GAs, i.e., with a surrogate model $f(x)$ being a precise Walsh decomposition of a problem. In case the Walsh coefficients are known, the Walsh decomposition can be used as a surrogate model and thus be directly optimized without any additional real evaluations. In general, to efficiently calculate Walsh coefficients, the problem structure needs to be known. This is because if a function is $k-$bounded (all its subfunctions depend on no more than k variables), all its Walsh coefficients of order greater than k are zeros. However, the total number of Walsh coefficients of order $\leq k$ is still large, namely, $\mathcal{O}(\ell^k)$. With one order j coefficient calculation requiring 2^j function evaluations, the calculation of all coefficients of order $\leq k$ is intractable. However, only Walsh coefficients corresponding to subsets of variables, which are dependent, and therefore, belong to one subfunction, need to be calculated. Under the assumption that the number of subfunctions is $\mathcal{O}(\ell)$ (for example, well-known adjacent NK-landscapes [11] and Concatenated Deceptive Traps

[12] fall under this category), all Walsh coefficients can be calculated with $\mathcal{O}(\ell 2^k)$ number of evaluations [10]. In BBO it is therefore important to efficiently discover the problem structure to avoid redundant calculations. Now, to find the subfunctions, we need a form of linkage learning, i.e., learning which blocks of dependent variables exist. Therefore, from the proposed perspective on surrogate-assisted optimization, the tasks of learning Walsh coefficients and linkage learning are tightly connected.

Apart from linkage learning algorithms inside GAs, there are many works on standalone linkage learning algorithms. Such algorithms are different from the ones typically used inside linkage learning-based GAs, as their main goal is finding all linkage sets (obtaining the full linkage structure). Such algorithms are usually probabilistic, i.e., their complexity is analyzed with the assumption that a correct linkage structure is discovered with a high probability $1-\epsilon$. Many linkage learning algorithms of this type are based on perturbations of solutions. The basic idea of using perturbations to learn linkage for problems with both overlapping and non-overlapping subfunctions was introduced in [13] and [14] (LINC and LIMD algorithms). These algorithms do pairwise checks of linkage between all pairs of variables, leading to the complexity $\mathcal{O}(2^k\ell^2)$. As k is assumed to be constant, the actual asymptotic complexity is $\mathcal{O}(\ell^2)$. In [10] a more practically efficient (though having the same complexity of $\mathcal{O}(\ell^2)$) algorithm was proposed. Moreover, in this work an algorithm to efficiently calculate Walsh coefficients after learning linkage structure was introduced. Instead of checking linkage between all pairs of variables, inevitably leading to the lower bound of complexity $\mathcal{O}(\ell^2)$, [15] introduced the idea of using binary search for finding a variable j linked with the variable i by iteratively dividing the set of variables possibly linked with i in two parts until a part contains only one variable. This approach is different from [10] as it does not use the Walsh coefficient concept to define linkage, but uses perturbations instead. This linkage learning method is followed by a simple greedy search to optimize the fitness function. The work of [15] is limited to separable functions (subfunctions that are non-overlapping). A similar idea of using binary search was later used in an algorithm for arbitrary k-bounded problems in [16]. A rigorous complexity analysis proved, that there exists an algorithm that learns the problem linkage structure with the complexity $\mathcal{O}(Mlog\ell)$, with M being the number of non-zero Walsh coefficients. It should be noted that this complexity is asymptotic. For practical usage, the constant term in the complexity is crucial and the constant term in this algorithm is $((16e)^k k^{3.5})$, making usage intractable for practical applications.

A completely different idea from perturbations-based linkage learning algorithms was introduced in [17]. Instead of learning linkage structure directly and then computing Walsh coefficients, it was proposed to estimate Walsh coefficients as weights of a linear regression model. In machine learning terms, the regression problem can be formulated with $\psi_s(x)$ being features and fitness values $f(x)$ being targets. All coefficients up to a beforehand selected order k and corresponding features in a regression problem are generated. To efficiently exploit the sparsity of Walsh coefficients, linear regression with $L1$ regularization is used. $L1$ regularized regression is capable of naturally forcing part of the coefficients to be zero. In the follow-up of this approach introduced in [18], the Walsh coefficients approximation is followed by a local search optimization procedure. Such an approach is shown to be efficient for problems with a moderate k ($k \leq 3$). For problems with subfunctions of higher orders, such a method requires a large population

of solutions (to train linear regression), as the number of Walsh coefficients rapidly grows with k: for a k-bounded function the total number of coefficients of order $\leq k$ is $\mathcal{O}(\ell^k)$.

We propose to combine the ideas from algorithms [10] and [15] in a way that leads to a new linkage learning algorithm. The basic idea of our algorithm is to detect pairwise dependencies between variables as the first step. Then, these pairwise dependencies are used to form linkage sets of higher orders. A subset of variables is considered as a candidate linkage set if all pairs of variables belonging to it are already known to be dependent variables. The Walsh coefficients calculation algorithm from [10] is used to calculate the Walsh coefficients after the linkage structure is learned. The algorithm continues working until the Walsh decomposition quality is perfect on a holdout set of solutions. After that, GOMEA with partial evaluations is used to effectively and efficiently optimize the Walsh decomposition surrogate model, so no additional evaluations are required. The algorithmic complexity of the Walsh coefficients calculation in the case of randomly constructed subfunctions is $\mathcal{O}(\ell log \ell)$. The two important conditions for such algorithmic complexity are the linear number of subfunctions ($\mathcal{O}(\ell)$) and $k-$boundedness of all subfunctions. Besides efficiency, the proposed optimization approach has the merit of being parameterless.

The remainder of this chapter is organized as follows. In Section 3.2 we describe some needed background, particularly, notation, theory, and key features of the GOMEA algorithm. In Section 3.3 the details of the proposed approach and the considered alternative approaches are provided. Then, in Section 3.4 the experimental setup and benchmark functions are specified. These are followed by an explanation of the obtained experimental results and the discussion. The chapter ends with the conclusions.

3.2. BACKGROUND

3.2.1. NOTATION AND THEORY

Definition 1. We say that there is a linkage or a dependency between two variables if there is a non-linear interaction between them. It means that the change in f that is observed when variables i and j both change cannot be represented as a linear function of the change in f that is observed if only variable i and only variable j changes.

Definition 2. Partial derivative of the i^{th} bit in a solution is defined as

$$\Delta f_i(x) = f(x[i \leftarrow 1]) - f(x[i \leftarrow 0]),$$

where the notation $x[i \leftarrow v]$ means setting the value of the i^{th} bit of a solution x to the value v. We use the concept of the partial derivative to detect pairwise dependencies between variables.

Definition 3. Variables i and j are linked if-and-only-if $\exists x : \Delta f_i(x[j \leftarrow 0]) \neq \Delta f_i(x[j \leftarrow 1])$. Intuitively, this means that two variables are linked if-and-only-if the contribution to the fitness of one variable depends on the value of the second variable.

Example 1. Let f be a pseudo-Boolean function over 4 variables, $f = x_1 + x_2 + x_1x_3 + x_2x_4$. For $x = (0110) : \Delta f_1(x[3 \leftarrow 0]) = f(1100) - f(0100) = 2 - 1 = 1; \Delta f_1(x[3 \leftarrow 1]) = f(1110) - f(0110) = 3 - 1 = 2; \Delta f_1(x[3 \leftarrow 0]) \neq \Delta f_1(x[3 \leftarrow 1])$, thus, there is a dependency between variables x_1 and x_3. Indeed, variables x_1 and x_3 occur in the non-linear subfunction x_1x_3.

Definition 4. A set of variables m is a linkage set if-and-only-if $\exists j : m \subseteq j$ with a non-zero corresponding Walsh coefficient w_j.
In other words, linkage sets represent sets of epistatically linked bits [10].

Definition 5. Following the notation from [10], we define a probe over a pseudo-Boolean function f, a subset of variables m and a subset of variables c for which $c \cap m = \emptyset$ (called a probe background) as follows:

$$P(f, m, c) = \frac{1}{2^{bc(m)}} \sum_{i \in B_m} (-1)^{bc(i)} f(i \oplus c),$$

where m is a mask denoting the bits to be tested; B_m is a subset of $\{0,1\}^\ell$ such that all variables are in m; $\oplus$ is the bitwise binary XOR operator, and $bc(i)$ is a bit counting function. A detailed explanation of probes properties can be found in [10].

Definition 6. Variables of set m are linked if-and-only-if $\exists c : m \cap c = \emptyset, P(f, m, c) \neq 0$.
This alternative definition of linkage between variables is used in the linkage learning algorithm based on probes [10]. For the case of m having 2 variables, this definition is equivalent to Definition 3 as $P(f, \{i, j\}, c) = \frac{1}{4}(f(c[i \leftarrow 0][j \leftarrow 0]) - f(c[i \leftarrow 1][j \leftarrow 0]) - f(c[i \leftarrow 0][j \leftarrow 1]) + f(c[i \leftarrow 1][j \leftarrow 1]) = \frac{1}{4}(\Delta f_i(c[j \leftarrow 1]) - \Delta f_i(c[j \leftarrow 0]))$. Therefore, variables i and j are linked if-and-only-if $\exists c : \Delta f_i(c[j \leftarrow 1]) - \Delta f_i(c[j \leftarrow 0]) \neq 0$ which is identical to Definition 3.

Example 2. Let f be the same pseudo-Boolean function as in Example 1. For mask $x = (1010)$ and $c = (0100) : P(f, x, c) = \frac{1}{4}(f(0100) - f(1100) - f(0110) + f(1110)) = \frac{1}{4}(1 - 2 - 1 + 3) = \frac{1}{4} \neq 0$, thus, there is a dependency between variables x_1 and x_3.

Definition 7. We assume that variables i, j jointly occur in a subfunction g if and only if there is a non-linear interaction between them. In terms of probes, it means that $\exists c : \{i, j\} \cap c = \emptyset, P(g, \{i, j\}, c) \neq 0$. Informally, it means that the non-linear interactions between variables exist only inside subfunctions g.

Note, that the occurrence of two variables in a subfunction means a non-linear interaction between them exists only if all subfunctions g are non-linear. However, such an assumption does not induce any loss of generality, as any term of a subfunction, that is linear in two variables, can be replaced by a sum of subfunctions depending on a single variable each. For instance, f from Example 1 can be represented, as $f = g_1 + g_2 + g_3$, where $g_1 = x_1 + x_2, g_2 = x_1 x_3$, and $g_3 = x_2 x_4$. In such a representation, g_1 is linear in x_1 and x_2, but we can rewrite the representation as $f = g_1 + g_2 + g_3 + g_4$, where $g_1 = x_1, g_2 = x_2, g_3 = x_1 x_3$, and $g_4 = x_2 x_4$. In this representation, there are no subfunctions that are linear in a pair of variables. As a corollary of this definition, there is a (non-linear) subfunction g depending on a subset of variables m if and only if $\forall \{i, j\} \subseteq m$ there is a linkage between variables i and j.

We provide an important theorem, which forms the basis of the Walsh coefficients calculation algorithm after the linkage structure is discovered. This theorem is proven in [10] and we therefore refer the interested reader to this chapter.

Theorem 1. A probe value with the all-zeros background $c = 0$ and the Walsh coefficients of function f are naturally connected, namely, by the formula:

$$P(f, j, 0) = w_j + \sum_{j \subset u} w_u \text{ , where } u \text{ are sets containing } j.$$

3.2.2. GENE-POOL OPTIMAL MIXING EVOLUTIONARY ALGORITHM

GOMEA is a model-based evolutionary algorithm that aims to efficiently assemble building blocks into highly-fit solutions by estimating and exploiting linkage information [8] in a manner that is very different from the algorithm for exact linkage learning that we propose in Section 3.3.2. The variation operator of GOMEA is called Gene-pool Optimal Mixing (GOM), which can be optionally followed by the Forced Improvements (FI) procedure. GOMEA is often used together with an Interleaved Multistart Scheme (IMS) that automatically sets the population size by starting and running multiple runs with increasing population size in an interleaved fashion, making its use parameterless [19]. Moreover, the GOM operator allows using partial fitness function evaluations to accelerate the optimization process whenever possible. We give a brief outline of the key components of GOMEA here and refer the interested reader to the literature for more details [8, 19, 20].

LINKAGE LEARNING

Multiple linkage learning algorithms and corresponding linkage structures have been introduced for GOMEA. In this chapter, we stick to a relatively simple, yet efficient linkage structure - the Linkage Tree (LT) [8, 20]. We call the obtained structure, containing subsets of linked variables the Family of Subsets (FOS). A FOS $\mathscr{F}$ is a set of sets: $\mathscr{F} = \{F_1 \dots F_{|\mathscr{F}|}\}$, where $F_i \subseteq \{1 \dots \ell\}$ for $i = 1 \dots |\mathscr{F}|$. The LT is built using Mutual Information between variables calculated from the solutions in a population. Particularly, the LT structure is the result of a hierarchical merge procedure. In the first step, each problem variable forms its own subset F. Then, the tree is constructed by iteratively merging two subsets. The process is stopped when there are only two subsets, the union of which contains all variables. The details of the LT building procedure are provided in [8].

GENE-POOL OPTIMAL MIXING

GOMEA uses the learned FOS to generate offspring solutions. First, a parent solution is copied to create a new offspring solution. Then, for each subset in the FOS (iterated over in random order), a donor solution is randomly selected from the population. For every subset, genes from loci corresponding to the selected FOS subset are copied from the donor to the offspring. The change is accepted only in case the fitness function value is at least as good as before the change.

FORCED IMPROVEMENTS

The FI procedure improves the fitness value of a solution in case the GOM operator has failed to make any changes to the offspring, or if this individual has not improved for $1 + log(populationSize)$ generations. The FI procedure consists of performing the mixing of the offspring with the global (taken over all populations) elitist solution. The mixing operator is the same as in the GOM operator. The change is now however accepted only

in case of a strict improvement of the fitness value. The FI procedure is immediately terminated when the first improvement occurs. In case the FI procedure failed to improve the solution fitness, the global elitist solution is copied to the offspring.

INTERLEAVED MULTISTART SCHEME

The goal of the IMS is to replace the manual setting of the population size parameter. Instead of evolving one population of a fixed size, several populations of increasing sizes are run simultaneously. The smallest population in the scheme is of a predetermined size, for simplicity, it can be of size 1. The next populations are double the size of the previous population. The populations are running in an interleaved fashion, which means that for each c^{IMS} offspring's generation iterations of a population of size *populationSize*, one iteration of the population with population size 2*populationSize* is performed. We set c^{IMS} to 4 [19]. A population is terminated if a larger population obtained a better average fitness value.

PARTIAL EVALUATIONS

If it is known beforehand how the problem can be decomposed into subfunctions, it is reasonable to use this knowledge to accelerate function evaluations. This is especially useful in GOMEA. Namely, if a subset selected for mixing (during GOM or FI) contains variables $x_{i_1},\ldots,x_{i_k}$, the subfunctions depending on these variables are $g_{j_1}(x),\ldots,g_{j_p}(x)$, the solution before mixing is x and its fitness is F, then the fitness of a solution after mixing is: $F' = F - g_{j_1}(x) - \cdots - g_{j_p}(x) + g_{j_1}(x') + \cdots + g_{j_p}(x')$. In many cases, the number of subfunctions p is much less than the total number of subfunctions, resulting in the required amount of computations to be much fewer than when performing a full fitness evaluation (computing all the subfunctions).

PARTIAL EVALUATIONS FOR A WALSH DECOMPOSITION

Suppose that we have a precise Walsh decomposition $f(x) = \sum_{s\in\{0,1\}^\ell} w_s\psi_s(x)$, i.e., Walsh coefficients w_s are known. Suppose we also know the fitness F of a solution x. If we need to calculate the fitness of x after changing variables $x_{i_1},\ldots,x_{i_k}$, we find the subsets of variables $s_1\ldots,s_p$ (of which the corresponding Walsh coefficients are $w_{s_1}\ldots,w_{s_p}$) that contain any variables from $\{x_{i_1},\ldots,x_{i_k}\}$ and calculate the new fitness as: $F' = F - w_{s_1}\psi_{s_1}(x) - \cdots - w_{s_p}\psi_{s_p}(x) + w_{s_1}\psi_{s_1}(x') + \cdots + w_{s_p}\psi_{s_p}(x')$.

3.3. SURROGATE-ASSISTED GENETIC ALGORITHMS

3.3.1. GENERAL OUTLINE OF THE PROPOSED APPROACH

We propose to combine the calculation of the Walsh coefficients and GOMEA. When the Walsh decomposition demonstrates perfect accuracy on a holdout set of solutions, we use the obtained Walsh decomposition as the surrogate model. Having as input a solution x, the surrogate entails the formula $\sum_{s\in\{0,1\}^\ell} w_s\psi_s(x)$. Walsh decomposition being perfect means that the coefficient of determination $R^2 = 1$. By definition, $R^2 = 1 - \frac{MeanSquaredError(Y,\hat{Y})}{var(Y)}$, where Y are true fitness values of a set of solutions and $\hat{Y}$ are estimations made by the surrogate. In practice, a $\epsilon-$deviation threshold for surrogate quality (i.e. $R^2 \geq 1-\epsilon$) can be used to handle possible numerical errors. The value of ϵ

should be small enough to ensure the surrogate model optimum is matched with the optimum of the original function.

Once the Walsh coefficients are precisely known, we can optimize the surrogate without performing any additional evaluations of the initial function. As Walsh calculation algorithms we consider our newly proposed method (further referred to as Efficient Linkage Learning, ELL), the probes-based linkage learning algorithm from [10] (further referred to as Probes based Linkage Learning, PLL), and the Walsh approximation based on Least Angle Regression (LARS) [17] (further referred to as LARS based Linkage Learning, LARSLL). In the ELL and the PLL first dependencies between variables are detected and then non-zero Walsh coefficients are calculated. In the LARSLL Walsh coefficients are directly approximated.

In all cases, we optimize the obtained Walsh decomposition with GOMEA using partial evaluations as described in Section 3.2.2. We identify the full optimization procedures consisting of the calculation of Walsh coefficients followed by GOMEA by ELL-GOMEA, PLL-GOMEA, and LARSLL-GOMEA, respectively.

3.3.2. GOMEA WITH THE EXTERNAL EFFICIENT LINKAGE LEARNING (ELL-GOMEA)

We use binary search and partial derivatives to design an efficient algorithm for the learning of pairwise dependencies. Solutions to use for testing whether there is a dependency between variables are randomly generated. After calling the binary search routine for detecting pairwise dependencies for each variable, we construct candidate linkage sets of increasing order (from 3 to ℓ). For each set of candidates, Walsh coefficients are calculated (all other Walsh coefficients are assumed to be zero) and the quality of the Walsh approximation is evaluated on a holdout set of solutions. The linkage tests are performed until the decomposition quality is perfect.

In detail, the proposed optimization approach consists of the following parts:

1. **Detecting all linkage sets of a problem.**
   1.1. The first part of the linkage learning procedure aims at detecting all pairwise dependencies between variables. In [10] it is shown that detecting order 2 dependencies between variables is the most expensive part of the linkage learning process in terms of the number of function evaluations. Therefore, we focus on designing an efficient algorithm for this purpose. This part of the algorithm is described in function *detectLinkagePairsBinarySearch* from Algorithm 3.3.2. Given variable i, the algorithm uses binary search to detect a variable j, which is linked to i.

   The linkage detection algorithm (*detectLinkagePairsBinarySearch*) is called for every variable. For each variable i, two solutions s_1, s_2 are generated randomly. For all variables $V = \{v_1, \ldots, v_{|V|}\}$, which are already known to be linked with i, genes in s_2 are set to the same values as genes in s_1: $s_2[v_1 \leftarrow s_1.v_1], \ldots, s_2[v_{|V|} \leftarrow s_1.v_{|V|}]$. The same applies to the i-th gene value: $s_2[i \leftarrow s_1.i]$. Now, if $\Delta f_i(s_1) \neq \Delta f_i(s_2)$ (two function evaluations required to calculate one $\Delta f(x)$ expression), it means that there is a variable j linked to i among $U = \{i : s_1.i \neq s_2.i\}$. The binary search can be used to efficiently discover this variable j. Note: strictly speaking, i can be linked to multiple variables that

are not discovered yet. The binary search detects one arbitrary variable j among them. We can divide U in two parts: $part_1 = \{u_1, \ldots, u_{\lfloor |U|/2 \rfloor}\}$, $part_2 = \{u_{\lfloor |U|/2 \rfloor + 1}, \ldots, u_{|U|}\}$. A new solution s_3 is built by combining values from $part_1$ loci of s_1 and $part_2$ loci of s_2:

$$s_3.i = \begin{cases} s_1.i & \text{if } i \in part_1, \\ s_2.i & \text{if } i \in part_2, \\ s_1.i = s_2.i & \text{otherwise.} \end{cases}$$

Thus, s_3 differs from s_1 in bits from $part_2$ only, and from s_2 in bits from $part_1$. Now, at least one of two cases holds: $\Delta f_i(s_1) \neq \Delta f_i(s_3)$ or $\Delta f_i(s_2) \neq \Delta f_i(s_3)$. In the first case, the variable j linked to i is localized in $part_2$. In the second case, the linked variable is localized in $part_1$. Thus, the size of the subset of candidate variables is halved (if both cases hold, an arbitrary one can be picked) and a binary search continues until the size of a considered part is 1.

After the algorithm reveals a variable j that is linked with a variable i, a pair $\{i, j\}$ is stored in a set containing all pairs of linked variables and the search for the next linked pairs continues.

1.2. In the second part of the linkage learning procedure, candidate linkage sets of higher order are constructed. The main idea of this part of the algorithm is the following: a set of variables $S(|S| = N)$ is a linkage set if and only if $\forall \{i, j\} \subseteq S$, $\{i, j\}$ is a linkage pair. Thus, having a linkage set S of order N, we can obtain all candidate linkage sets of order $N+1$ by checking for all variables $i_{k_1}, \ldots, i_{k_{\ell-N}}$ not present in S whether there is a dependency between a variable i_{k_j} and all variables from S. Note, that such a constructed set is not necessarily a linkage set, but as our experiments show, in practice, it is more efficient to consider such sets as candidates and calculate Walsh coefficients for them compared to attempting to prove, that a set is indeed a linkage set using probes. This procedure is described in function *linkageSetsOfHigherOrder* of Algorithm 3.3.2.

2. **Calculating the Walsh decomposition coefficients**. To calculate the Walsh decomposition coefficients, we use an algorithm idea proposed in [21] and its specification from [10]. It is based on Theorem 1. It uses the notion of the probe and is based on the formula (the proof is provided in [10]) $P(f, j, 0) = w_j + \sum_{j \subset u} w_u$. Thus, $P(f, j, 0) = w_j$ for a Walsh coefficient of the highest order (i.e., $\forall u : j \subset u \; : \; w_u = 0$). Walsh coefficients of lower orders can be calculated using a top-down approach. A detailed description of this Walsh coefficients calculation algorithm is provided in Algorithm 3.3.1. To calculate the value of a Walsh coefficient of order k, $\mathcal{O}(2^k)$ function evaluations are required.
3. **Finding an optimum of the function**. If the obtained Walsh decomposition is perfectly accurate, then optimizing the problem is equal to optimizing the Walsh decomposition. We use a holdout set of solutions to validate the quality of a calculated Walsh decomposition. The question of choosing the holdout set size is addressed in Section 3.3.4. The coefficient of determination R^2 defines the quality of the decomposition. The approximate Walsh decomposition $\hat{w}$ estimates the fit-

ness value $\hat{y}$ of a solution x by the formula $\hat{y} = \sum_{s \in \{0,1\}^\ell} \hat{w}_s \psi_s(x)$. The coefficient of determination R^2 on a set of solutions is defined as $R^2 = 1 - \frac{MeanSquaredError(Y,\hat{Y})}{var(Y)}$, where Y are true fitness values. In Algorithm 3.3.2 R^2 is calculated by the function *calculateApproximationQuality(holdoutPopulation, WalshCoefficients)*. After the perfect decomposition (i.e. $R^2 = 1$) is obtained, meaning that the precise values of all Walsh coefficients w_s are known, we optimize (maximize) the Walsh decomposition $\sum_{s \in \{0,1\}^l} w_s \psi_s(x)$ using GOMEA with partial evaluations. Note, that in this part of the algorithm, no evaluations of the initial function are performed.

COMPLEXITY ANALYSIS

We theoretically infer the $\mathcal{O}(\ell log \ell)$ complexity upper bound of the ELL algorithm, later the experimental results support the $\mathcal{O}(\ell log \ell)$ complexity of the algorithm. Following the complexity analysis in [10] our analysis is restricted to the class of additively decomposable, $k-$bounded functions with randomly chosen subfunctions.

First, we estimate the required number of tests to make (the number of *detectLinkagePairsBinarySearch* calls for each variable) to detect all order 2 dependencies. Given two random solutions $s1, s2$ and variables i, j we estimate the probability that there is a success in detecting a dependency between these variables: the probability that *detectLinkagePairsBinarySearch* reveals that i and j are linked having i as the input. Let J be the number of linked variables to i. To guarantee that *detectLinkagePairsBinarySearch* finds the linked variable j, all dependent variables with i should be equal in s_1 and s_2 except the variable j. If i is involved in M subfunctions, J is $< Mk$. Thus, the probability that all these variables are equal in s_1 and s_2 except gene j, is $> 2^{-Mk}$. In case of random subfunctions and number of subfunctions $\mathcal{O}(\ell)$, M is $\mathcal{O}(k)$, therefore, the probability of detecting a dependency between variables i and j with i as input is at the worst case $\mathcal{O}(2^{-Mk}) = p(k)$ (does not depend on ℓ).

Suppose, for each variable, N calls of *detectLinkagePairsBinarySearch* are made. Now, we calculate the probability of detecting all variables $\{j_1, \ldots, j_J\}$ linked with i with these calls assuming that no linked variables with i are already detected. We can divide N executed calls into J groups of $\frac{N}{J}$ calls. We calculate the probability that during the first group of calls, j_1 was found to be linked with i, during the second group of calls linkage with j_2 was detected, etc. As the probability of success in linkage detecting of a particular variable with i is p, the probability of not detecting it, using $\frac{N}{J}$ attempts, is $(1-p)^{\frac{N}{J}}$. Therefore, the probability of detecting it (at least in one of the attempts) is $1-(1-p)^{\frac{N}{J}}$. Hence, the probability of detecting all J linked variables with i, each within its own group of *detectLinkagePairsBinarySearch* calls, is $(1-(1-p)^{\frac{N}{J}})^J$. Note, that the actual probability of detecting J linked variables within N calls is larger, as a dependency with a variable can be found during all N calls and not only during the selected $\frac{N}{J}$ calls. All linked variables for all ℓ variables should be detected. Under the assumption that all variables have exactly J linked variables, the probability that all linked variables are found for all variables is $((1-(1-p)^{\frac{N}{J}})^J)^\ell$. We want this probability to be larger than a constant probability δ (reasonably close to 1). Note, that the actual probability, that all pairwise linkage sets are discovered by N calls of *detectLinkagePairsBinarySearch* for each variable is larger, as, first, some variables can have less than J linked variables,

and second, here we calculated each dependency $\{i, j\}$ as discovered twice: finding that variable j is linked with i and the other way around. Now, we need to solve the inequality $((1-(1-p)^{\frac{N}{J}})^J)^\ell > \delta$. The full derivation is provided in the Appendix (Derivation A). By solving it, we get $N > Cln(1-\lambda^{\frac{1}{\ell}})$ (C and λ are constants). As proven in the Appendix (Derivation B), $\mathcal{O}(ln(1-\lambda^{1/\ell})) = \mathcal{O}(log\ell)$. Therefore, $N = \mathcal{O}(log\ell)$ is the upper bound of *detectLinkagePairsBinarySearch* calls needed for every variable to detect all its dependencies with high probability δ.

If the upper bound of *detectLinkagePairsBinarySearch* calls needed for each variable is $\mathcal{O}(log\ell)$, the total number of calls is not larger than $\mathcal{O}(\ell log\ell)$. One binary search procedure in *findLinkage* requires $\mathcal{O}(log\ell)$ evaluations. However, the binary search is invoked only if there is a dependency between variables (when the *if* clause in line 6 of *detectLinkagePairsBinarySearch* is true). The number of binary search calls is therefore $\mathcal{O}(\ell)$. Thus, the total number of evaluations to detect all order 2 dependencies with high probability δ is $\mathcal{O}(\ell log\ell)$. As proven in [10], in the case of functions with randomly chosen subfunctions, the number of order k variables subsets, such that it is not a linkage set, but has a subset of order $k-1$ which is a linkage set, is $\mathcal{O}(1)$ for $k \geq 3$. Thus, calculating Walsh coefficients for spurious linkage sets candidates of orders higher than 2 (generated by *linkageSetsOfHigherOrder*) does not increase the complexity. The Walsh coefficients calculation for all built candidate subsets (including order 2 subsets) requires $\mathcal{O}(\ell 2^k)$ evaluations. Therefore, the overall complexity of obtaining the Walsh decomposition with the ELL algorithm is $\mathcal{O}(\ell log\ell)$.

Algorithm 3.3.1: The algorithm for Walsh coefficients calculation [10].

```
Function calculateWalshCoefficients(linkageSets):
    allZerosProbes ← ∅
    WalshCoefficients ← ∅
    for subset ∈ linkageSets do
        if subset ∉ allZerosProbes then
            allZerosProbes[subset] ← P(f, subset, 0)
    linkageSets ← sorted linkageSets by subset size in descending order
    for subset ∈ linkageSets do
        if WalshCoefficients[subset] ≠ NULL then
            WalshCoefficients[subset] ← WalshCoefficients[subset] + allZerosProbes[subset]
        else
            WalshCoefficients[subset] ← allZerosProbes[subset]
        for s ⊆ subset do
            if WalshCoefficients[s] ≠ NULL then
                WalshCoefficients[subset] ← WalshCoefficients[s] − WalshCoefficients[subset]
            else
                WalshCoefficients[s] ← −WalshCoefficients[subset]
    return WalshCoefficients
```

Algorithm 3.3.2: The ELL algorithm (binary search-based linkage learning).

Function `detectLinkagePairsBinarySearch`(*i*, *linkagePairs*)**:**
- $solution_1, solution_2 \leftarrow$ random solutions
- **for** $\{i, v\} \in linkagePairs$ **do**
  - $solution_2.v \leftarrow solution_1.v$
- $solution_2.i \leftarrow solution_1.i$
- **if** $\Delta f_i(solution_1) - \Delta f_i(solution_2) \neq 0$ **then**
  - $j \leftarrow$ `findLinkage` $(solution_1, solution_2, i)$
  - $linkagePairs \leftarrow linkagePairs \cup \{i, j\}$
- **return** *linkagePairs*

Function `findLinkage`$(solution_1, solution_2, i)$**:**
- $U = i_1, \ldots, i_{|U|} \leftarrow \{i : solution_1.i \neq solution_2.i\}$
- **if** $|U| = 1$ **then**
  - **return** i_1
- $solution_3 \leftarrow solution_2$
- $solution_3 \leftarrow solution_3[i_1 \leftarrow solution_1.i_1], \ldots, [i_{\lfloor |U|/2 \rfloor} \leftarrow solution_1.i_{\lfloor |U|/2 \rfloor}]$
- **if** $\Delta f_i(solution_1) - \Delta f_i(solution_3) \neq 0$ **then**
  - **return** `findLinkage` $(solution_1, solution_3, i)$
- **else**
  - **return** `findLinkage` $(solution_2, solution_3, i)$

Function `linkageSetsOfHigherOrder`(*linkageSets*, *linkagePairs*, *maxOrder*)**:**
- $linkageSetsOfMaxOrder \leftarrow \emptyset$
- **for** *set* $\in$ *linkageSets of order maxOrder* $- 1$ **do**
  - **for** *superset of set of size maxOrder* **do**
    - **if** *all pairs in superset* $\in$ *linkagePairs* **then**
      - $linkageSetsOfMaxOrder \leftarrow linkageSetsOfMaxOrder \cup \{superset\}$
- **return** *linkageSetsOfMaxOrder*

Function `efficientLinkageLearning`(*holdoutPopulation*)**:**
- $linkagePairs \leftarrow \emptyset$
- **while** `true` **do**
  - **for** $i = 1, \ldots,$ *numberOfVariables* **do**
    - $linkagePairs \leftarrow$ `detectLinkagePairsBinarySearch` (i, *linkagePairs*)
  - $linkageSets \leftarrow \{\{\}, \{1\}, \ldots, \{\ell\}\}$
  - **for** $maxOrder = 2, \ldots,$ *numberOfVariables* **do**
    - *currentHigherOrderSets* $\leftarrow$ `linkageSetsOfHigherOrder`(*linkageSets*, *linkagePairs*, *maxOrder*)
    - **if** *currentHigherOrderSets* $= \emptyset$ **then**
      - **break**
    - $linkageSets \leftarrow linkageSets \cup currentHigherOrderSets$
  - *WalshCoefficients* $\leftarrow$ `calculateWalshCoefficients`(*linkageSets*)
  - $R^2 \leftarrow$ `calculateApproximationQuality` (*holdoutPopulation*, *WalshCoefficients*)
  - **if** $R^2 = 1$ **then**
    - **return** *WalshCoefficients*

3.3.3. THE CONSIDERED ALTERNATIVE APPROACHES

PROBES-BASED LINKAGE LEARNING (PLL)

As the first alternative approach, we choose the parameterless modification of the linkage learning and Walsh coefficients calculation algorithm proposed in [10] followed by the optimization of the obtained Walsh decomposition by GOMEA. This algorithm is based on the probe notion. As a subset j is a linked group if $P(f, j, c) \neq 0$, the algorithm tries to detect the linkage with a randomly selected background c. The quality of the decomposition is measured in the same way as in Algorithm 3.3.2: a perfect decomposition means $R^2 = 1$ on a holdout population. In each iteration, the algorithm starts by detecting order 2 linkage sets. Then it moves on to order 3 groups: a set of 3 variables is selected for testing by probes if all its subsets of 2 variables are already detected as linked sets. In general, after performing linkage sets detection of order s, candidates for order $s+1$ linkage sets are constructed: a set of size $s+1$ is a candidate if all its subsets of size s are already detected linkage sets. If no linkage sets of order p are detected, it means that no linkage sets of order $> p$ can be detected and the algorithm iteration is finished. In the original algorithm, the number of backgrounds to test is set manually. In our implementation, we do one test for each candidate set of linked variables until the perfect decomposition is obtained. Thus, the algorithm usage becomes parameterless. The full algorithm description is provided in Algorithm 3.3.3.

Algorithm 3.3.3: The PLL algorithm.

```
Function checkLinkageByProbes(setToCheck):
    probeBackground ← random solution
    for i ∈ setToCheck do
        probeBackground.i ← 0
    if P(f, setToCheck, probeBackground) ≠ 0 then
        return true
    return false
Function probesBasedLinkageLearning(holdoutPopulation):
    linkageSets ← {{} ,{1},...,{ℓ}}
    while true do
        for maxOrder ← 2...ℓ do
            candidateSets ← ∅
            setsOfMaxOrderDetected ← false
            for superset of order maxOrder in supersets of sets ∈ linkageSets do
                if all subsets of the superset ∈ linkageSets then
                    candidateSets ← candidateSets ∪ {superset}
            for set ∈ candidateSets do
                if checkLinkageByProbes (set) = true then
                    linkageSets ← linkageSets ∪ set
                    setsOfMaxOrderDetected ← true
            if setsOfMaxOrderDetected = false then
                break
        WalshCoefficients ← calculateWalshCoefficients (linkageSets)
        R² ← calculateApproximationQuality (holdoutPopulation, WalshCoefficients)
        if R² = 1 then
            return WalshCoefficients
```

LARS-BASED LINKAGE LEARNING (LARSLL)

The original algorithm is introduced in [17]. We make a straightforward modification to make its usage parameterless. After obtaining an accurate Walsh decomposition, we use GOMEA in the same way as we use it after running the ELL and the PLL algorithms. The basic idea of LARSLL is to consider the task of learning Walsh coefficients as a supervised machine learning problem. If there are solutions $x_1,\dots,x_n$ and corresponding fitness values $y_1,\dots,y_n$, each solution x_i with fitness value y can be represented using Walsh coefficients $w_{k_1},\dots,w_{k_S}$ as $y_i = w_{k_1}\psi_{k_1}(x)+\dots+w_{k_S}\psi_{k_S}(x)$. In machine learning terms, ψ are the features, y are the target values, and w are the weights to be learned. Weights w should be learned by the minimization of the Mean Squared Error loss function: $MSE = \sum_{i=1}^{n}(\hat{y}-y)^2$, where $\hat{y}$ are approximate values calculated using an estimation of weights $\hat{w}$: $\hat{y} = \hat{w}_{k_1}\psi_{k_1}(x)+\dots+\hat{w}_{k_S}\psi_{k_S}(x)$. If a function is $k-$bounded, then many of the Walsh coefficients are zero. Therefore, it is reasonable to use linear regression with $L1$ regularization to learn coefficients as such a regression model naturally forces part of the weights to be zero. As shown in [17], the regularized Least Angle Regression (LARS) is an efficient approach. Note, that in contrast to the above-mentioned algorithms, this algorithm aims at approximating Walsh coefficients as accurately as possible, instead of learning the precise values of them by discovering the full linkage structure of a problem beforehand.

In our implementation, we automatically select both the value of k and the required number of solutions in the training set for regression. The full algorithm description is provided in Algorithm 3.3.4. Starting from a population of size 1, we iteratively double the population size. For each population size, the algorithm starts with masks of order 1. The order of masks is gradually increased until the total number of masks becomes larger than the population size. We stop there because our preliminary experiments show that LARS regression is not capable of accurately learning the coefficients if the number of masks (i.e., the features) is larger than the population size (i.e., the number of samples). For each considered maximal order k of masks, all possible masks of orders $1,\dots,k$ are generated (by the function *generateMasksUpToOrder*), and Walsh coefficients are estimated by the LARS regression solver. The algorithm stops working when the quality of approximation R^2 measured on a holdout set of solutions is $\geq R^2-\epsilon$. Such $\epsilon-$allowed deviation from 1 is introduced for numerical reasons, with $\epsilon = 10^{-5}$.

Algorithm 3.3.4: The LARSLL algorithm.

```
Function LARSBasedLinkageLearning(holdoutPopulation):
    population ← generatePopulationOfSize (1)
    while true do
        maxOrder ← 1
        while masks.size < population.size do
            masks ← generateMasksUpToOrder (maxOrder)
            weights ← learnRegression (masks, population)
            R² ← calculateApproximationQuality (holdoutPopulation, weights)
            if R² ≥ 1 − ε then
                return weights
            maxOrder ← maxOrder + 1
        population ← population ∪ generatePopulationOfSize (population.size)
```

3.3.4. HOLDOUT POPULATION SIZE

In all considered algorithms, the holdout population size is important for a proper evaluation of the obtained Walsh coefficients-based surrogate model. Strictly speaking, to guarantee, that the quality of the obtained Walsh decomposition is indeed perfect, i.e., all dependencies between variables are captured and the Walsh decomposition values $\sum_{s\in\{0,1\}^\ell} w_s\psi_s(x) = f(x)\ \forall x \in \{0,1\}^\ell$, all possible combinations of bits values for all subfunctions should be present in the holdout population. Namely, if there is a subfunction depending on $x_{i_1},\ldots,x_{i_k}$ variables, in a holdout population there should be a corresponding solution for any of the 2^k possible values of $(x_{i_1},\ldots,x_{i_k})$. The probability of all possible bits assignments for all subfunctions occurring in the holdout population depends on the number of subfunctions and their order. However, in a real application, we do not know in advance these parameters of the optimized function. Thus, a proper manual setting of the holdout population size seems unrealistic. To solve this issue, we propose to use a high-level wrapper for all the considered algorithms with a scheme of a gradually increasing holdout population size.

Algorithm 3.3.5: The high-level wrapper for surrogate-assisted optimization algorithms based on Walsh decomposition.

```
Function runAlgorithm():
    holdoutPopulationSize ← ℓ
    T ← initial time limit // arbitrary initial time limit
    WalshCoefficients ← NULL
    elitist ← NULL
    while true do
        holdoutPopulation ← generatePopulationUpToSize (holdoutPopulationSize)
        R² ← calculateApproximationQuality (holdoutPopulation, WalshCoefficients)
        if WalshCoefficients = NULL or R² < 1 − ε then
            WalshCoefficients ← WalshCoefficientsCalculation (holdoutPopulation)
            // e.g., using ELL
        s ← runOptimization (T, WalshCoefficients) // Optimizing the Walsh decomposition
            with time limit T
        if fitness (s) > fitness (elitist) then
            elitist ← s
        holdoutPopulationSize ← 2 holdoutPopulationSize
        T ← 2T
    return elitist
```

First, a holdout population of fixed size is generated. The Walsh coefficients calculation *WalshCoefficientsCalculation* (e.g., by the *ELL*, the *PLL*, or the *LARSLL* algorithms) is then performed obtaining the Walsh decomposition with perfect quality for the current *holdoutPopulation*. Next, GOMEA, with time limit T, is applied to optimize it (in *runOptimization(T, WalshCoefficients)* function call). Then, the holdout population size is doubled and the learning of Walsh coefficients followed by running the optimization algorithm with a double time limit $2T$ is repeated. The best so far found solution (*elitist*) is tracked. This scheme is described in Algorithm 3.3.5. Basically, the algorithm can continue working forever, as better solutions can be found with large holdout population sizes (as new dependencies between variables can be discovered). In practice, it can be shut down by a time limit or when an a priori known optimal value is found. In the

experiments, we set the size of the initial population *holdoutPopulationSize* equal to the number of variables ℓ.

3.4. EXPERIMENTS

3.4.1. BENCHMARK PROBLEMS

We consider a set of well-known pseudo-Boolean benchmark functions. Each needs to be maximized. All functions on which we test the algorithms are additively decomposable, have blocks of dependent variables up to size k (k is different for different problems), and have the number of subfunctions $\mathcal{O}(\ell)$.

The first benchmark functions are the well-known concatenated deceptive traps [12] of orders $k = 4, 6, 8$. We also consider the circularly overlapping deceptive traps function with $k = 4, 6$. The number of subfunctions in a separable Trap function is $\frac{\ell}{k}$, in the version with overlapping subfunctions it is ℓ.

$$f_{TrapK}(x) = \sum_{i=1}^{\ell/k} f^{sub}_{TrapK}\left(\sum_{j=1}^{k} x_{ik+j}\right),$$

$$f_{TrapKOverlapping}(x) = \sum_{i=1}^{\ell} f^{sub}_{TrapK}\left(\sum_{j=1}^{k} x_{(i+j)\%(\ell+1)}\right),$$

$$f^{sub}_{TrapK}(u) = \begin{cases} k & \text{if } u = k, \\ k-1-u & \text{otherwise.} \end{cases}$$

Another function we consider is the NK landscapes with maximum overlap (also called NK-S1 landscapes) [11] with blocks of $k = 3, 5$. Corresponding problems are further referred to as $N3 - S1$ and $N5 - S1$. Similarly to overlapping deceptive traps, this function contains overlapping blocks of dependent variables, but with subfunctions depending on the block position:

$$f_{NK}(x) = \sum_{i=1}^{l-k} f^{sub}_{NK}(x_{(i,i+1,\ldots,i+k)}),$$

where the values of f^{sub}_{NK} are tabular values, samples from the uniform distribution in $[0; 1]$ interval.

Another benchmark function we consider is the MaxCut problem. Given a weighted undirected graph (V, E) the goal is to find a partition of the vertices in two sets such that the sum of weights of edges running between vertices in different partitions is maximized. For a binary GA, this problem can be encoded as a binary vector of size $|V|$ ($\ell = |V|$), where a 0 or a 1 in position i means that the i-th vertex belongs to the first and the second partition, respectively. The considered MaxCut instances are 2D square lattice graphs consisting of ℓ rows, with ℓ vertices in each row. Each of the inner vertices is connected to 4 neighbors. Edges weights are integer values from $[1, 5]$. The number of edges in these instances is $2\sqrt{\ell}(\sqrt{\ell} - 1) = \mathcal{O}(\ell)$.

Finally, we do experiments with a real-world NP-hard problem, MAXSAT. MAXSAT is an optimization version of a well-known boolean satisfiability problem (SAT). Particularly, we consider uniformly random MAX3SAT problem instances commonly used in Evolutionary

Algorithms benchmarking. These instances are defined in conjunctive normal form (CNF) with 3 variables in each clause. The fitness function in the case of the MAXSAT problem is given by:

$$f_{MAXSAT}(x) = -\sum_{i=0}^{m-1} \neg(\vee_{j=0}^{p_i-1} x_{f_{i_j}} S_{f_{i_j}}),$$

where p_i is the subfunction size (for MAX3SAT $p_i = 3\ \forall i$), f_i determines which variables are contained in the clause (subfunction) with index i, and S can be either a negation operator (turning a binary x into its opposite value) or keeping the value of x intact. The considered MAXSAT instances contain $\mathcal{O}(\ell)$ subfunctions and are known to have the global optimum with value 0 (i.e. a satisfying solution exists). For all considered MAXSAT instances the coefficient $\frac{m}{\ell} \approx 4$ which means that these instances are likely of substantial complexity.

3.4.2. IMPLEMENTATION DETAILS

We use C++ to implement all algorithms. In the LARSLL algorithm, we implement the regression model using the MLPack library [22] with OpenMP parallelization support. In all algorithms, all evaluated solutions are cached, thus the number of evaluations in our experiments means the number of different evaluated solutions.

The source code of all considered algorithms and conducted experiments is released online[1].

3.4.3. EXPERIMENTS DESIGN

For each problem, we consider several dimensionalities. We run experiments for $NK-S1$ and overlapping traps with up to 1280 variables, for separable traps with up to 1920 variables, MaxCut with up to 1600 variables, and MAXSAT with up to 100 variables (considering larger MAXSAT instances are not computationally feasible as it is an NP-hard problem and therefore the number of function evaluations needed to find the optimum by GOMEA is growing exponentially).

In our experiments, we compare the performance of the proposed ELL-GOMEA, the standard GOMEA, PLL-GOMEA, and LARSLL-GOMEA. For all problems and all considered dimensionalities, we do 30 runs. We study the scalability of the considered algorithms in terms of the number of function evaluations (of the original fitness function) and the total wall clock time to achieve the optimum. An algorithm is considered to have successfully solved a problem if it achieves an optimum in all 30 runs. For GOMEA a time limit is set to 1 hour. For other algorithms the time limit of 1 hour was set for the linkage learning parts only (not including the following optimization by GOMEA) as we suppose, that function evaluations of the initial functions may be substantially more expensive than the evaluation of the Walsh decomposition surrogate.

In the scheme described in Section 3.3.4, the initial time limit for GOMEA is set to the maximal time GOMEA needs to solve this problem instance. Though the outer loop in Algorithm 3.3.5 can be repeated forever, as a larger holdout population size can help to reveal more linkage sets and, therefore, find a better solution during optimization, in

[1] https://github.com/ArkadiyD/ELL-GOMEA

the experiments we terminate the overall execution of the instance when an optimum is found.

3.4.4. RESULTS

The scalability plots showing the comparison between GOMEA, ELL-GOMEA, PLL-GOMEA, and LARSLL-GOMEA are presented in Figure 3.4.2. They demonstrate the superior performance of ELL-GOMEA in all considered problems. The scalability graphs of PLL-GOMEA show the expected $\mathcal{O}(\ell^2)$ complexity and, therefore, it is not competitive with ELL-GOMEA. LARSLL-GOMEA, as expected, fails to scale well on problems with $k \geq 3$. On MaxCut it scales similar to PLL-GOMEA (as it is supposed to have $\mathcal{O}(\ell^k)$ complexity, which is $\mathcal{O}(\ell^2)$ in case of the MaxCut), but worse than ELL-GOMEA. The gap in performance between GOMEA and ELL-GOMEA is the largest on NK-S1 and Overlapping Trap6 problems. For the NK-S1 problem, it is easier to learn linkage with ELL-GOMEA as subfunctions are non-linear everywhere and the *findLinkage* routine from Algorithm 3.3.2 detects linkage for any pair of different solutions. Traps require more checks to detect all dependencies, as deceptive traps subfunctions are linear except for the jump to the all-ones solution. In general, such subfunctions are the most difficult for linkage learning by the algorithm used in ELL-GOMEA. The overlapping trap is even more difficult as it contains more subfunctions. Nevertheless, the scalability of ELL-GOMEA is better than the scalability of GOMEA and, therefore, for large dimensionalities, ELL-GOMEA requires fewer evaluations (of the original function) to achieve the optimum. Similar behavior can be observed for the MAXSAT problem.

Scalability of the considered algorithms in terms of total wall clock time is shown in Figure 3.4.5. As expected, the total wall clock time required by ELL-GOMEA algorithm to find an optimum is larger than for GOMEA, as it includes the linkage detection part and at least one GOMEA run for surrogate model optimization. Interestingly, for most problems (Overlapping Traps, Trap4, MaxCut, NK-landscapes, and MAXSAT) the difference between ELL-GOMEA and GOMEA becomes smaller as the number of variables grows. Also, we see that the total wall clock time of ELL-GOMEA algorithm scales polynomially. Note, that as ELL-GOMEA requires fewer evaluations of the original function, it can achieve the global optimum faster than GOMEA if the evaluations are expensive.

We do statistical tests to verify that ELL-GOMEA requires fewer evaluations than GOMEA to achieve the optimum. We use Mann-Whitney U test with $\alpha = 0.05$ and Bonferroni correction. The superior performance of ELL-GOMEA over GOMEA is statistically significant for all considered dimensions for Overlapping Trap6, Trap4, Trap6, and N5-S1 Landscape; for Overlapping Trap4 for $\ell \geq 160$; for Trap8 for $\ell \geq 240$; for N3-S1 for $\ell \geq 40$; for MaxCut for $\ell \geq 49$; for MAXSAT for $\ell = 100$. ELL-GOMEA performance inferiority compared to GOMEA is statistically significant only for Overlapping Trap4 with $\ell = 40$, Trap8 with $\ell = 40$, and MAXSAT with $\ell = 20$.

We also study how ELL-GOMEA performance depends on subfunctions size k. For this purpose, we perform experiments on concatenated separable traps of sizes 4,6,8 and overlapping traps of sizes 4,6 and plot the scalability lines in one graph for each problem type. The graphs are presented in Figure 3.4.1. These results demonstrate that the scalability lines for one problem with different values of k are approximately parallel, which means that in asymptotic complexity the k is involved only as a constant factor.

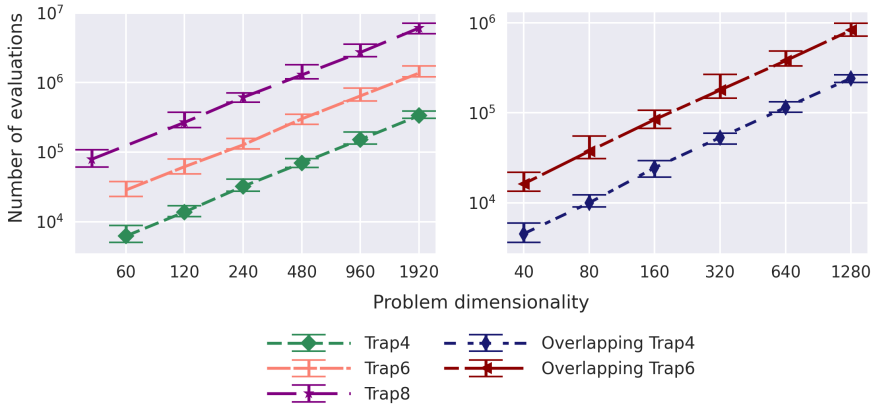


Figure 3.4.1. Scalability of ELL-GOMEA on the problems of the same type but with different subfunctions size k. The bars show the intervals of the number of evaluations, excluding the lowest and highest values (which corresponds to ≈ 93% interval). The markers indicate the median values.

We also study how the theoretically inferred complexity of ELL matches with the experimental results. First, we check how the required number of *detectLinkagePairsBinarySearch* calls for each variable depends on ℓ. It is reasonable to make this check for trap functions, as they represent the worst-case functions for ELL. The results are presented in Figure 3.4.3. It can be seen, that the required number of linkage checks corresponds to the theoretical estimate of $\mathcal{O}(log(\ell))$ obtained in Section 3.3.2.

Then, we consider the overlapping Trap4 and Trap6 problems and analyze the algorithm complexity obtained in the experiments. Particularly, we fit the function $a\ell log\ell + b$ (a, b are real values) to the experimental number of function evaluations for each considered problem size. The results are presented in Figure 3.4.4. The R^2 coefficients of the fitted curves are > 0.999. Note, that these scalability plots represent the total number of evaluations, including the calculation of Walsh coefficients and evaluations of solutions in holdout populations.

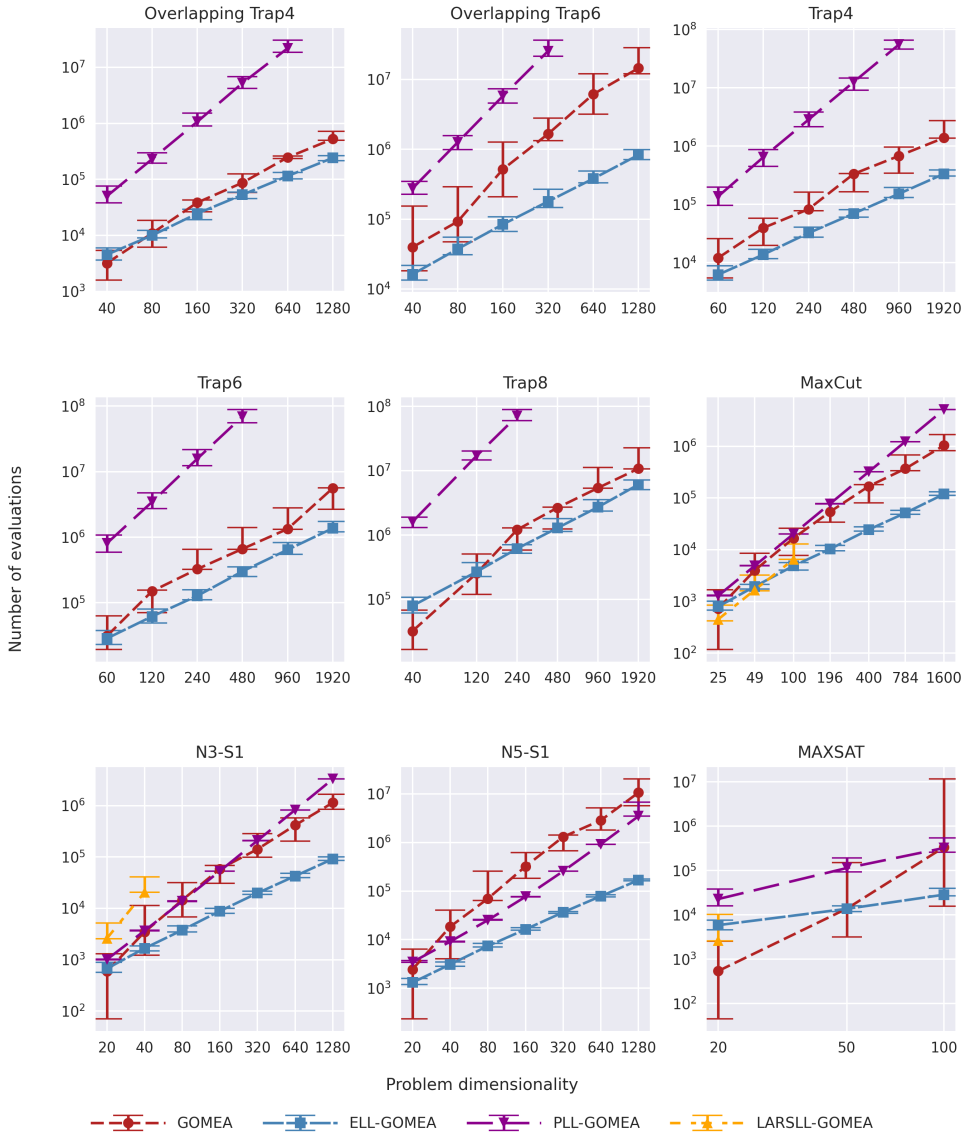


Figure 3.4.2. Scalability of the considered algorithms in terms of the required number of unique evaluations of the original fitness function to reach the optimum, summarized over 30 runs. The information about a particular problem instance is presented for an algorithm if it finds an optimum in all 30 runs. The bars show the intervals of the number of evaluations, excluding the lowest and highest values (which corresponds to ≈ 93% interval). The markers indicate the median values.

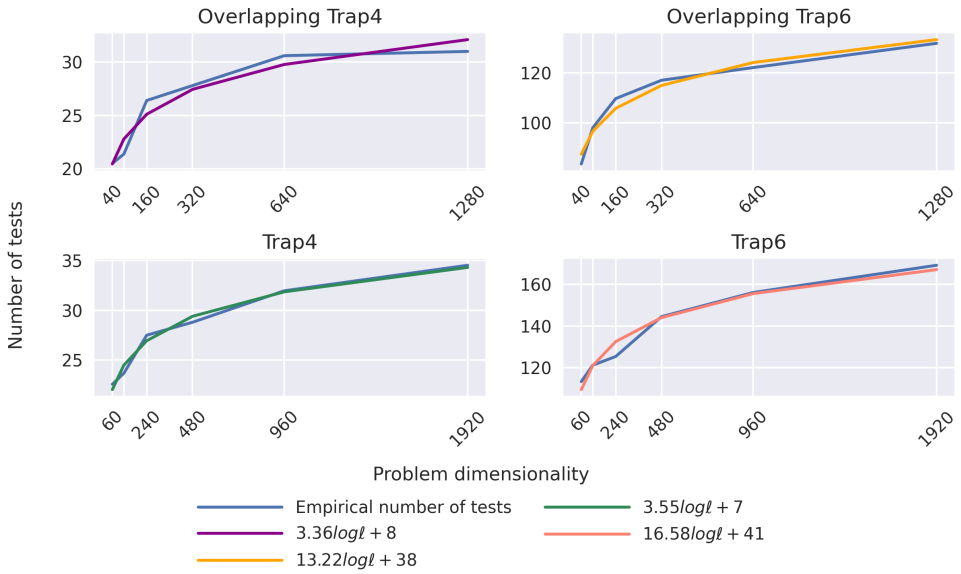


Figure 3.4.3. The empirical number of linkage tests for each variable in the ELL required to fully reveal the linkage structure. The values are averaged over 30 runs. The fittest curves $alog(\ell)+b$ are shown, and the coefficients are provided in the legend. R^2 coefficients of the fittings are >0.95 suggesting that the theoretical number of required tests $\mathcal{O}(log\ell)$ (shown in lines of blue color) matches with the empirical one.

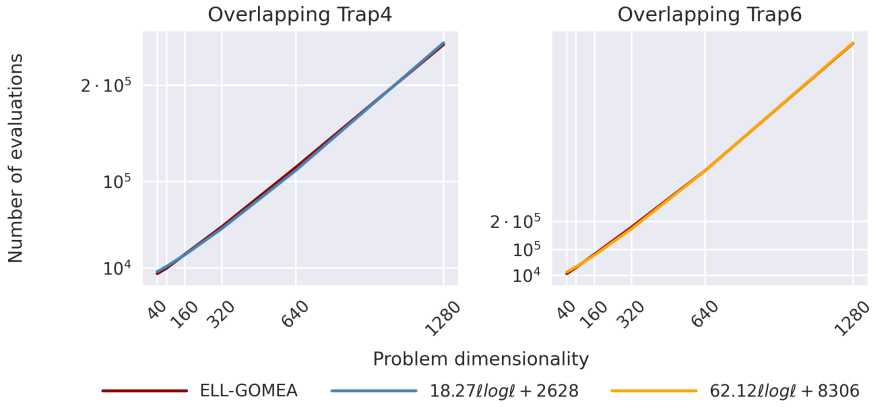


Figure 3.4.4. The curve $a\ell log\ell+b$ given by the theoretical complexity of $\mathcal{O}(\ell log\ell)$ fitted to the actual number of evaluations. The values of the empirical results are averaged over 30 runs. The coefficients a and b of the fitted curves are shown in the legend. R^2 coefficients of the fittings are >0.999 suggesting that the theoretical complexity matches with the empirical one.

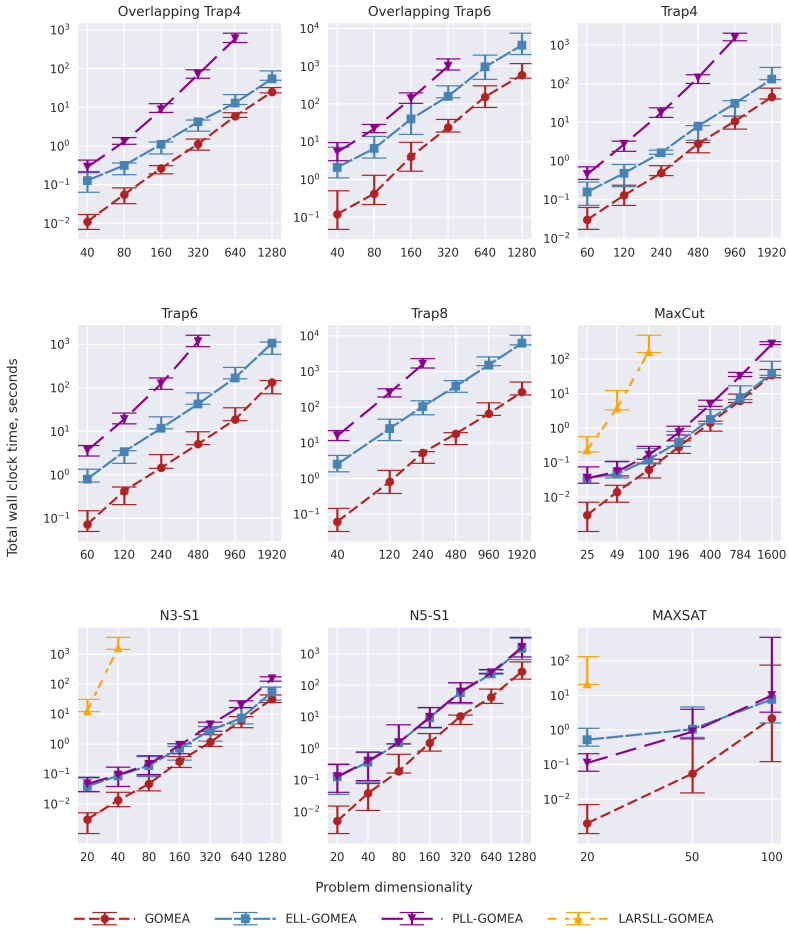


Figure 3.4.5. Scalability of the considered algorithms in terms of the total wall clock time until an optimum is found, summarized over 30 runs. The information about a particular problem instance is presented for an algorithm if it finds an optimum in all 30 runs. The bars show the intervals of the wall clock time excluding the lowest and highest values (which corresponds to ≈ 93% interval). The markers indicate the median values.

3.4.5. DISCUSSION

In this chapter, we focused on finding the global optimum of a problem instead of finding good, near-optimal solutions. For real-world problems the global optimum is often not known or hardly achievable, therefore finding just good solutions will suffice. However, in this chapter, we want to focus on the algorithm design, analysis, and experiments for the scenario when the global optimum needs to be found. Such experimental setup is common for Evolutionary Algorithms research. In fact, this makes our contribution in this chapter much like any other attempt at more efficient black-box optimization

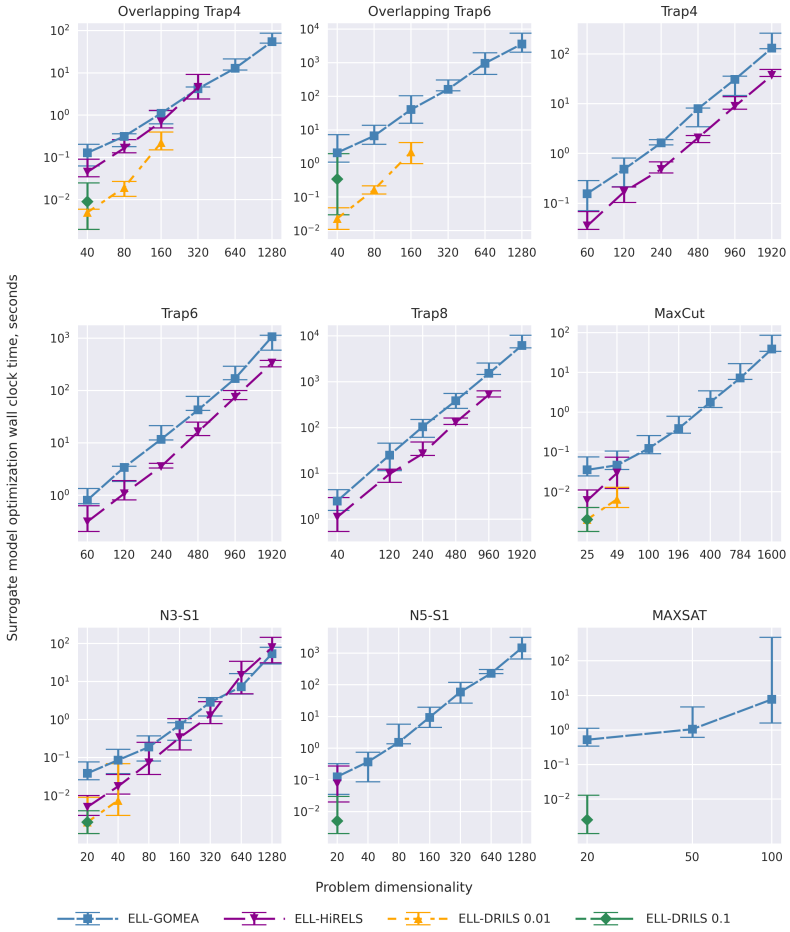


Figure 3.4.6. Comparison of GOMEA and gray-box optimization algorithms for Walsh decomposition surrogate model optimization. Walsh decomposition is obtained by the ELL algorithm. Scalability is shown in terms of wall clock time taken by an algorithm to optimize surrogate model (find the optimum) excluding time for obtaining Walsh decomposition, summarized over 30 runs. The information about a particular problem instance is presented for a gray-box algorithm if it finds an optimum within the same time limit which was used for GOMEA. The bars show the intervals of the wall clock time excluding the lowest and highest values (which corresponds to $\approx 93\%$ interval). The markers indicate the median values.

performance with a GA, except that we use a special type of surrogate model rather than the more common focus of a better variation operator. Investigating algorithm behavior and performance in a setup when finding near-optimal solutions is good enough would be more in-line with typical surrogate-assisted model-based GA research. However, this is considered beyond the scope of this chapter, yet a genuinely interesting question for

future work. Another left-out research question that is closely related to the previous one is applying the introduced algorithm in the case when the perfect Walsh decomposition model cannot be obtained (e.g. function is not k-bounded or function evaluations are noisy) or the function evaluations are so expensive that it is not feasible. For such a case, a special study of stopping criteria based on surrogate model quality is needed.

In our experiments, we used GOMEA as the optimization algorithm for the obtained surrogate model (the Walsh decomposition). As our experiments show, especially when combined with ELL, we can solve various problems with a smaller number of function evaluations. However, we realize that such an approach does not help to solve problems which GOMEA fails to solve in reasonable time. For instance, in our preliminary experiments, we found out that random NK-landscapes with k=5 (also called embedded landscapes [23]) and sums of traps with randomly selected variable positions for each trap subfunction (embedded landscapes with deceptive subfunctions) are intractable for GOMEA: it fails to find an optimum even for a moderate number of variables (≥ 50). Overlapping Trap8 for $\ell \geq 320$ and MAXSAT for $\ell \geq 200$ also cannot be solved by GOMEA within 1 hour. However, potentially the class of problems solvable by ELL-GOMEA can be extended if a more powerful optimization algorithm is developed and used. Recently, a novel gray-box algorithm was introduced in [24], but it focuses on more simple NK landscapes instances, with subfunctions values selected from integer values from 0 to Q, where Q is moderate (< 100). We tried to replace GOMEA as the surrogate model optimizer with two gray-box algorithms, namely, Hierarchical Recombinative Local Search (HiReLS) and Deterministic Recombination and Iterated Local Search (DRILS) [24] with different values of mutation probability. As we assume that the real function evaluations are much more expensive than the surrogate ones, we are interested in finding the optimum of the surrogate model as fast as possible regardless of the number of surrogate evaluations required. We test the ability of the HiRELS and DRILS algorithms to solve a problem instance faster than GOMEA. The graphs with these results are shown in Figure 3.4.6. They show that GOMEA is generally preferable to HiRELS and DRILS. DRILS cannot solve considered problems faster than GOMEA. HiRELS is faster than GOMEA on separable Traps and moderate-size instances of N3-S1 but GOMEA is faster and scales better on other problems which are known to be more difficult to solve (MAXSAT, Overlapping Traps, N5-S1). Therefore, we stick to GOMEA as a surrogate model optimizer. Nevertheless, designing an efficient gray-box algorithm for solving difficult, currently intractable for GOMEA, problems is still highly desirable.

Another important question is real-world applications of the proposed algorithm. So far not many works address the question of designing k-bounded fitness functions for real-world problems. One of such examples is a new fitness function for clustering [25], but it does not demonstrate superior results on clustering benchmarks. In this chapter, we showed, that the function property of being k-bounded can be efficiently exploited in a black-box setting, and, therefore, designing novel k-bounded fitness functions for real-world optimization problems becomes an important question of the future work. Another future work subject is extending the proposed approach to non-bounded functions, including hierarchical ones, e.g., the well-known Hierarchical If-And-Only-If benchmark function (HIFF), which currently the proposed approach cannot solve (but GOMEA itself can).

3.5. CONCLUSION

We introduced a novel approach to binary surrogate-assisted genetic algorithms in a black-box setting. We consider a precise Walsh decomposition of the optimized function as the surrogate. To efficiently calculate Walsh coefficients, we propose to use an efficient algorithm for linkage learning first and then calculate non-zero coefficients, i.e., coefficients corresponding to sets of dependent variables. After obtaining an accurate Walsh decomposition, we consider it as a surrogate and use GOMEA to optimize it without any additional evaluations of the initial function.

In our experiments on the well-known benchmark functions, we compare the proposed approach with standard GOMEA, and with two different algorithms for approximating Walsh coefficients followed by optimization using GOMEA. The set of benchmark functions includes separable and overlapping traps, adjacent NK-landscapes, MaxCut, and the NP-hard problem MAXSAT. The proposed algorithm shows better polynomial scalability in terms of original fitness function evaluations than all considered alternatives.

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO). We thank the Maurits en Anna de Kock Foundation for financing a high-performance computing system.

APPENDIX

DERIVATION A

Inequality to solve: $((1-(1-p)^{\frac{N}{J}})^{J})^{\ell} > \delta$, variable: ℓ, constants: $N, J > 0; 0 < p, \delta < 1$

$((1-(1-p)^{\frac{N}{J}})^{J})^{\ell} > \delta$

$(1-(1-p)^{\frac{N}{J}})^{J\ell} > \delta$

$1-(1-p)^{\frac{N}{J}} > \delta^{\frac{1}{J\ell}}$

$(1-p)^{\frac{N}{J}} < 1-\delta^{\frac{1}{J\ell}}$

$log_{1-p}((1-p)^{\frac{N}{J}}) > log_{1-p}(1-\delta^{\frac{1}{J\ell}})$ (the inequality sign is changed as $0 < 1-p < 1$)

$\frac{N}{J} > log_{1-p}(1-\delta^{\frac{1}{J\ell}})$

$N > J log_{1-p}(1-\delta^{\frac{1}{J\ell}})$

$N > \frac{J}{ln(1-p)} ln(1-\delta^{\frac{1}{J\ell}})$

$N > C ln(1-\delta^{\frac{1}{J\ell}}) = C ln(1-\lambda^{\frac{1}{\ell}})$,where C, λ are constants ,$0 < \lambda < 1$.

DERIVATION B

Goal: prove that $\mathcal{O}(\ln(1-\lambda^{1/\ell})) = \mathcal{O}(log\ell)$ if $0 < \lambda < 1$

Problem reformulation: find $\lim_{\ell \to +\infty} \frac{ln(1-\lambda^{1/\ell})}{ln(\ell)}$

As $0 < \lambda < 1$, $\lim\limits_{\ell \to +\infty} ln(1-\lambda^{1/\ell}) = -\infty$

Moreover, $\lim\limits_{\ell \to +\infty} ln(\ell) = +\infty$

Because of (1) and (2), L'Hôpital's rule can be applied:

$$\lim_{\ell \to +\infty} \frac{ln(1-\lambda^{1/\ell})}{ln(\ell)} = \lim_{\ell \to +\infty} \frac{\frac{\lambda^{1/\ell} ln\lambda}{\ell^2(1-\lambda^{1/\ell})}}{\ell^{-1}} = \lim_{\ell \to +\infty} \frac{\lambda^{1/\ell} ln\lambda}{\ell(1-\lambda^{1/\ell})}$$

Substitution: $y = \lambda^{1/\ell}, \ell = \frac{1}{log_\lambda y}$

$$\lim_{\ell \to +\infty} \frac{\lambda^{1/\ell} ln\lambda}{\ell(1-\lambda^{1/\ell})} = \lim_{y \to 1} \frac{y ln\lambda}{\frac{1}{log_\lambda y}(1-y)} = \lim_{y \to 1} \frac{y ln\lambda log_\lambda y}{1-y} = \lim_{y \to 1} \frac{y ln y}{1-y}$$

L'Hôpital's rule can be applied for the second time as $\lim\limits_{y \to 1} y ln y = \lim\limits_{y \to 1}(1-y) = 0$

Applying L'Hôpital's rule for the second time:

$$\lim_{y \to 1} \frac{y ln y}{1-y} = \lim_{y \to 1} \frac{ln y + 1}{-1} = -1$$

Thus, $\mathcal{O}(ln(1-\lambda^{1/\ell})) = \mathcal{O}(ln\ell) = \mathcal{O}(log\ell)$

REFERENCES

[1] A. Diaz-Manriquez, G. Toscano Pulido, J. Barron-Zambrano, and E. Tello. “A review of surrogate assisted multiobjective evolutionary algorithms”. In: *Computational Intelligence and Neuroscience* 2016 (June 2016), pp. 1–14.

[2] D. Thierens and D. E. Goldberg. “Mixing in Genetic Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers Inc. 1993, pp. 38–47.

[3] D. Thierens and P. A. N. Bosman. “Predetermined versus learned linkage models”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2012, pp. 289–296.

[4] M. Pelikan and H. Muehlenbein. “The Bivariate Marginal Distribution Algorithm”. In: *Advances in Soft Computing*. Ed. by R. Roy, T. Furuhashi, and P. K. Chawdhry. London: Springer London, 1999, pp. 521–535.

[5] F. G. Lobo and D. E. Goldberg. “The parameter-less genetic algorithm in practice”. In: *Information Sciences* 167.1 (2004), pp. 217–232.

[6] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. “BOA: the Bayesian Optimization Algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Orlando, Florida: Morgan Kaufmann Publishers Inc., 1999, pp. 525–532.

[7] D. Thierens. “The Linkage Tree Genetic Algorithm”. In: *International Conference on Parallel Problem Solving from Nature*. Ed. by R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph. 2010, pp. 264–273.

[8] P. A. N. Bosman and D. Thierens. “Linkage neighbors, optimal mixing and forced improvements in genetic algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 585–592.

[9] M. Zaefferer et al. “Efficient global optimization for combinatorial problems”. In: *Proceedings of the 2014 annual conference on genetic and evolutionary computation*. 2014, pp. 871–878.

[10] R. B. Heckendorn and A. H. Wright. “Efficient linkage discovery by limited probing”. In: *Evolutionary Computation* 12.4 (2004), pp. 517–545.

[11] M. Pelikan, K. Sastry, D. E. Goldberg, M. V. Butz, and M. Hauschild. “Performance of evolutionary algorithms on NK-landscapes with nearest neighbor interactions and tunable overlap”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2009, pp. 851–858.

[12] K. Deb and D. E. Goldberg. “Sufficient conditions for deceptive and easy binary functions”. In: *Annals of Mathematics and Artificial Intelligence* 10.4 (1994), pp. 385–408.

[13] M. Munetomo and D. E. Goldberg. *Identifying Linkage by Nonlinearity Check*. Tech. rep. 1998.

[14] M. Munetomo and D. E. Goldberg. “Linkage Identification by Non-monotonicity Detection for Overlapping Functions”. In: *Evolutionary Computation* 7.4 (1999), pp. 377–398.

[15] M. J. Streeter. “Upper bounds on the time and space complexity of optimizing additively separable functions”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Ed. by K. Deb. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 186–197.

[16] S.-S. Choi, K. Jung, and J. H. Kim. “Almost tight upper bound for finding Fourier coefficients of bounded pseudo-Boolean functions”. In: *Journal of Computer and System Sciences* 77.6 (2011), pp. 1039–1053.

[17] S. Verel, B. Derbel, A. Liefooghe, H. Aguirre, and K. Tanaka. “A surrogate model based on Walsh decomposition for pseudo-Boolean functions”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 181–193.

[18] F. Leprêtre, S. Verel, C. Fonlupt, and V. Marion. “Walsh functions as surrogate model for pseudo-boolean optimization problems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 303–311.

[19] N. H. Luong, H. La Poutré, and P. A. N. Bosman. “Multi-objective gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start Scheme”. In: *Swarm and Evolutionary Computation* 40 (2018), pp. 238–254.

[20] D. Thierens and P. A. N. Bosman. “Optimal mixing evolutionary algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2011, pp. 617–624.

[21] H. Kargupta and B. Park. “Gene expression and fast construction of distributed evolutionary Representation”. In: *Evolutionary Computation* 9 (2001), pp. 43–69.

[22] R. R. Curtin et al. “MLPACK: A scalable C++ machine learning library”. In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 801–805.

[23] R. B. Heckendorn. “Embedded Landscapes”. In: *Evolutionary Computation* 10.4 (2002), pp. 345–369.

[24] F. Chicano, D. Whitley, G. Ochoa, and R. Tinós. “Optimizing one million variable NK landscapes by hybridizing deterministic recombination and local search”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 753–760.

[25] R. Tinós, Z. Liang, F. Chicano, and D. Whitley. “A new evaluation function for clustering: The NK internal validation criterion”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 2016, pp. 509–516.

4

CONVOLUTIONAL NEURAL NETWORK SURROGATE-ASSISTED GOMEA

We introduce a novel surrogate-assisted Genetic Algorithm (GA) for expensive optimization of problems with discrete categorical variables. Specifically, we leverage the strengths of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA), a state-of-the-art GA, and, for the first time, propose to use a convolutional neural network (CNN) as a surrogate model. We propose to train the model on pairwise fitness differences to decrease the number of evaluated solutions that is required to achieve adequate surrogate model training. In providing a proof of principle, we consider relatively standard CNNs, and demonstrate that their capacity is already sufficient to accurately learn fitness landscapes of various well-known benchmark functions. The proposed CS-GOMEA is compared with GOMEA and the widely-used Bayesian optimization-based expensive optimization frameworks SMAC and Hyperopt, in terms of the number of evaluations that is required to achieve the optimum. In our experiments on binary problems with dimensionalities up to 400 variables, CS-GOMEA always found the optimum, whereas SMAC and Hyperopt failed for problem sizes over 16 variables. Moreover, the number of evaluated solutions required by CS-GOMEA to find the optimum was found to scale much better than GOMEA.

The contents of this chapter are based on the following publication: **A. Dushatskiy**, A. M. Mendrik, T. Alderliesten, and P. A. N. Bosman. "Convolutional neural network surrogate-assisted GOMEA ". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 753–761.

4.1. INTRODUCTION

Optimization problems with time-consuming objective function evaluations (expensive optimization) arise in different domains. With the recent progress in deep learning, expensive optimization has become a topic of great interest as there are several connected expensive optimization tasks, such as deep neural network hyperparameter optimization. In this chapter, we consider single-objective binary problems in a Black Box Optimization (BBO) setting with the assumption that function evaluations are expensive (i.e., training a deep neural network).

Many of the currently used expensive optimization algorithms are based on Bayesian Optimization (BO) and consider problems with real-valued variables [1]. The traditional BO algorithms, which use Gaussian Process models, have two major problems: a limit on the number of variables (in most works the functions in the experiments have only several variables) and difficulties with applications to problems with discrete categorical variables [2]. Two advanced algorithms commonly used for expensive optimization tasks, including applications to deep learning, are: SMAC [3] and Hyperopt [4]. SMAC has the option to use Random Forests (RF) instead of Gaussian Processes to better solve mixed integer and integer problems. Hyperopt is an implementation of the Tree-structured Parzen Estimator (TPE) [5], which is also a BO algorithm. SMAC and Hyperopt have been reported to be able to handle problems with several tens of variables [3, 4, 6] and perform better than Gaussian Process-based algorithms for discrete problems [7].

Model-based optimization (MBO) algorithms for discrete problems are increasingly a topic of great interest [1]. These algorithms rely on an accurate objective function approximation with a surrogate model. In a similar way to BO, most algorithms generate one new solution per iteration. The surrogate models reported to perform the best are Radial Basis Function Networks (RBFN) [8], Kriging [9] adapted to binary variables, and a model based on Walsh functions decomposition [10]. However, the reported working examples for binary functions are still limited to a moderate number of variables (up to 20-30) [1].

Another class of algorithms that can potentially be used for solving expensive optimization problems is surrogate-assisted Genetic Algorithms (GAs) [11]. GAs are powerful search methods, but potentially require many function evaluations. Using surrogate models is a natural way to reduce the number of real function evaluations by replacing a part of them with surrogate ones, which are considered to consume much less time. A surrogate-assisted GA, therefore, has the potential to be highly powerful for expensive optimization, retaining the competent search characteristics of the GA, while reducing overall run time through surrogate function evaluations. Note that a surrogate-assisted GA approach differs from using a GA in BO or MBO algorithms for finding the next point to evaluate. Common types of surrogate models integrated into GAs are polynomials, Kriging, RBFN, and Support Vector Regression (SVR) [11]. To the best of our knowledge, there is currently no literature on surrogate-assisted GAs for solving problems with discrete categorical variables. However, many real-world problems have discrete (binary) components, such as architectural choices, in the form of connections, of deep neural networks.

In this chapter, we consider a novel type of surrogate model, namely a Convolutional Neural Network (CNN) [12]. Though there are existing works in which artificial neural

networks are used [13], to our knowledge, no works use a CNN as the surrogate model in a GA. Here, we study the best way to realize the integration and the resulting scalability of terms of required evaluations to solve well-known benchmark problems to optimality. We consider as the baseline the Gene Pool Optimal Mixing Evolutionary Algorithm (GOMEA) [14] so as to assess the true potential added value of a surrogate model by testing it in conjunction with a state-of-the-art GA. A key feature of GOMEA is its ability to exploit problem structure in the form of linkage (i.e., interdependent problem variables). This makes GOMEA highly suited to solve complex optimization problems in a BBO setting, with excellent scalability.

4.2. CNN SURROGATE MODEL

We have chosen to use a CNN as the approximator due to its known capacity and extrapolation ability, as opposed to decision trees, for which output values are bounded by the values in the training dataset [15]. Compared to Fully-Connected neural networks, a CNN can learn the same information using much fewer parameters because filters are applied to different input data locations, essentially giving the CNN the power to learn higher-order combinations efficiently. Thus, the overfitting problem is less likely to occur.

4.2.1. PAIRWISE REGRESSION

We consider the task of fitness function approximation as a supervised machine learning regression problem. In general, the approximation procedure entails:

$$\min_f \; Loss\left([y_1,\ldots,y_N],[f(x_1),\ldots,f(x_N)]\right),$$

where f is an approximation function, $f:\{0,1\}^l \mapsto \mathbb{R}$; $x \in \{0,1\}^l$ are solutions, $y \in \mathbb{R}$ are fitness function values; *Loss* is a loss function, e.g., Mean Squared Error (MSE): $MSE(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$; N is the training dataset size. In machine learning terms, the solutions represent input features of a machine learning model and the fitness function values represent target values.

Despite the great potential of CNNs, a downside is that they need large numbers of data points for training. To alleviate this issue and make the most out of previously evaluated solutions, we propose to approximate the *difference* of fitness function values of two solutions rather than directly approximate the fitness of a solution. As input the CNN thus gets two solutions. As output, it produces the predicted difference of their fitness values. The training dataset is formed by all possible *ordered* pairs of solutions. Thus, the size of the training dataset becomes n^2, with n the number of evaluated solutions, increasing the number of training samples by an order of magnitude. This effect is demonstrated in Figure 4.2.1 for the Onemax function described in Section 4.4.1 with $l = 100$. The pairwise regression needs fewer data points (i.e., evaluations of solutions) to produce accurate predictions on the validation set (solutions in both training and validation datasets are randomly generated in this example). The training procedure is described in Section 4.2.2.

After training it is possible to predict the fitness value of an arbitrary solution s by using the solutions from the training dataset $t_1,\ldots,t_N$, where N is the training dataset

size, as pivotal solutions. Specifically, we form the pairs $(t_1, s), \ldots, (t_N, s), (s, t_1), \ldots, (s, t_N)$. For each pair, the fitness difference is predicted and thereby the fitness value estimation is calculated. The resultant fitness value estimation of s is averaged over all pairs. Moreover, it is possible to use only part of the training solutions as pivotal ones to accelerate the prediction process. The full prediction procedure is described in Algorithm 4.2.1.

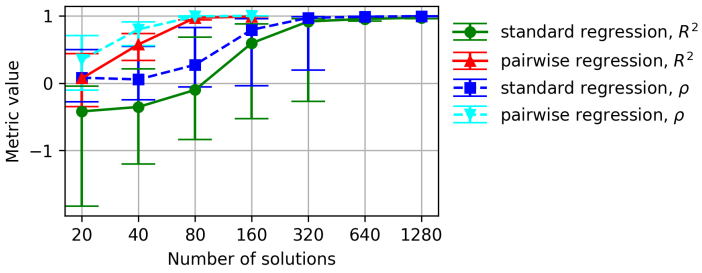


Figure 4.2.1. R^2 coefficient and Spearman correlation coefficient ρ for increasing numbers of evaluated solutions used as training samples for a CNN in case of direct regression and pairwise regression using the same internal CNN architecture. The approximated function is Onemax. For each number of solutions, data generation and training are done 30 times. The dots represent the median values, the bars indicate the $2nd$ and $29th$ order statistics.

Algorithm 4.2.1: Fitness prediction after pairwise regression.

```
Function predictFitnessValue(s):
    pivotalSolutions ← subset of train data of size maxPivotalSize
    predictions ← ∅
    for refS ∈ pivotalSolutions do
        // refFitness is the fitness of the reference solution refS
        predictions ← predictions ∪ refFitness + modelPredict(s, refS)
        predictions ← predictions ∪ refFitness − modelPredict(refS, s)
    finalPrediction ← mean (predictions)
    return finalPrediction
```

4.2.2. TRAINING PROCEDURE

Since we consider binary problems, we apply a standard simple data transformation before training: subtract 0.5 from input x to center it around 0 and divide target values y by the maximum absolute value seen in the training dataset to clip them between -1 and 1, as the initial weights of the CNN are generated randomly also from the $[-1;1]$ interval.

We train the CNN with a hold-out validation set. For the division into training and validation sets, systematic sampling is used: the target values are sorted and every qth sample is taken into the validation set. In the experiments, q is set to 5 (i.e., 80% of data is used for training, 20% for validation). After obtaining the training and validation sets, we form pairs of samples. Specifically, imitating the prediction procedure described in Section 4.2.1, the CNN is validated on pairs containing one solution from the training dataset, and one from the validation one.

One training epoch consists of processing all pairs of samples from the training set. The loss that is minimized during training is the MSE. As a training algorithm we use standard backpropagation with the gradient-descent based optimizer Adam [16], with the following parameters: $\alpha = 0.0005, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ (the standard parameters from [16], except the learning rate, which is slightly lowered to make the learning process more robust). An early stopping criterion is used for training: the training is stopped if the error on the validation pairs of solutions is not improving for several consecutive epochs. In our implementation, this number of epochs is dependent on the training dataset size: it varies from 2 for large datasets of size $N \geq 10^5$ to 6 for smaller datasets of size $N < 10^4$ (for datasets of size $10^4 \leq N < 10^5$ it is set to 4).

4.2.3. CNN ARCHITECTURE

We would like the surrogate model to capture information about the impact of blocks of dependent, both closely and distantly located, variables on the fitness values. For this purpose dilated convolutional filters [17] can be used, which are capable of capturing dependencies between distant variables. They are also capable of capturing the dependencies between tightly located variables if the dilation factor, which determines the distance between variables to which the filter is applied, is set to 1. Each filter has the following hyperparameters: the filter size k that determines the number of variables captured by the filter at once, the stride st that determines the distance between successive moves of a filter along the input, the dilation factor d and weights w that are learned during the training procedure. Given a string of length l_{in}, it produces a string of length $l_{out} = \frac{l_{in} - d(k-1) - 1}{st} + 1$. Usually, several filters with the same hyperparameters, but with independent weights are applied at once, each producing its own output. These filters with separately learned weights can capture different dependencies or aspects thereof.

Another important feature of the proposed neural network architecture is the input representation in the form of two stacked solutions. This allows the convolutional filters to naturally capture the information about differences in corresponding genes of two solutions and their connection with differences in fitness values. The working principle of dilated filters in our context is demonstrated in Figure 4.2.2.

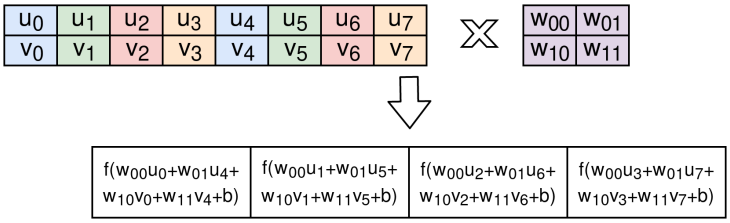


Figure 4.2.2. Example of applying a dilated filter to two stacked solutions u and v with 8 variables. Filter size $k = 2$, dilation factor $d = 4$, stride $st = 1$, w_{ij} are filters weights, b is a bias term, f is an activation function. Different colors represent the variables captured by the filter at each position while moving along the input. As a result, the filter produces a new string of length 4.

In this chapter, we consider the convolutional neural network architecture with 2 convolutional layers, an optional fully-connected (dense) hidden layer, and an output layer,

which is also fully connected. The architecture for the particular problem is optimized with the grid-search procedure [18]; more details are provided in Section 4.2.3. In the second convolutional layer, the filters are regular convolutional filters without dilation, because the first convolutional layer is supposed to already capture the dependencies between distant variables. We use C_1 and C_2 filters in the convolutional layers, respectively. Moreover, the size of a fully-connected layer is C_3. The structure of the neural network is shown in Figure 4.2.3. The activation functions of all layers, except the output one, are Rectified Linear Units (ReLUs): $f(x) = \max(0, x)$; the output layer has a linear activation function: $f(x) = x$. After the second convolutional layer, dropout [19] is applied with a common ratio of 0.2 to reduce model overfitting.

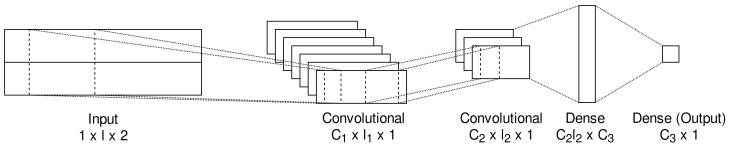


Figure 4.2.3. The CNN architecture used, with 1 hidden dense layer. l is the number of variables, l_1, l_2 are the second dimensions of tensors after processing by two convolutional layers. The input is two stacked solutions.

HYPERPARAMETER SEARCH

We perform an often-used grid search to find the best architecture and hyperparameters for a particular problem. In preliminary experiments, it was found that C_1, C_2, C_3 can be set to 100, 30, 30 respectively to provide a reasonable model complexity. These hyperparameters are not optimized during the grid search procedure.

The hyperparameters and their corresponding domains (search is performed over integer values) are presented in Table 4.2.1: k_1, st_1, d_1 are the filter size, stride size, and dilation factor size of filters in the first convolutional layer respectively; k_2 is the filter size in the second convolutional layer, N_{fc} is the number of fully-connected layers (the hidden ones). The search is performed in two stages: in the first stage the hyperparameters of the first convolutional layer are optimized, the second layer is fixed with $k_2 = 1, st_2 = 1, d_2 = 1$, and N_{fc} is fixed at 1. In the second stage, the best found hyperparameters for the first layer are fixed and the search is performed for k_2 and N_{fc} hyperparameters. The search is divided into two stages to accelerate it. We consider this division to be acceptable as the hyperparameters of the first layer have a greater impact than the other ones on capturing the problem structure and hence on the CNN accuracy. The search is terminated in case the model quality (Spearman correlation on the validation set as described in Section 4.2.4) on a set of hyperparameters is ≥ 0.97.

Parameter	k_1	st_1	d_1	k_2	N_{fc}
Interval	$[1; \min(10, \frac{l}{2})]$	$[1; \frac{l}{4}]$	$[1; \frac{f_1}{2}]$	$[1; 2f_1]$	[0,1]

Table 4.2.1. Hyperparameters and intervals of search.

4.2.4. SURROGATE MODEL QUALITY

Common metrics for assessing the performance of machine learning algorithms for solving regression problems (e.g. MSE, R^2) can be applied to assess the quality of the surrogate model. However, when using a surrogate model in a GA we are more interested in consistency between solution ranks as ranked by the real fitness values and the surrogate values than in small differences between real and predicted fitness values. Thus, we define the surrogate model quality (in all further mentions) as the Spearman correlation coefficient: $r_s(x,y) = \frac{cov(rg_x, rg_y)}{\sigma_{rg_x}\sigma_{rg_y}}$, where x and y are respectively real and surrogate fitness values of solutions in a validation set, rg_x and rg_y are their ranks among the considered set of solutions.

4.3. CONVOLUTIONAL SURROGATE-ASSISTED GOMEA

4.3.1. GOMEA

The key components of GOMEA are linkage learning, the Gene-pool Optimal Mixing (GOM) operator, the Forced Improvements (FI) procedure, and the Interleaved Multistart Scheme (IMS). We briefly outline each of these components here and refer the interested reader to the literature for more details [14, 20, 21].

LINKAGE LEARNING

The goal of linkage learning algorithms is to reveal the nature of the linkage between variables, which can often be defined in terms of subsets of dependent variables. Multiple linkage learning algorithms and corresponding linkage structures have been introduced for GOMEA. In this chapter we use the Linkage Tree (LT) [14] as the linkage structure because this structure has the ability to capture the structure of hierarchical problems and outperforms in most cases other linkage structures, as stated, e.g., in [21]. The structure obtained after linkage learning contains subsets of linked variables and is called the Family of Subsets (FOS).

GENE-POOL OPTIMAL MIXING

After learning the linkage structure, offspring solutions are generated. Offspring generation with the GOM starts with copying a parent solution. Then, the linkage structure is iterated over in random order. For each subset, a donor solution is randomly selected from the population and genes from loci in the current linkage subset are copied from donor to offspring. The change is accepted only if the fitness function value does not deteriorate.

FORCED IMPROVEMENTS

If the GOM operator has failed to improve the fitness of an offspring (after iterating over all the subsets in the linkage structure), or if the local elitist (of this population) solution has not improved for $\log_{10}(populationSize)$ generations, then the FI procedure is applied. The FI procedure performs mixing of the offspring with the global (taken over all populations) elitist solution. All subsets of the linkage structure are considered in random order. For each linkage subset, the genes from corresponding loci from the elitist solution are copied to the offspring; the change is accepted only in case of a real improvement of the fitness

value. Once an improvement occurs, the FI procedure is immediately terminated. If the offspring has not improved even after this part of FI, it is set to the elitist solution.

INTERLEAVED MULTISTART SCHEME

The IMS aims to offer a reasonable alternative to needing to set the population size parameter by hand. Several populations of increasing sizes are run in an interleaved fashion. The smallest population in the scheme is of fixed size. Here, we use a population of size 8. The next populations are double the size of the previous population. The populations are interleaved, such that for each c^{IMS} generation iterations of a population of size *populationSize*, one iteration of the population with population size $2\,populationSize$ is performed. In our implementation, c^{IMS} is set to 4. A population is terminated if a population with a larger population size has a better average objective value.

4.3.2. ADDING THE SURROGATE MODEL TO GOMEA

The CNN described in Section 4.2 is used as a surrogate model in GOMEA. To do so, function evaluations are replaced by fitness value estimations made by the surrogate model under certain conditions. Moreover, each population in the IMS has its own surrogate model.

PRELIMINARY ACTIONS BEFORE IMS LOOP

The general outline of CS-GOMEA is provided in Algorithm 4.3.1. Before starting to use the surrogate model, initial training data is collected. We call this process a *warm-up period*. After initial *warmupSize* random solutions' generation, a CNN hyperparameter search is performed. Then, more random solutions are generated until the model quality surpasses the threshold T. Re-training of the model is performed every time after *generationStep* solutions are generated. In our experiments, we set *generationStep* to 10, as a compromise between the number of model trainings and the number of evaluated solutions at each step, *warmupSize* is adjusted with respect to the problem size and difficulty as described in Section 4.4.2.

If the surrogate model has failed to achieve a quality that is above the threshold T after a maximal allowed number of solution evaluations, then, probably, the predictive ability of this model will not be sufficient to find the optimum with GOMEA if surrogate fitness is used. Thus, a special algorithm mode called a mixed populations mode is then activated: a subset of the solutions in each population will evolve separately. In these subsets, only real evaluations are used, and, moreover, in generating offspring only solutions from these subsets are used as parents to avoid any bias in solutions induced by the surrogate model. The size of this subset of the population is determined by the model quality. The lower the quality is, the larger this subset of the population must be. Hence, we used $sizeMixed = \lceil populationSize \cdot (T - modelQuality) \rceil$, but *sizeMixed* is not allowed to be larger than half the population size. In preliminary experiments, we have observed that it is reasonable to set T to 0.9.

Algorithm 4.3.1: General outline of CS-GOMEA.

```
Function Main(warmupSize):
    generateRandomSolutions(warmupSize)
    performHyperparamatersSearch(warmupSize)
    // evaluatedArchive maintains the archive of the solutions for which the real
       fitness was calculated
    while modelQuality < T and evaluatedArchive.size < 2warmupSize do
        generateRandomSolutions(generationStep)
        model ← trainModel(evaluatedArchive)
        modelQuality ← checkModelQuality(model)
    if modelQuality < T then
        mixedPopulationsMode ← true
    runIMS()
```


REAL AND SURROGATE FITNESS VALUES

As stated in other studies [22], mixing real and surrogate fitness values in a population might be an issue. This is because comparisons between real fitness values and surrogate ones are compromised, as the surrogate model does not predict the fitness values with perfect accuracy. Thus, we store only surrogate fitness values, except when in mixed populations mode. In that case, the real fitness values are stored for the subset of the population that uses only real fitness values. As each population has its own surrogate model, the fitness values in a population are the estimates made by the model belonging to this population. Also, we maintain surrogate fitness values for each local elitist solution in each population. A global elitist solution for all populations is furthermore maintained, for which only the real fitness is stored.

CS-GOMEA POPULATION LOOP

The main loop of the optimization process in CS-GOMEA is listed in Algorithm 4.3.2 [1].

Algorithm 4.3.2: Main CS-GOMEA loop.

```
Function runPopulation(populationIndex):
    populationSize ← basePopulationSize · 2^populationIndex
    P ← initializePopulation(populationSize)
    while population not terminated do
        learnFOS(P)
        generateOffspring(P)
        trainModel(randomArchive ∪ ⋃_{k=0}^{populationIndex} optArchive_k)
        updateSurrogateValues(P)
        for s ∈ P do
            doRealEvaluation(s)
```

One iteration over the population includes creating the offspring solutions, retraining the model, updating the surrogate values of the population, and performing real evaluations for offspring solutions to check whether there is an improvement of the elitist. Using the LT, one application of the GOM operation potentially requires $2l-2$ evaluations, as

[1] In the IMS, no instance of GOMEA is run exactly as outlined in Algorithm 4.3.2. Rather, only c^{IMS} generations are done every time an instance is called upon (without re-initialization).

this is the number of internal nodes in an LT, see [21]. Hence, evaluating all solutions fully at the end of a generational cycle requires a minor fraction of $\approx \frac{N}{N(2l-2)} = \frac{1}{2l-2}$ real evaluations (in non-mixed populations mode), thus potentially increasing the efficiency of the CNN integration with problem size. During the offspring generation described in Algorithms 4.3.3, 4.3.4 and 4.3.5 there are several cases when a real evaluation can be executed. Here, we describe them in detail, along with underlying ideas.

1. *If the surrogate value of the solution is the new elitist surrogate value.*
   The reason for doing a real evaluation, in this case, is the assumed correlation between the model predictions and the real fitness values. Because of this, the new surrogate elitist has a good chance to be a real elitist solution, too.

2. *If the real elitist solution has not been updated after populationSize offspring generations.*
   In this case, we execute real evaluations more aggressively to find out whether any solution is the real new elitist. If the surrogate value of the solution is close to the elitist surrogate value, then a real evaluation is performed. The fitness distance of a solution with surrogate value v to the local elitist surrogate with value $v_{elitist}$ is determined by $\delta = \frac{v_{elitist} - lowerBound}{v - lowerBound}$, where *lowerBound* is the minimum value found in the initial random set of solutions (after a warm-up period). The correction by *lowerBound* is required to make δ independent of the range of fitness values. The real evaluation is performed if $\delta < \delta_{threshold}$. During the preliminary experiments, we have found that a good value for $\delta_{threshold}$ is 1.02.

3. *If the solution has not been changed both after the first stage of generating the offspring and the FI.*
   In this case, a real evaluation is performed, as it is expected that this solution, the surrogate value of which has not been improved during GOM and FI, has a high real fitness value as well.

Algorithm 4.3.3: Offspring generation.

```
Function generateOffspring(P):
    if mixedPopulationsMode then
        realPopulationSize ← ⌈populationSize · (0.9 − modelQuality)⌉
        if realPopulationSize > populationSize/2 then
            realPopulationSize ← populationSize/2
    for s ∈ P do
        if mixedPopulationsMode and sIndex < realPopulationSize then
            offspring ← generateOffspringWithoutSurrogate (s) else
                offspring ← generateOffspringUsingSurrogate (s)
```

Algorithm 4.3.4: Offspring generation without a surrogate model.

```
Function generateOffspringWithoutSurrogate(parent):
    offspring ← parent
    for subset ∈ FOS do
        donor ← random solution from P[0 ... realPopulationSize − 1]
        candidate ← generateSolution (offspring, subset, donor)
        // generate new solution with GOM
        doRealEvaluation (candidate)
        if change is accepted then offspring ← candidate
    if offspring not changed then
        performForcedImprovements ()
    return offspring
Function doRealEvaluation(s):
    r ← evaluate(s)
    if r > r_elitist then r_elitist ← r
    add(s,r) to evaluated solutions archive
```

Algorithm 4.3.5: Offspring generation using a surrogate model.

```
Function generateOffspringUsingSurrogate(parent):
    offspring ← parent
    realEvaluationPerformed ← false
    for subset ∈ FOS do
        donor ← random solution from the population
        candidate ← generateSolution (offspring, subset, donor)
        // generate new solution with GOM
        v ← predictFitnessValue (candidate)
        if realEvaluationPerformed = false then
            if v > v_elitist then
                v_elitist ← v // updating local surrogate elitist
                doRealEvaluation (candidate)
                realEvaluationPerformed ← true
            else
                if elitist not improved for populationSize offspring then
                    δ = (v_elitist − lowerBound) / (v − lowerBound)
                    if δ < δ_threshold then
                        doRealEvaluation (candidate)
                        realEvaluationPerformed ← true
        if change is accepted then
            offspring ← candidate
    if offspring not changed then
        performForcedImprovements (offspring)
    if offspring not changed then
        doRealEvaluation (offspring)
    return offspring
```

EVALUATED SOLUTIONS ARCHIVE

As the real evaluations of solutions are presumed to be expensive, we maintain an archive of already evaluated solutions to avoid repeated evaluations. Each population maintains its own archive of solutions acquired by the steps of the optimization algorithm ($optArchive_i$). The solutions acquired by the random generation are placed in the special archive *randomArchive*, which is shared among all populations.

Both types of archive are re-used during the re-training of the surrogate model. Using random solutions is important as they are distributed throughout the search space while the solutions obtained during the optimization process tend to be located in a certain part of the space. Specifically, in the k-th population in the IMS for model training and validation, the solutions from $randomArchive \cup optArchive_0 \cup \cdots \cup optArchive_k$ are used.

4.4. EXPERIMENTS

4.4.1. OPTIMIZATION PROBLEMS

We consider a set of well-known benchmark functions of binary variables. All functions need to be maximized.

The first function is Onemax, in which the variables are independent:

$$f_{Onemax}(x) = \sum_{i=0}^{l-1} x_i$$

The other functions, on which we test the algorithms, have blocks of dependent variables. Two of such functions are the well-known deceptive traps [23] of order $k = 4$, with tight (dependent variables are located close to each other) and loose encoding (dependent variables are distant):

$$f_{TrapK-Tight}(x) = \sum_{i=0}^{l/k-1} f^{sub}_{TrapK}\left(\sum_{j=0}^{k-1} x_{ik+j}\right),$$

$$f_{TrapK-Loose}(x) = \sum_{i=0}^{l/k-1} f^{sub}_{TrapK}\left(\sum_{j=0}^{k-1} x_{i+(l/k)j}\right),$$

$$f^{sub}_{TrapK}(u) = \begin{cases} 1 & \text{if u=k,} \\ \frac{k-1-u}{k} & \text{otherwise.} \end{cases}$$

The other function with blocks of dependent variables is the Hierarchical If-And-Only-If (HIFF) function. It is considered to be more difficult as blocks of different sizes are presented (blocks hierarchy):

$$f_{Hiff}(x) = \sum_{k \in \{1,2,4 \ldots \frac{l}{2}, l\}} \sum_{i=0}^{l/k-1} f^{sub}_{Hiff}(x_{ik \ldots (i+1)k-1}),$$

$$f^{sub}_{Hiff}(u) = \begin{cases} 1 & \text{if } \sum_{j=0}^{k} u_j = k \text{ or } \sum_{j=0}^{k} u_j = 0, \\ 0 & \text{otherwise.} \end{cases}$$

The final function that we consider is the NK-landscapes with maximum overlap and blocks of $k = 5$ [24]. This function contains blocks of dependent variables, but with fitness values of subfunctions depending on the block position:

$$f_{NK-S1}(x) = \sum_{i=0}^{l-k} f^{sub}_{NK-S1}(x_{(i,i+1,\ldots,i+k-1)}),$$

where the values of $f^{sub}_{NK-S1}(x_{(i,i+1,\ldots,i+k-1)})$ are tabular values, generated randomly in interval $[0;1]$.

4.4.2. EXPERIMENTAL SETUP

The proposed CS-GOMEA is compared with GOMEA, SMAC and Hyperopt. In order to make the comparison fair, in all algorithms we count evaluations of unique solutions only. We consider the problems described in Section 4.4.1 with the following problem sizes: for Trap4-Tight, Trap4-Loose, and HIFF problems $l \in \{8, 16, 32, 64, 128, 256\}$, for Onemax $l \in \{25, 50, 100, 200, 400\}$, for NK-landscapes $l \in \{25, 50\}$. We run SMAC and Hyperopt for Trap4 and HIFF problems for $l \in \{8, 16\}$, for Onemax $l = 25$, for NK-landscapes $l = 25$. We do not find it reasonable to run SMAC and Hyperopt for larger problem sizes as they fail to find an optimum in some runs even for the considered problem sizes. For each problem instance we perform 30 runs (for CS-GOMEA each run is a full optimization procedure including the neural network hyperparameter search).

The stopping criterion for GOMEA and CS-GOMEA was either finding an optimal solution or exceeding a time limit. The time limit for most problems was set to 4 hours. For NK-S1 the time limit was set to 6 hours. For Trap4 and HIFF problem instances with $l = 256$, the time limit was set to 8 hours. The time limit also determines the maximum allowed time for the CNN hyperparameter search which is not allowed to consume more than half of the overall computational budget. For SMAC and Hyperopt the stopping criteria was executing 10000 function evaluations, except for NK-S1, for which it was 20000 evaluations.

We study the scalability of the algorithms, i.e., the relation between the number of required evaluations to find the optimum and the problem size. The results are presented in Figure 4.4.1. Only the runs with the optimum achieved are reported. Tables with experimental results can be found in Table 4.4.2.

Besides scalability, we analyze convergence speed, i.e. whether the elitist solution is being gradually improved or the algorithm rapidly finds a solution close to the optimum (what is expected when an accurate surrogate model is used). Convergence behavior analysis might be important for practical usage in real-world expensive optimization tasks, as it might be sufficient to quickly find a good quality solution. Convergence plots are presented in Figure 4.4.2.

WARM-UP SIZE HYPERPARAMETER

We adjust the value of the *warmupSize* hyperparameter with respect to the problem size (Table 4.4.1), starting with *warmupSize*= 100 for moderate dimensionalities. For larger dimensionalities doubling the problem dimensionality is accompanied by approximately doubling the training dataset size. For practical usage, it means that we can start with a

problem	ℓ					
	8	**16**	**32**	**64**	**128**	**256**
Trap4-Tight	100	100	100	150	200	300
Trap4-Loose	100	100	100	150	200	300
HIFF	100	100	100	150	200	300
	25	**50**	**100**	**200**	**400**	
Onemax	100	100	100	150	200	
NK-S1	200	200				

Table 4.4.1. Values of *warmupSize* hyperparameter.

4

moderate *warmupSize* (e.g., 100) and gradually increase it until the fitness of the found elitist solution stops improving.

IMPLEMENTATION AND EXPERIMENTAL ENVIRONMENT DETAILS

The main algorithm code is written in C with Python function calls. Neural network operations are implemented in Python using the Pytorch framework. The experiments are run on an Intel Xeon E5-2630 CPU core and an Nvidia Titan X (Pascal architecture) GPU. Code is available at https://github.com/ArkadiyD/CS-GOMEA.

4.4.3. RESULTS

For Onemax, Trap4-Tight, Trap-Loose, HIFF and NK-S1 problems CS-GOMEA was able to find an optimal solution in all executed runs for all considered problem sizes. For the first four problems, the number of evaluations required by CS-GOMEA scales substantially better than for GOMEA. According to our calculations, the median minimal function evaluation time, that makes CS-GOMEA usage reasonable (i.e., decreasing overall run time compared to GOMEA) is ≈ 5 seconds for all problems with $l \leq 16$ and ≈ 1 second for problems with $l > 16$. Such evaluation time is well below the function evaluation time in various real-world optimization problems, especially in the deep learning field.

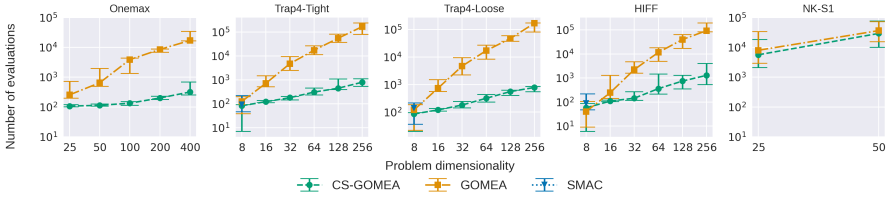


Figure 4.4.1. Scalability of considered algorithms in terms of the required number of evaluations to reach the optimum, summarized over thirty runs. The algorithms are presented only for the problem instances in which an optimum was achieved in all 30 runs. There are no such problem instances for Hyperopt. The bars show the intervals of the required number of evaluations, excluding the lowest and highest values (what corresponds to $\approx 93\%$ interval). The points indicate the median values.

The comparison with SMAC and Hyperopt shows that they are competitive with CS-GOMEA only for low-dimensional problems. For dimensionality 8, SMAC found the optimum in all runs for Trap4 and HIFF problems. For dimensionality 16, the optimum was not found in all runs. Hyperopt failed to find the optimum in some cases, even for dimensionality 8. For dimensionality 25 for Onemax and NK-S1, both of the algorithms have failed to find the optimal solution in some runs. Moreover, as shown in Figure 4.4.2, their convergence speed is lower than for CS-GOMEA.

problem	algorithm	ℓ					
		8	**16**	**32**	**64**	**128**	**256**
Trap4-Tight	**CS-GOMEA**	83	122	180	308	446	795
	GOMEA	126	716	4762	17219	54908	167720
Trap4-Loose	**CS-GOMEA**	86	121	181	318	556	784
	GOMEA	120	741	4673	17077	46760	167916
HIFF	**CS-GOMEA**	58	109	144	360	743	1283
	GOMEA	40	245	2235	11587	39757	92195
		25	**50**	**100**	**200**	**400**	
Onemax	**CS-GOMEA**	107	114	134	201	318	
	GOMEA	256	649	3871	8505	16967	
NK-S1	**CS-GOMEA**	5619	29202	-	-	-	
	GOMEA	7966	36308	-	-	-	

Table 4.4.2. Number of median required evaluations to achieve an optimum by CS-GOMEA and GOMEA for all problem instances. 30 runs are executed for each problem instance.

The NK-landscapes function turned out to be the most difficult for the surrogate model to approximate. The achieved model quality seems to be insufficient to solve the problem using predominantly the surrogate model. To be able to achieve the optimum, the mixed populations mode is required. In this mode, the number of evaluations grows significantly, but still, CS-GOMEA finds the optimum with fewer evaluations than GOMEA.

4.4.4. STATISTICAL TESTS

To verify the significance of the results, we perform statistical tests. As the normality of the distribution of the required number of evaluations to achieve an optimum cannot be assumed, we perform the Mann-Whitney U test [25]. For each problem and each dimensionality, we perform a statistical test, testing that the number of evaluations to achieve the optimum by GOMEA is larger than by CS-GOMEA. The considered level of statistical significance is $\alpha = 0.05$. Bonferroni correction is applied. According to the conducted experiments, CS-GOMEA requires statistically significantly fewer evaluations than GOMEA for Onemax, Trap4-Tight, Trap4-Loose for all considered values of l. For HIFF with $l \geq 16$.

4.5. DISCUSSION

The proposed type of surrogate model demonstrated high approximation accuracy for several functions with blocks of dependent variables, including the HIFF problem. NK-

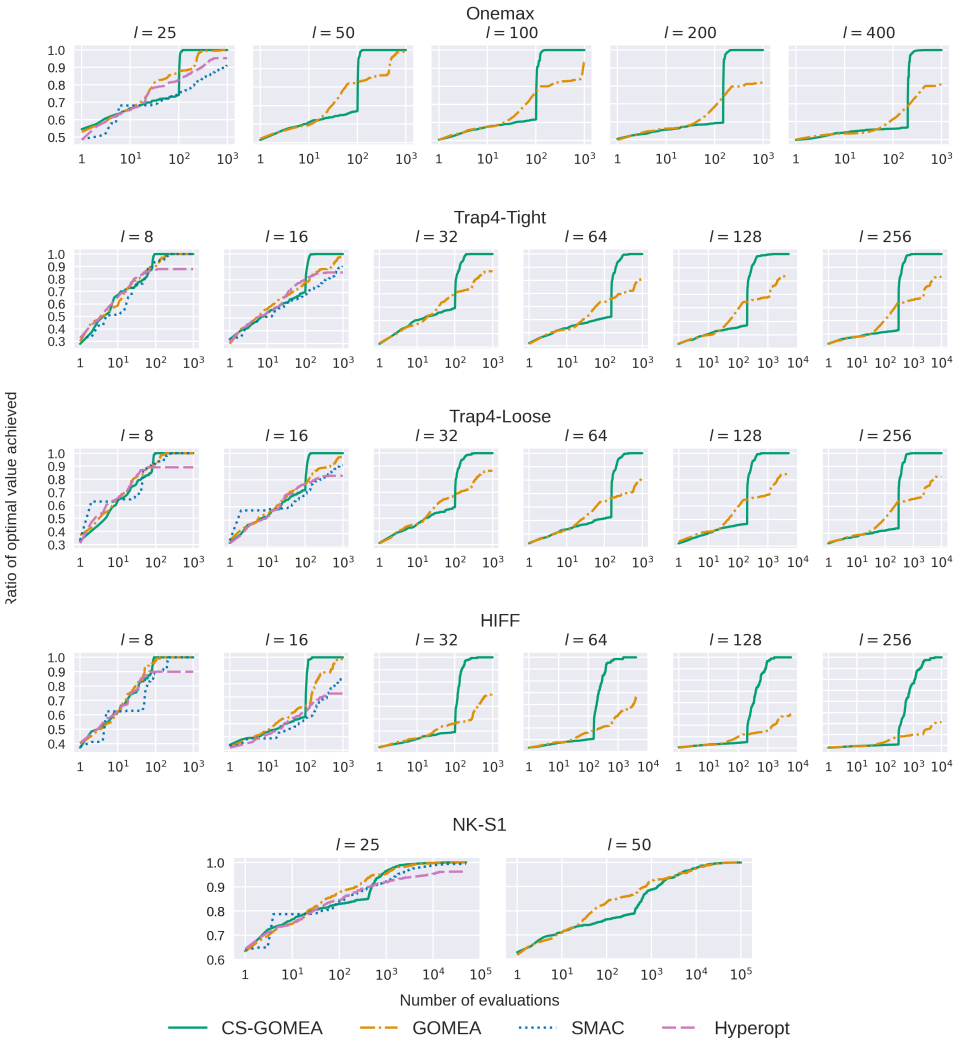


Figure 4.4.2. Convergence of all considered algorithms for all considered problems in terms of ratio to the optimal value. For each problem instance and for every number of evaluations, the fitness value of the elitist solution is averaged over 30 runs. The ranges for the horizontal axis are chosen dependent on problem size and CS-GOMEA performance.

landscapes are however difficult to be approximated by the proposed CNN due to the CNN main principle of assuming the same subfunctions occurring in different solution locations (the convolutional filters weights are re-used in the convolutions) while in NK-landscapes such subfunctions are random and thus different everywhere. In the current chapter, we do not in any way exploit the linkage information captured in the LT in surrogate modeling. Rather, we focused on providing a baseline proof-of-principle

that a CNN-based surrogate-assisted GOMEA has merit. However, we believe linkage integration is an important research question for future work, i.e., taking into account the subsets of dependent variables in neural network architecture design instead of moving the filters along the input for all variables. This may well be highly beneficial for model quality on such difficult non-regular problems as NK-S1.

4.6. CONCLUSIONS

We have introduced a novel surrogate-assisted GA for expensive discrete (boolean) optimization problems. The key novel features of our algorithm are keeping the strengths of the GOMEA algorithm while using a convolutional neural network (CNN) as a surrogate model with a pairwise regression approach for model training to be able to train the CNN with small numbers of samples.

Experiments on a well-known set of benchmark functions show that the proposed integration requires much less function evaluations than GOMEA to reach the optimum and scales much better than GOMEA. The small number of solutions required to find the optimum is promising for the ultimate goal of using the algorithm for expensive (real-world) optimization problems, which is part of our near-future research.

The comparison with the existing Bayesian optimization-based, expensive discrete algorithms SMAC and Hyperopt showed that using the proposed surrogate-assisted GA is more promising for the considered type of optimization problems because, firstly, CS-GOMEA is more scalable and can solve problems with much larger problem sizes, and, secondly, even for small problem sizes CS-GOMEA requires less numbers of evaluations to achieve the optimum.

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO). We gratefully acknowledge the support of NVIDIA Corporation with providing Titan X GPU cards used for this research.

REFERENCES

[1] "Model-based methods for continuous and discrete global optimization". In: *Applied Soft Computing* 55 (2017), pp. 154–167.

[2] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. "Taking the human out of the loop: a review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[3] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 507–523.

[4] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. "Hyperopt: a Python library for model selection and hyperparameter optimization". In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[5] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[6] F. Hutter, H. Hoos, and K. Leyton-Brown. "An evaluation of sequential model-based optimization for expensive blackbox functions". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2013, pp. 1209–1216.

[7] K. Eggensperger et al. "Towards an empirical foundation for assessing Bayesian optimization of hyperparameters". In: *NIPS workshop on Bayesian Optimization in Theory and Practice*. Vol. 10. 2013, p. 3.

[8] A. Moraglio and A. Kattan. "Geometric generalisation of surrogate model based optimisation to combinatorial spaces". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2011, pp. 142–154.

[9] M. Zaefferer et al. "Efficient global optimization for combinatorial problems". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2014, pp. 871–878.

[10] S. Verel, B. Derbel, A. Liefooghe, H. Aguirre, and K. Tanaka. "A surrogate model based on Walsh decomposition for pseudo-Boolean functions". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 181–193.

[11] A. Diaz-Manriquez, G. Toscano Pulido, J. Barron-Zambrano, and E. Tello. "A review of surrogate assisted multiobjective evolutionary algorithms". In: *Computational Intelligence and Neuroscience* 2016 (June 2016), pp. 1–14.

[12] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. "Object recognition with gradient-based learning". In: *Shape, Contour and Grouping in Computer Vision*. Springer, 1999, pp. 319–345.

[13] A. Massaro and E. Benini. "A surrogate-assisted evolutionary algorithm based on the genetic diversity objective". In: *Applied Soft Computing* 36 (Aug. 2015), pp. 87–100.

[14] P. A. N. Bosman and D. Thierens. "Linkage neighbors, optimal mixing and forced improvements in genetic algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 585–592.

[15] T. Hengl, M. Nussbaum, M. N. Wright, G. B. Heuvelink, and B. Gräler. "Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables". In: *PeerJ* 6 (2018), e5518.

[16] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *Proceedings of the 3th International Conference on Learning Representations*. Ed. by Y. Bengio and Y. LeCun. 2015.

[17] F. Yu and V. Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

[18] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. "An empirical evaluation of deep architectures on problems with many factors of variation". In: *Proceedings of the International Conference on Machine Learning*. ACM. 2007, pp. 473–480.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[20] N. H. Luong, H. La Poutré, and P. A. N. Bosman. "Multi-objective gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start Scheme". In: *Swarm and Evolutionary Computation* 40 (2018), pp. 238–254.

[21] D. Thierens and P. A. N. Bosman. "Optimal mixing evolutionary algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2011, pp. 617–624.

[22] K. S. Bhattacharjee, H. K. Singh, T. Ray, and J. Branke. "Multiple surrogate assisted multiobjective optimization using improved pre-selection". In: *2016 IEEE congress on evolutionary computation (CEC)*. IEEE. 2016, pp. 4328–4335.

[23] K. Deb and D. E. Goldberg. "Sufficient conditions for deceptive and easy binary functions". In: *Annals of Mathematics and Artificial Intelligence* 10.4 (1994), pp. 385–408.

[24] M. Pelikan, K. Sastry, D. E. Goldberg, M. V. Butz, and M. Hauschild. "Performance of evolutionary algorithms on NK-landscapes with nearest neighbor interactions and tunable overlap". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2009, pp. 851–858.

[25] H. B. Mann and D. R. Whitney. "On a test of whether one of two random variables is stochastically larger than the other". In: *The Annals of Mathematical Statistics* (1947), pp. 50–60.

5

A NOVEL SURROGATE-ASSISTED EVOLUTIONARY ALGORITHM APPLIED TO PARTITION-BASED ENSEMBLE LEARNING

We propose a novel surrogate-assisted Evolutionary Algorithm for solving expensive combinatorial optimization problems. We integrate a surrogate model, which is used for fitness value estimation, into a state-of-the-art P3-like variant of the Gene-Pool Optimal Mixing Algorithm (GOMEA) and adapt the resulting algorithm for solving non-binary combinatorial problems. We test the proposed algorithm on an ensemble learning problem. Ensembling several models is a common machine learning technique to achieve better performance. We consider ensembles of several models trained on disjoint subsets of a dataset. Finding the best dataset partitioning is naturally a combinatorial non-binary optimization problem. Fitness function evaluations can be extremely expensive if complex models, such as Deep Neural Networks, are used as learners in an ensemble. Therefore, the number of fitness function evaluations is typically limited, necessitating expensive optimization techniques. In our experiments, we use five classification datasets from the OpenML-CC18 benchmark and Support-vector Machines as learners in an ensemble. The proposed algorithm demonstrates better performance than alternative approaches, including Bayesian optimization algorithms. It manages to find better solutions using just several thousand fitness function evaluations for an ensemble learning problem with up to 500 variables.

The contents of this chapter are based on the following publication: **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "A novel surrogate-assisted evolutionary algorithm applied to partition-based ensemble learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2021, pp. 583–591.

5.1. INTRODUCTION

Expensive combinatorial optimization is a combinatorial optimization subfield, in which fitness function evaluations are (computationally) expensive. In contrast to the case of optimization problems with inexpensive function evaluations, in case of problems with expensive function evaluations, search algorithms are usually evaluated by their ability to find good solutions within a limited number of function evaluations rather than by a required number of evaluations to find the global optimum. Expensive optimization problems arise in various domains. One traditional application is simulation-based optimization in engineering design [1, 2, 3]. Combinatorial expensive optimization has recently achieved increased attention due to the popularity of the Neural Architecture Search (NAS) field [4, 5]. The goal of NAS is to find the best Neural Network architecture for a particular task. A function evaluation that corresponds to a partial [6] or end-to-end [7] Deep Neural Network (DNN) training procedure can be extremely expensive, taking up to several hours [8]. A related application is finding an ensemble of deep learning models through data partitioning, for instance, to tackle the problem of data heterogeneity in medical image analysis ([9], Chapter 7). Here too, the efficiency of the search algorithm is extremely important, as each function evaluation again requires training a DNN. Therefore, novel efficient algorithms for combinatorial expensive optimization are highly demanded.

Bayesian Optimization (BO) is a traditional approach to solving expensive optimization problems. The most common BO algorithms are based on Gaussian processes, aimed at real-valued variables. For their use in the combinatorial domain, they were shown to have caveats [10]. Consequently, novel BO algorithms were developed, designed to handle categorical variables. Replacing the Gaussian Process surrogate model with Random Forests [11] was proposed in Sequential Model-based Algorithm Configuration (SMAC) [12]. Another popular BO algorithm, which was shown to perform better than traditional Gaussian process-based BO in categorical and mixed domains, is the Tree-structured Parzen Estimator (TPE) [13] and its implementation in the Hyperopt package [14]. Recently, two BO algorithms with better performance than SMAC and Hyperopt were proposed: Bayesian Optimization of Combinatorial Structures (BOCS) [15] and Combinatorial Bayesian Optimization using the Graph Cartesian Product (COMBO) [16]. The main idea of both these algorithms is to explicitly model interactions between variables. While BOCS takes into account only second-order interactions, COMBO has no limit on the order of interactions and builds a sparse Bayesian regression model over possible high-order interactions between variables. BOCS is limited to binary variables only, while COMBO can handle variables with higher cardinality. COMBO showed state-of-the-art performance on both common binary optimization benchmark problems such as Ising Spin-glass and MAXSAT and an example of NAS. However, this comes at a cost of extremely expensive computations that grows fast as the number of function evaluations increases for maintaining the surrogate model. For instance, only experiments with up to 300 function evaluations were feasible in [16].

Surrogate-assisted Evolutionary Algorithms (EAs) are an interesting alternative approach to solving expensive optimization problems. Using surrogate models to replace part of the function evaluations done by an EA is a natural way to reduce the number of evaluations while keeping the powerful search characteristics of EAs. Common surrogate

models are polynomials, Kriging, Radial Basis Function Network (RBFN), or Support Vector Regression (SVR) [17]. However, surrogate-assisted EAs are not as widely used for combinatorial optimization problems as for real-valued ones. A recent approach for combinatorial optimization integrates a Convolutional Neural Network (CNN) with an advanced model-based EA called Convolutional Neural Network Surrogate-assisted GOMEA (CS-GOMEA), but it is limited to binary problems (Chapter 4).

Local Search (LS) is a well-known simple, yet often effective search algorithm. The main principle is to perform greedy search (only improvements are accepted) in the vicinity of a current solution, leading to quick convergence to a local optimum. It was recently shown that LS can be very efficient for combinatorial formulations of NAS, achieving competitive performance with state-of-the-art EAs and outperforming a commonly adopted baseline in NAS - Random Search (RS) [5, 18].

Ensembling is a common machine learning technique to improve performance. Usually, individual models are simple learners such as decision trees. A classic approach is bagging [19], where several models are trained on randomly selected subsets of samples. An alternative approach is to train models on disjoint subsets of samples [20, 21]. In contrast to bagging, with such an approach it is possible to train each model on homogeneous data, increasing chances of better performance. Finding the best partitioning then, however, is a potentially expensive optimization problem.

In this chapter, we propose a novel surrogate-assisted EA based on a state-of-the-art P3-like variant of GOMEA. Unlike CS-GOMEA (Chapter 4), it is not limited to binary problems, does not make assumptions about problem regularity (subfunctions of fixed size), and uses a more efficient population management scheme - Parameter-less Population Pyramid (P3). To the best of our knowledge, this is the first time a surrogate model is integrated into a P3-like EA. We also propose a simple, yet shown to be an efficient adaptive mechanism to control and balance the number of performed real and surrogate fitness evaluations. The introduced algorithm is applied to an ensembling problem defined on supervised classification datasets. Support-Vector Machines (SVMs) are used as learners in the considered ensembles.

5.2. SEARCH PROBLEMS AND ALGORITHMS

5.2.1. PROBLEM FORMULATION

In general, we consider unconstrained combinatorial optimization problems with categorical variables. Potentially, the cardinality of problem variables differs. However, for simplicity of notation in this chapter, we assume that all variables have cardinality α, meaning that all variables have α possible values. Mathematically, the considered problems can be formulated as a global optimization problem $\max_{x \in \mathscr{D}} f(x)$, where $f(x) : \mathscr{D} \rightarrow \mathbb{R}$, $\mathscr{D} = \{0, \ldots, \alpha - 1\}^{\ell}$, ℓ is the number of variables, and α is the alphabet size.

5.2.2. LOCAL SEARCH

In contrast to RS, LS traverses the search space in an ordered, local manner, quickly converging to a local optimum. In a random restart scenario, LS starts from a randomly generated solution. We specifically consider first improvement neighborhood search as our LS. A single iteration of LS consists of considering all variables in a random order and

assigning to each variable the value in its domain that corresponds to the best fitness function value. Iterations are repeated until a solution is not changed. After that, a new random solution is generated and the search loop is repeated. The Random Restart Local Search pseudocode is provided in Algorithm 5.2.1.

5.2.3. AN ADAPTATION OF P3 FOR NON-BINARY PROBLEMS

We adapt the state-of-the-art model-based Evolutionary Algorithm known as the Parameter-less Population Pyramid (P3) [22] for solving non-binary combinatorial optimization problems. P3 is a variant of the Gene-pool Optimal Mixing Algorithm (GOMEA) [23] that uses local search, a complementary donor search, and a fitness-pyramid structured growing population. We call the P3 variant used in this chapter *P3-GOMEA*. Below, we give a brief overview of its features. Pseudocode is provided in Algorithm 5.2.2 and Algorithm 5.2.3.

LINKAGE MODEL

The idea of using a so-called Linkage Model for guiding the evolution process through variation restricted to specific variables and immediately testing for improvements was introduced with the concept of Optimal Mixing Evolutionary Algorithms (OMEAs) [24]. A Linkage Model is commonly defined in the form of a set of (possibly overlapping) subsets of variables. Such a structure is called a Family Of Subsets (FOS). The goal is to use information about dependencies between variables to perform crossover more efficiently. In the case of Black-Box optimization these dependencies are not known a priori, and, therefore, they have to be learned during optimization. A much-used approach is to first estimate pairwise dependencies from the population and build higher-order models upon this. For pairwise dependencies learning, either Mutual Information (MI) [24] or Normalized Mutual Information (NMI) [22] is often used. Both MI and NMI calculations can be naturally extended to the case of non-binary variables [25]. P3-GOMEA uses the NMI measure. After pairwise dependencies are learned, they can be used for Linkage Model construction.

The FOS model known as the Linkage Tree (LT), which uses a hierarchical structure to store sets of dependent variables, was shown to often be the most efficient one [24]. An LT is a binary tree with $2\ell - 1$ nodes. LT leaves are singletons with variables. The root of an LT is a set containing all variables. All other nodes are linkage subsets, F^i which are unions of disjoint subsets of children k, j of node i: $F^i = F^j \cup F^k, F^j \cap F^k = \emptyset$.

P3-GOMEA uses a filtered LT Model as proposed in [26], in which redundant subsets are removed. When two subsets F^j and F^k are merged into a subset F^i, it may happen that the MI or NMI measure between them is maximal (one). It is supposed that there is then no merit in using these subsets in variation separately, as it may disrupt a building block F^i. Thus, keeping subsets F^j and F^k in a FOS is not reasonable, and only the parent subset F^i is kept.

GENE-POOL OPTIMAL MIXING

The Gene-pool Optimal Mixing operator (GOM) is the variation operator used in the GOMEA family of algorithms. GOM is applied to a single solution and outputs a single solution that is never worse than the input solution. To improve a solution, GOM loops

over the contents of the FOS model. For each linkage subset F^i, GOM attempts to overwrite the values of the variables in F^i of the solution in consideration, with values from a donor solution, often chosen at random from the population. If this overwriting action does not cause the fitness of the solution to become worse, the copy action is accepted. Otherwise, the donor material is rejected and the action is undone. The pseudocode of GOM is provided in Algorithm 5.2.3.

PARAMETERLESS POPULATION MANAGEMENT SCHEME

Choosing an optimal population size is not trivial, therefore, a population management scheme without a population size parameter has much practical value. Several such schemes were proposed for the GOMEA family of algorithms [27]. In this chapter, we consider P3 [22], which was shown to be efficient [27], is fully parameterless, and can be naturally adapted to a surrogate-assisted EA.

Solutions are stored in a pyramidal structure. Each layer of the pyramid is a set of solutions (duplicates are not stored). Linkage Models are learned separately for each pyramid level. In each iteration, only one solution is evolved. A new solution is generated randomly and added to the bottom layer of the pyramid. Then, by using solutions from the current pyramid level as donors, the current solution is evolved using GOM. If GOM leads to fitness improvement, the resulting solution is added to the next pyramid level, and GOM is performed at the next pyramid level. Otherwise, processing this solution is finished, and a new iteration starts with a new, randomly generated solution.

Algorithm 5.2.1: Random Restart Local Search Algorithm.

```
Function LocalSearch():
    do
        improved = false
        s ← generateRandomSolution()
        do
            s.fitness ← evaluate(s)
            for i ∈ randomPermutation({0,...,ℓ−1}) do
                s' ← s
                for j ∈ {0,...,α−1} \ s_i do
                    s'_i ← j
                    s'.fitness ← evaluate(s')
                    if s'.fitness > s.fitness then
                        s_i ← s'_i
                        improved ← true
        while improved
    while budget is not exhausted
```

HILL CLIMBER

Before evolving a solution using GOM, an LS algorithm (also called a Hill Climber in [22]) can be applied to quickly bring the solution to a local optimum. Such an approach is used in the original P3 algorithm [22]. However, our preliminary experiments showed that in the case of a limited number of allowable function evaluations (several thousand evaluations) such use of LS leads to similar performance as stand-alone LS (without P3)

because the majority of function evaluations are spent on the LS phase. Therefore, we do not include LS in our P3-GOMEA.

5.2.4. SURROGATE-ASSISTED EA

We integrate, in a novel way, a surrogate model into the above-described P3-GOMEA to replace a part of the real function evaluations with surrogate function evaluations. Our proposed approach to integrating surrogate models can be applied to various search algorithms. In this chapter, we focus on a surrogate-assisted version of the P3-GOMEA algorithm that we refer to as *SA-P3-GOMEA*. The changes required to P3-GOMEA to obtain SA-P3-GOMEA are provided in Algorithm 5.2.2 and Algorithm 5.2.3.

SURROGATE MODEL INTEGRATION

The outline of the proposed surrogate model integration into a search algorithm is as follows. First, some initial random solutions are evaluated. In general, the number of initial solutions can be tuned, but we use a simple approach that is also used in COMBO, namely, ℓ random solutions are generated and evaluated. Then, a surrogate model is trained on these solutions. Next, the search is performed the same as without a surrogate model, but whenever solutions need to be evaluated, mainly surrogate (estimated by the surrogate model) evaluations are performed. Real function evaluations are performed only in case of high predicted fitness value. The rationale is that solutions with high surrogate fitness values should also have high real fitness values. However, for the sake of biasing the search competently toward high-quality solutions, we must be sure of just exactly how good this solution is.

The condition of performing a real function evaluation is defined as follows. Suppose, R is an array of solutions with known real fitness values and F is an array of corresponding surrogate fitness values. A real function evaluation is performed for a solution s, if its surrogate fitness f is greater than a threshold $\mathcal{T}$. This threshold is calculated as λ multiplied by the maximum value stored in the array F. The value of λ is dynamically adjusted. In the beginning, $\lambda = 1$, meaning that a real function evaluation for a solution is performed only if its surrogate fitness exceeds the current surrogate elitist value. If an algorithm has not found new real elitists for a while, then we suppose that the current surrogate model is not accurate enough and it is reasonable to perform real evaluations more elaborately. Therefore, the value of λ is then multiplied by a hyperparameter η ($0 < \eta < 1$), effectively relaxing the condition for a real fitness evaluation. Once an algorithm finds a new elitist, λ is reset to 1. After evolving one solution in SA-P3-GOMEA, the surrogate model is retrained, and values of F are re-calculated. The frequency of the model retraining can be potentially adjusted, but we stick to this natural retraining schedule. Note that the proposed surrogate-assisted EA has only one numeric hyperparameter η.

As a surrogate model is unlikely to predict surrogate fitness values with perfect precision, comparing real fitness values and surrogate fitness values is undesirable and leads to inferior performance [28]. Therefore, for the comparison of the fitness values of the two solutions, the following strategy is used. If both solutions already have a real fitness value, they are compared via these. Otherwise, surrogate fitness values are calculated for both solutions and used for the comparison. The pseudocode of the comparison function is provided in Algorithm 5.2.3.

Algorithm 5.2.2: P3-GOMEA and Surrogate-assisted P3-GOMEA (SA-P3-GOMEA). The lines which are added in the surrogate-assisted algorithm are shown in green font color.

Parameters: A relaxation parameter η
Function `EA()`:
- $iter \leftarrow 0$
- $Pyramid \leftarrow [\emptyset]$
- $R \leftarrow \emptyset$
- **for** $i = 0, \ldots, \ell - 1$ **do**
  - $R \leftarrow R \cup$ `generateRandomSolution()`
- `trainSurrogateModel(`R`)`
- $F \leftarrow$ `predictSurrogateFitness(`R`)`
- $\lambda \leftarrow 1$
- $\mathcal{T} \leftarrow$ `setThreshold(`F, λ`)`
- **while** $\neg$ *terminationCriterionSatisfied* **do**
  - $elitist \leftarrow NULL$ `// elitist value is updated if necessary when real evaluations are performed`
  - $p \leftarrow$ `generateRandomSolution()`
  - $Pyramid^0 \leftarrow Pyramid^0 \cup \{p\}$
  - $solutionsAdded \leftarrow$ `true`
  - $currentTopLevel \leftarrow |Pyramid| - 1$
  - $\mathcal{L} \leftarrow 0$
  - $elitistBefore \leftarrow elitist$
  - **while** $\mathcal{L} \leq currentTopLevel$ ***and*** $solutionsAdded$ **do**
    - $\mathcal{F} \leftarrow$ `learnLinkageModel(`$Pyramid^{\mathcal{L}}$`)`
    - $o \leftarrow$ `GOM(`$p, Pyramid^{\mathcal{L}}, \mathcal{F}$`)`
    - **if** `compareSolutions(`o, p`)` **then**
      - **if** $\mathcal{L} = currentTopLevel$ **then**
        - $Pyramid^{\mathcal{L}+1}$.append($\emptyset$)
      - $Pyramid^{\mathcal{L}+1} \leftarrow Pyramid^{\mathcal{L}+1} \cup \{o\}$
      - $solutionsAdded \leftarrow$ `true`
    - $p \leftarrow o$
    - $\mathcal{L} \leftarrow \mathcal{L} + 1$
  - $elitistAfter \leftarrow elitist$
  - **if** $\neg$`compareSolutions(`$elitistAfter, elitistBefore$`)` **then**
    - $\lambda \leftarrow \lambda\eta$
  - `trainSurrogateModel(`R`)`
  - $F \leftarrow$ `predictSurrogateFitness(`R`)`
  - $\mathcal{T} \leftarrow$ `setThreshold(`F, λ`)`

Algorithm 5.2.3: Key functions used in the P3-GOMEA and SA-P3-GOMEA algorithms. The lines of code which are not used in the surrogate-assisted algorithm are shown in orange font color. The lines which are added in the surrogate-assisted algorithm are shown in green font color.

```
Variables: Real elitist solution elitist, threshold 𝒯, quantile λ, set of solutions with known real fitness values R
Function GOM(o, 𝒫, ℱ):
    backup ← o
    changed ← false
    ℱ ← sortFOSInAscendingOrder(ℱ)
    for i ∈ {0,1,...,|ℱ|−1} do
        donorsList = randomPermutation({𝒫_0, 𝒫_1, ..., 𝒫_{n−1}})
        for j ∈ {0,1,...,n−1} do
            d ← donorsList[j]
            o_Fi ← d_Fi
            if o_Fi ≠ d_Fi then
                evaluateSolution(o)
                if acceptChange (o) then
                    backup_Fi ← o_Fi
                    changed ← true
                else
                    o_Fi ← backup_Fi
                    break
    return o
Function evaluateSolution(x):
    x.fitness ← calculateFitness(x)
    x.realFitnessCalculated ← true
    x.surrogateFitness ← predictSurrogateFitness(x)
    if x.surrogateFitness > 𝒯 then
        x.fitness ← calculateFitness(x)
        x.realFitnessCalculated ← true
        R ← R ∪ {x}
    if x.realFitnessCalculated then
        if x.fitness > elitist.fitness then
            elitist ← x
            λ ← 1
Function compareSolutions(x,y):
    if x.realFitnessCalculated and y.realFitnessCalculated then
        if x.realFitness > y.realFitness then return 1
        if x.realFitness = y.realFitness then return 0
        if x.realFitness < y.realFitness then return -1
    else
        if x.surrogateFitness > y.surrogateFitness then return 1
        if x.surrogateFitness = y.surrogateFitness then return 0
        if x.surrogateFitness < y.surrogateFitness then return -1
```

SURROGATE MODEL

Choosing the best surrogate model is often the key to good performance in surrogate-assisted optimization. However, the main goal of this chapter is not to propose the best-performing surrogate model, but rather to introduce a new adaptive mechanism for surrogate model integration, use it in P3-GOMEA, and test its performance on an ensembling problem. To underline the generality of our proposed mechanism, we experiment with four types of surrogate model.

1. Multi-Layer Perceptron (MLP). In contrast to a CNN used in Chapter 4, the considered Neural Networks are fully connected as we do not want to assume any regularity in input data such as subfunctions of fixed size. We use a fixed model structure with two hidden layers, each having a number of neurons equal to the input size.
2. Decision Tree-based Gradient Boosting [29]. This type of model was shown to perform well on a variety of tasks [30, 31, 32] and can naturally handle categorical variables.
3. Support Vector Regression (SVR). This is a common choice for surrogate-assisted EAs [17].
4. Random Forest (RF).

To allow the correct processing of categorical variables by MLP, RF, and SVR, input solutions are one-hot encoded. Thus, if an optimization domain has ℓ variables with alphabet size α, surrogate models get samples of dimensionality $\ell\alpha$ as input.

Gradient Boosting, RF, and SVR have several hyperparameters such as kernel type for SVR, and learning rate for Gradient Boosting. Hyperparameters are tuned only once (before the first model training) using 3-fold cross-validation and simple grid search. We consider tuning the MLP architecture to be out of the scope of this chapter. The list of tunable hyperparameters and considered values are provided in Table 5.5.3.

5.3. PARTITION-BASED ENSEMBLE LEARNING

We consider a well-known supervised classification machine learning problem. The task is to find a partitioning of the dataset into K disjoint subsets $C_0, \ldots C_{K-1}$ ($C_0 \cap C_1 \cap, \ldots, \cap C_{K-1} = \emptyset$) such that the aggregated performance of models $M_0, \ldots, M_{K-1}$ trained on these subsets is maximized.

5.3.1. EVALUATION OF FITNESS

The aggregated performance is the fitness measure in our EAs. To compute it, each model predicts class probabilities on the validation dataset (validation and train datasets do not overlap). Then, for each sample, these probabilities are averaged and the class with the maximum averaged probability is chosen as the final prediction. After obtaining predicted classes for all samples, the final fitness value is calculated as the standard accuracy metric. We use SVM as learners in constructed ensembles because of its good performance and computational efficiency (see Section 5.4.2).

We assume that fitness function evaluations are deterministic, i.e., the training of SVMs used in an ensemble is deterministic. Moreover, to count only truly different evaluations (without calculating solutions that define identical groups of samples twice

Dataset	Samples	Features	Classes
segment	2310	16	7
spambase	4601	57	2
wall-robot-navigation	5456	24	4
kc1	2109	21	2
optdigits	5500	40	11

Table 5.4.1. Datasets specification. All datasets are downloaded from the OpenML datasets collection.

i.e., 001122 encodes the same partitioning as 112200), all solutions along with their fitness values are stored in a normalized form. To do so, first, a solution is transformed into an actual partition: each subset k has N_k samples: $C_k = \{p_{i_0}, \ldots, p_{i_{N_k-1}}\}$. Then, subsets C are sorted by the minimal index p of a sample belonging to a subset. A normalized solution form is then obtained by assigning values to variables according to the sorted order of subsets. Such solution normalization guarantees that different solutions defining the same dataset partition are evaluated and counted only once. When a new solution s is considered to be evaluated, first, it is normalized to s' and looked up in the collection of evaluated solutions. If s' has not been evaluated, then evaluation is performed, and the obtained fitness value is stored for s'. Note, since we consider the optimization function to be a black box, the normalized representation is only used to calculate fitness. The solution itself is left unchanged.

5.4. EXPERIMENTAL SETUP

5.4.1. DATASETS

We consider five datasets from the OpenML-CC18 benchmark [33]. All datasets are supervised classification tasks and have at least 2000 samples. The selected datasets each have a different number of classes (between 2 and 11), and the number of features ranges from 16 to 57. All features are numeric. Specifications are provided in Table 5.4.1. From each dataset, 500 randomly selected samples are left out for the validation set, and the same number of randomly selected samples is left out for the test set. The remaining samples are shuffled and ℓ random samples are used for a considered optimization problem with ℓ variables (i.e., used for training).

5.4.2. ENSEMBLE TRAINING

We use standard regularized non-linear SVM with Radial Basis Function (RBF) kernel from the Scikit-Learn package. Before training, all features in the datasets are scaled to the unit interval. The regularization parameter of the SVM models is not tuned, but the default value of 1.0 is used. This choice is addressed in the Discussion.

5.4.3. CONSIDERED OPTIMIZATION ALGORITHMS

As simple baselines, we include in our experiments RS and LS. We also compare the performance of the proposed SA-P3-GOMEA to P3-GOMEA. Additionally, we consider

three modern BO algorithms. COMBO [16] has shown state-of-the-art performance on several combinatorial optimization functions. Hyperopt [14] is an implementation of Tree Parzen Estimator (TPE). SMAC [12] is a BO algorithm specifically designed to handle categorical variables by using an RF surrogate model. Finally, we compare the achieved ensemble performance to training one SVM model on all samples.

5.4.4. PROBLEM SIZES AND RUNTIME BUDGET

We consider various values of ℓ (i.e., the number of samples in the training dataset), and α (i.e., the number of subsets in a partition), specifically: $\ell \in \{100, 250, 500\}$ and $\alpha \in \{2, 5, 10\}$. On each dataset in each problem configuration (a combination of ℓ and α), all considered search algorithms perform 10 runs. Therefore, there are 50 runs by each algorithm for each problem instance.

As the global optima of the considered problems are not known, we compare the algorithms by the convergence trajectory of their averaged performance value. The averaged performance value after n real function evaluations is calculated as the elitist fitness function value after that many evaluations averaged over all 50 runs.

In our main setup, the fitness function is deterministic, and only unique dataset partitions are evaluated (as described in Section 5.3.1). P3-GOMEA, SA-P3-GOMEA, LS, and RS are run without a time limit until 5000 real function evaluations are performed. Additionally, we consider the case of noisy fitness evaluations, where ensemble training is stochastic, i.e., re-evaluating the same partition results in a different fitness. In that case, all evaluations are counted.

We found that the considered BO algorithms, especially COMBO and SMAC, are much slower than P3-GOMEA and SA-P3-GOMEA. We, therefore, set a time limit of 3 hours for BO algorithm runs for $\alpha = 2, 5$ and 5 hours for $\alpha = 10$. These values correspond to the observed upper bounds (rounded to hours) of SA-P3-GOMEA run time for corresponding problem instances. We found that even for the smallest considered problem ($\ell = 100, \alpha = 2$), COMBO could produce ℓ randomly solutions in the initialization phase, but only 3 additional solutions in the subsequent search phase after 4 hours of running. Hence, we do not include COMBO in the presented results as within the given time limit it works basically as RS.

5.4.5. IMPLEMENTATION DETAILS

Fitness functions for all conducted experiments are implemented in Python (version 3.7) with Scikit-Learn, and NumPy packages. All considered search algorithms use identical fitness functions. Code of COMBO[1], SMAC[2], and TPE[3] algorithms are downloaded from the corresponding repositories. P3-GOMEA, SA-P3-GOMEA, and LS algorithms are implemented in C++ with calls of Python fitness functions. All implementations are available at the Github repository [4]. Experiments are conducted on a machine with Intel(R) Xeon(R) Silver 4110 CPU.

[1] https://github.com/QUVA-Lab/COMBO
[2] https://github.com/automl/SMAC3
[3] https://github.com/hyperopt/hyperopt
[4] https://github.com/ArkadiyD/SAGOMEA

5.5. RESULTS

First, we analyze whether two hyperparameters, the relaxation hyperparameter η and the surrogate model type (both described in Section 5.2.4), have a significant effect on performance. We test all surrogate models with $\eta = 0.99$ and $\eta = 0.999$. We do not consider lower η values, as they imply a more elaborate use of real fitness evaluations, leading to a performance close to non-surrogate P3-GOMEA. These results are provided in Figure 5.5.1. We conclude that 1) while both η values provide solid performance, $\eta = 0.999$ is slightly preferable and hence, we use it in further experiments; 2) SVR provides the best performance among the considered types of surrogate models. RF is a close runner-up, while MLP and Gradient Boosting models perform worse. We hypothesize that the better performance obtained with SVR is because SVR is less susceptible to overfitting and can generalize well from a limited number of training samples (i.e., extrapolate to unseen solutions). Moreover, it has fewer tunable hyperparameters than, for instance, Gradient Boosting models, and it is feasible to carefully tune them using grid search. Therefore, in further experiments, when we refer to SA-P3-GOMEA, SVR is always used as the surrogate model.

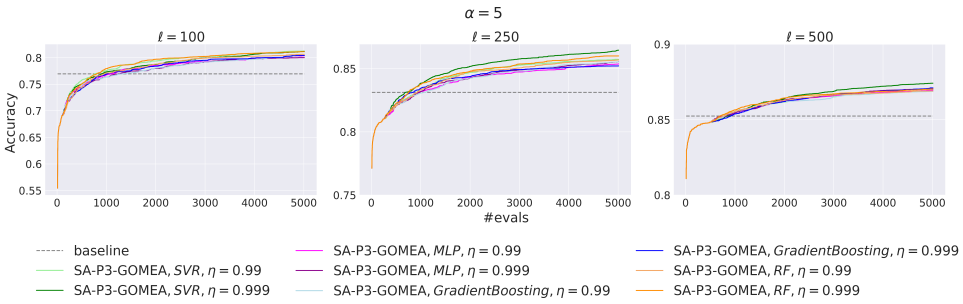


Figure 5.5.1. Convergence graphs of average performance for different SA-P3-GOMEA hyperparameters over five datasets and ten optimization runs for each dataset, for $\alpha = 5$ (partition into five subsets). Baseline denotes non-ensemble training using all samples in the training dataset, ℓ is the number of training samples, i.e., the number of variables in the optimization problem.

Next, we compare the results of SA-P3-GOMEA to other search algorithms (see Figure 5.5.2; Table 5.5.2). We observe that P3-GOMEA outperforms LS and RS in most cases. Moreover, the difference becomes larger as the number of variables increases. In general, SA-P3-GOMEA demonstrates solid performance on all problem instances. It always achieves the best accuracy within 500 evaluations, except for the smallest considered problem instance ($\ell = 100, \alpha = 2$), where after 2000 evaluations SMAC finds better solutions. We observe that SMAC does not scale well to larger problem instances as its computational overhead grows fast when both ℓ and α increase. In case of $\alpha = 10$ the problem apparently becomes too difficult within the given evaluations limit, as the achieved accuracy is below the baseline for $\ell = 100, 250$ and the performance of SA-P3-GOMEA is similar to P3-GOMEA. We suppose that in case of $\alpha = 10$ the surrogate modeling task is of too high dimensionality ($\ell\alpha$) and therefore, more solutions are needed to get accurate fitness function predictions. However, importantly, due to the adaptive nature of our new

surrogate mechanism, even when it is not possible to accurately estimate fitness values, the performance of SA-P3-GOMEA does not become worse than P3-GOMEA.

From the practical perspective of the goal of ensembling, it is further important to note that a better validation accuracy is achieved than the baseline in most cases. Though the accuracy decreases as α increases, we believe it is possible to find better solutions if more function evaluations are allowed.

5.5.1. RUNTIME

On problem instances with $\alpha = 2,5$ the runtime of SA-P3-GOMEA is below 2.5 hours. For $\alpha = 10$, runtime does not exceed 4.5 hours.

5.5.2. STATISTICAL SIGNIFICANCE TESTS

We employ statistical hypothesis testing to compare the performance of SA-P3-GOMEA with P3-GOMEA and TPE. For each value of α and each value of ℓ (3 values of α, 3 values of ℓ) we do a pairwise statistical test to test a hypothesis that one algorithm performs better than another. Performance is assessed as the best achieved real fitness value during a run. We use the Wilcoxon pairwise test with Holm correction, resulting in a corrected significance level of 0.1. The p-values are presented in Table 5.5.1. For most problem instances, SA-P3-GOMEA outperforms P3-GOMEA and TPE.

α	ℓ	**SA-P3-GOMEA** is better than **P3-GOMEA**	**SA-P3-GOMEA** is better than **TPE**
2	100	0.025	0.006
	250	0.0	0.0
	500	0.002	0.006
5	100	0.002	0.0
	250	0.0	0.0
	500	0.069	0.002
10	100	0.002	0.0
	250	0.023	0.0
	500	0.13	0.0

Table 5.5.1. Statistical significance testing of a performance difference in pairs of best-performing algorithms. Reported $p-$values are obtained by pairwise Wilcoxon test and are uncorrected. Statistically significant results at $\alpha = 0.1$ with Holm correction ($m = 9$) are typed in blue font color.

5.5.3. GENERALIZATION STUDY

In machine learning, a validation dataset is usually used for tuning hyperparameters, model selection, etc. In our setup, we search for the best ensemble configuration based on the accuracy score on the validation dataset. However, to evaluate the generalization ability of the obtained ensembles, we also look at the results on test datasets, which are held out during the search. These results are not related to the evaluation of the optimization potential of the considered algorithms but are important to assess the practical value of the considered ensembling technique. The results which are presented in Figure 5.5.3, demonstrate that convergence curves on test datasets are similar to the ones on validation datasets and SA-P3-GOMEA is the best performing algorithm in most cases.

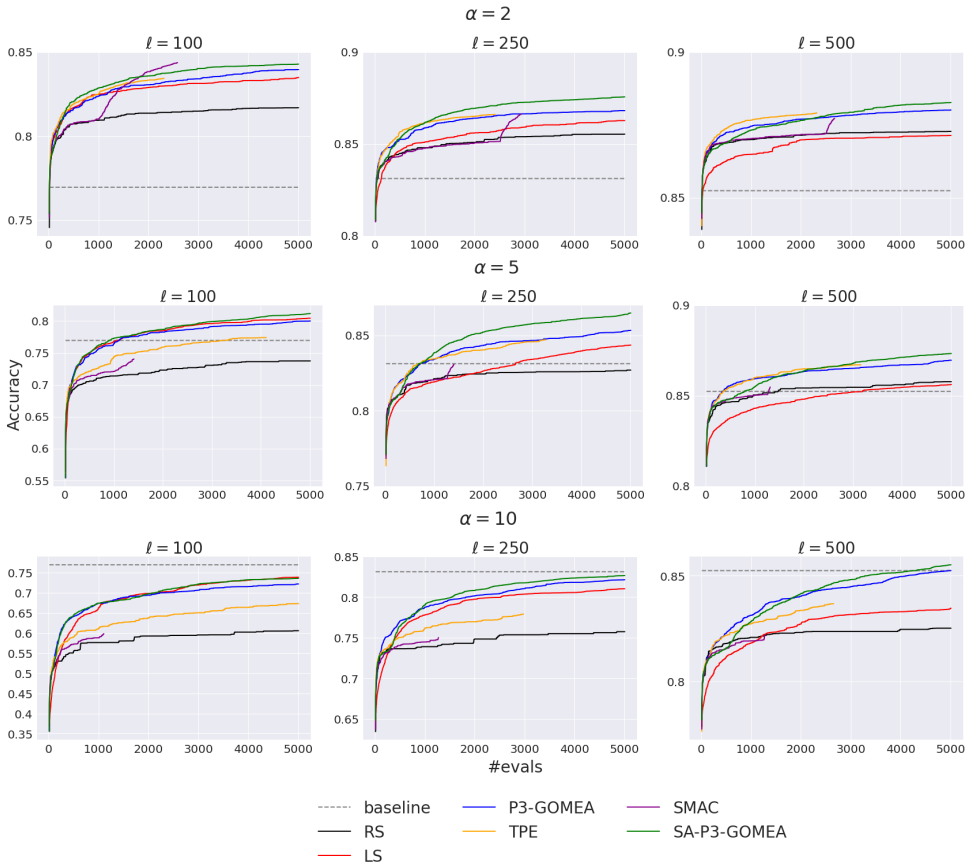


Figure 5.5.2. Convergence graphs of average performance for different values of α (the number of partition subsets). ℓ denotes the number of training samples, i.e., the number of variables in a corresponding optimization problem. Baseline denotes averaged validation accuracy when SVM is trained on all samples of training datasets of size ℓ. For TPE and SMAC, the values are presented for fewer than 5000 evaluations as these algorithms could not produce 5000 solutions within the time limit. Note that y-axis scales are different and depend on the accuracy range of the corresponding problem.

α	ℓ	dataset	P3-GOMEA	TPE	SA-P3-GOMEA
2	100	*segment*	**0.843**	0.842	0.83
		spambase	0.904	0.9	**0.912**
		wall-robot-navigation	0.713	0.715	**0.734**
		kc1	**0.865**	**0.865**	**0.865**
		optdigits	0.873	0.864	**0.874**
	250	*segment*	0.866	0.866	**0.868**
		spambase	0.918	0.916	**0.921**
		wall-robot-navigation	0.76	0.761	**0.78**
		kc1	0.869	0.87	**0.872**
		optdigits	0.929	0.928	**0.937**
	500	*segment*	0.857	**0.862**	0.86
		spambase	0.915	0.915	**0.916**
		wall-robot-navigation	0.802	0.801	**0.805**
		kc1	0.869	0.871	**0.872**
		optdigits	0.958	0.954	**0.96**
5	100	*segment*	0.798	0.769	**0.833**
		spambase	0.896	0.896	**0.897**
		wall-robot-navigation	0.697	0.691	**0.705**
		kc1	**0.865**	0.864	**0.865**
		optdigits	0.745	0.653	**0.758**
	250	*segment*	0.861	0.852	**0.869**
		spambase	0.905	0.906	**0.907**
		wall-robot-navigation	0.73	0.73	**0.76**
		kc1	0.867	0.868	**0.869**
		optdigits	0.904	0.892	**0.919**
	500	*segment*	**0.855**	0.851	**0.855**
		spambase	**0.912**	**0.912**	0.911
		wall-robot-navigation	0.779	0.777	**0.795**
		kc1	0.869	**0.87**	0.867
		optdigits	0.933	0.93	**0.938**
10	100	*segment*	0.695	0.654	**0.719**
		spambase	0.88	0.865	**0.884**
		wall-robot-navigation	0.659	0.629	**0.678**
		kc1	0.862	0.863	**0.864**
		optdigits	0.514	0.411	**0.535**
	250	*segment*	0.84	0.766	**0.85**
		spambase	**0.898**	0.895	0.895
		wall-robot-navigation	0.707	0.691	**0.715**
		kc1	0.85	0.844	**0.86**
		optdigits	0.811	0.714	**0.813**
	500	*segment*	**0.844**	0.83	0.842
		spambase	**0.904**	0.902	0.903
		wall-robot-navigation	0.739	0.718	**0.748**
		kc1	**0.864**	0.856	0.863
		optdigits	0.912	0.889	**0.919**

Table 5.5.2. Average accuracy of the elitist solutions (averaged over 10 performed runs) for different algorithms for all datasets and considered problem configurations. The best performing algorithms for each dataset and configuration are typed in bold font.

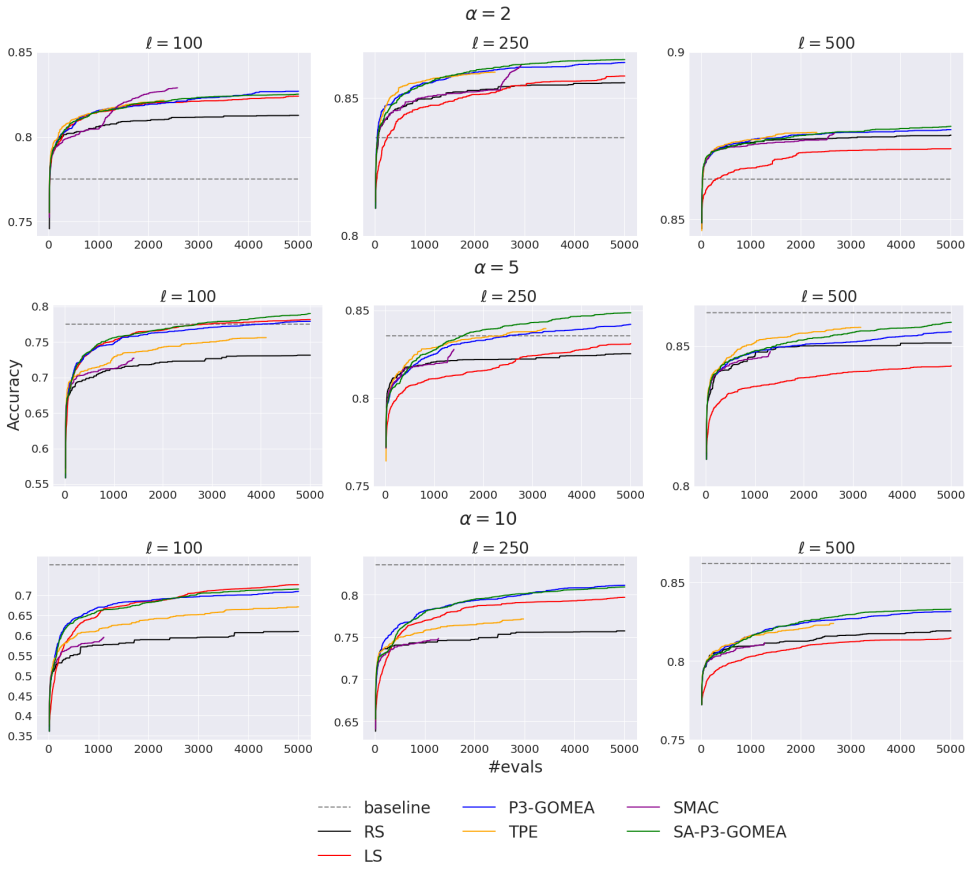


Figure 5.5.3. Averaged convergence results on *test* dataset for different values of α (the number of partition subsets). ℓ denotes the number of training samples, i.e., the number of variables in a corresponding optimization problem. Baseline denotes averaged validation accuracy when SVM is trained on all samples of training datasets of size ℓ. For TPE and SMAC, the values are presented for fewer than 5000 evaluations as these algorithms could not produce 5000 solutions within the given time limit. Note that y-axis scales are different and depend on the accuracy range of the corresponding problem.

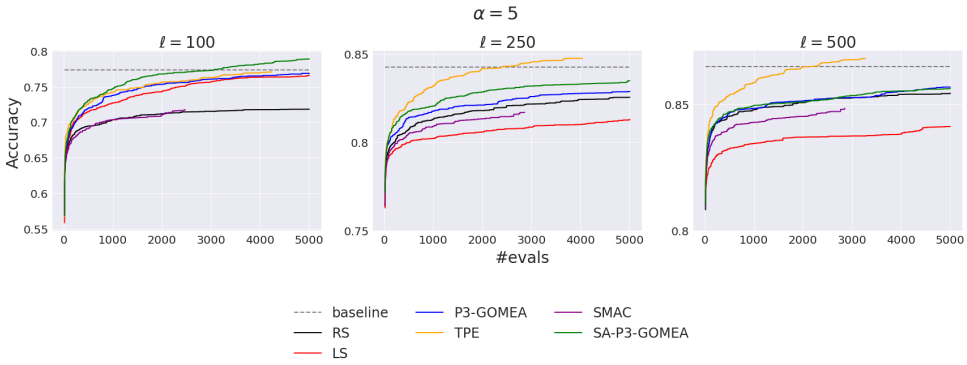


Figure 5.5.4. Averaged convergence results for the case of the *noisy* fitness function. ℓ denotes the number of training samples, i.e., the number of variables in a corresponding optimization problem. Baseline denotes averaged validation accuracy when SVM is trained on all samples of training datasets of size ℓ with 10 different random seeds for each dataset. SMAC is run with a specific setting for noisy fitness function handling. Note that y-axis scales are different and depend on the accuracy range of the corresponding problem.

5

model	hyperparameter	values
SVR	kernel	[rbf, sigmoid]
CatBoostRegressor	iterations	300
	learning_rate	[1.0, 0.1, 0.01]
	depth	[3, 6, 9, 12]
	early_stopping_rounds	5
MLP	activation_function	ReLU
	dropout	0.2
	n_hidden_layers	2
RF	num_trees	10
	min_samples_split	[1, 3, 10]
	min_samples_leaf	[1, 3, 10]
	ratio_features	[5/6, 1.0]

Table 5.5.3. Hyperparameters of considered types of surrogate models. Hyperparameters with multiple listed values are tuned using grid search. SVR model is Support Vector Regression implementation from Scikit-Learn Python library. CatBoostRegressor is Gradient Boosting Regression with categorical variables support from the CatBoost library. MLP is Multilayer Perceptron implementation using PyTorch. RF is Random Forest implementation from Pyrfr (the same library is used in SMAC). Hyperparameters that are not listed in the table are set to default values.

5.6. DISCUSSION AND FUTURE WORK

The main goal of this chapter was to introduce a novel efficient surrogate-assisted EA for solving combinatorial non-binary optimization problems with expensive function

evaluations by integrating a surrogate model into a P3-like EA through a new adaptive mechanism with only one hyperparameter. Though we found the value of $\eta = 0.999$ for this hyperparameter to work well for the considered problem, it might need tuning for different expensive optimization problems.

Below, we identify and discuss interesting research topics for future work. Through additional experiments, we also considered the case when fitness evaluations are noisy, i.e., when the same dataset partitioning can result in different ensemble accuracy scores because learners in ensembles are initialized with different random seeds. The results for $\alpha = 5$ are provided in the Figure 5.5.4. SA-P3-GOMEA performs better than P3-GOMEA for $\ell = 100, 250$ and not worse for $\ell = 500$. However, we observe that TPE performs significantly better for $\ell = 250, 500$. We note that, in contrast to TPE, neither P3-GOMEA nor SA-P3-GOMEA is specifically designed for solving noisy optimization problems. Potentially, specialized design choices aimed at solving noisy problems can improve their performance. However, the achieved accuracy values are much lower compared to the deterministic fitness function setup. We therefore conclude, that for practical usage, deterministic ensemble training is preferable. Not only does it make the search problem easier for all considered algorithms, better solutions with fewer function evaluations can ultimately be found.

We chose SVM models as learners in the ensemble due to the solid tradeoff of SVM between computational efficiency and good performance. To achieve the best possible predictive accuracy, one may want to use modern machine learning models such as Gradient Boosting, or a mixture of models of different types with carefully tuned hyperparameters. Using the proposed approach to achieve state-of-the-art performance on a supervised machine learning problem is an interesting potential future use case.

We did not conduct experiments with more expensive function evaluations, e.g., using deep learning models in an ensemble. As we aimed to study and analyze the performance of the introduced surrogate-assisted EA in multiple setups, including deep learning models was not computationally feasible. However, validating the performance of SA-P3-GOMEA with such learners in an ensemble for specific applications is considered important future research.

Related to this, all considered types of surrogate models are general-purpose regression models and are not designed specifically for the dataset partition problem. We believe that it is possible to further improve the performance of SA-P3-GOMEA if specialized surrogate models, capable of explicitly modeling the task at hand, are designed. Developing such specialized models is however non-trivial and therefore considered a separate research topic.

5.7. CONCLUSION

We have introduced a novel surrogate-assisted Evolutionary Algorithm for expensive combinatorial optimization problems based on a state-of-the-art model-based EA and a novel adaptive surrogate fitness evaluation control mechanism. The considered model-based EA is a variant of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) that uses a Parameter-less Population Pyramid (P3) scheme. To the best of our knowledge, this is the first time a surrogate model is integrated into a P3 scheme. We have demonstrated that SA-P3-GOMEA achieves state-of-the-art performance on the problem of partitioning

a dataset for the purpose of ensembling machine learning models. We experimented with different types of surrogate models for fitness values estimation and found that, among the models considered, SVR provides the best performance. The proposed SA-P3-GOMEA outperforms non-surrogate-assisted P3-GOMEA, local search, and various Bayesian Optimization algorithms on various ensemble learning problems with up to 500 variables of cardinality 10.

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO). We thank the Maurits en Anna de Kock Foundation for financing a high-performance computing system.

REFERENCES

[1] J. P. C. Kleijnen, W. van Beers, and I. van Nieuwenhuyse. “Constrained optimization in expensive simulation: novel approach”. In: *European Journal of Operational Research* 202.1 (2010), pp. 164–174.

[2] S. Koziel and L. Leifsson. “Efficient knowledge-based optimization of expensive computational models using adaptive response correction”. In: *Journal of Computational Science* 11 (2015), pp. 1–11.

[3] K. Rashid, W. J. Bailey, B. Couet, D. Wilkinson, et al. “An efficient procedure for expensive reservoir-simulation optimization under uncertainty”. In: *SPE Economics & Management* 5.04 (2013), pp. 21–33.

[4] T. Elsken, J. H. Metzen, and F. Hutter. “Neural architecture search: a survey”. In: *Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[5] T. Den Ottelander, A. Dushatskiy, M. Virgolin, and P. A. N. Bosman. “Local search is a remarkably strong baseline for neural architecture search”. In: *Proceedings of the Evolutionary Multi-Criterion Optimization: 11th International Conference*. Springer. 2021, pp. 465–479.

[6] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. “Understanding and simplifying one-shot architecture search”. In: *Proceedings of the International Conference on Machine Learning*. 2018, pp. 550–559.

[7] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4780–4789.

[8] G. Ghiasi, T.-Y. Lin, and Q. V. Le. “NAS-FPN: learning scalable feature pyramid architecture for object detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2019.

[9] Z. Mirikharaji, K. Abhishek, S. Izadi, and G. Hamarneh. “D-LEMA: deep learning ensembles from multiple annotations-application to skin lesion segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 1837–1846.

[10] E. C. Garrido-Merchán and D. Hernández-Lobato. “Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes”. In: *Neurocomputing* 380 (2020), pp. 20–35.

[11] L. Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–32.

[12] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Sequential model-based optimization for general algorithm configuration”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 507–523.

[13] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. “Algorithms for hyper-parameter optimization”. In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[14] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. “Hyperopt: a Python library for model selection and hyperparameter optimization”. In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[15] R. Baptista and M. Poloczek. “Bayesian optimization of combinatorial structures”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2018, pp. 462–471.

[16] C. Oh, J. Tomczak, E. Gavves, and M. Welling. “Combinatorial Bayesian optimization using the graph Cartesian product”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 2914–2924.

[17] A. Diaz-Manriquez, G. Toscano Pulido, J. Barron-Zambrano, and E. Tello. “A review of surrogate assisted multiobjective evolutionary algorithms”. In: *Computational Intelligence and Neuroscience* 2016 (June 2016), pp. 1–14.

[18] C. White, S. Nolen, and Y. Savani. “Local search is state of the art for NAS benchmarks”. In: *arXiv preprint arXiv:2005.02960* (2020).

[19] L. Breiman. “Bagging predictors”. In: *Machine Learning* 24.2 (1996), pp. 123–140.

[20] E. Alpaydin. “Voting over multiple condensed nearest neighbors”. In: *Lazy learning*. Springer, 1997, pp. 115–132.

[21] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. “Learning ensembles from bites: a scalable and accurate approach”. In: *Journal of Machine Learning Research* 5 (2004), pp. 421–451.

[22] B. W. Goldman and W. F. Punch. “Fast and efficient black-box optimization using the parameter-less population pyramid”. In: *Evolutionary Computation* 23.3 (2015), pp. 451–479.

[23] P. A. N. Bosman and D. Thierens. “Linkage neighbors, optimal mixing and forced improvements in genetic algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 585–592.

[24] D. Thierens and P. A. N. Bosman. “Optimal mixing evolutionary algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2011, pp. 617–624.

[25] N. H. Luong, M. O. W. Grond, H. La Poutré, and P. A. N. Bosman. “Scalable and practical multi-objective distribution network expansion planning”. In: *2015 IEEE Power & Energy Society General Meeting*. IEEE. 2015, pp. 1–5.

[26] P. A. N. Bosman and D. Thierens. “More concise and robust linkage learning by filtering and combining linkage hierarchies”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2013, pp. 359–366.

[27] W. den Besten, D. Thierens, and P. A. Bosman. “The multiple insertion pyramid: a fast parameter-less population scheme”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2016, pp. 48–58.

[28] K. S. Bhattacharjee, H. K. Singh, T. Ray, and J. Branke. “Multiple surrogate assisted multiobjective optimization using improved pre-selection”. In: *2016 IEEE congress on evolutionary computation (CEC)*. IEEE. 2016, pp. 4328–4335.

[29] T. Chen and C. Guestrin. “Xgboost: a scalable tree boosting system”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 785–794.

[30] J. Zhong et al. “XGBFEMF: an XGBoost-based framework for essential protein prediction”. In: *IEEE Transactions on NanoBioscience* 17.3 (2018), pp. 243–250.

[31] Z. Chen et al. “XGBoost classifier for DDoS attack detection and analysis in SDN-based cloud”. In: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2018, pp. 251–256.

[32] A. V. Dorogush, V. Ershov, and A. Gulin. “CatBoost: gradient boosting with categorical features support”. In: *arXiv preprint arXiv:1810.11363* (2018).

[33] B. Bischl et al. “OpenML benchmarking suites and the OpenML100”. In: *arXiv preprint arXiv:1708.03731* (2017).

6

HEED THE NOISE IN PERFORMANCE EVALUATIONS IN NEURAL ARCHITECTURE SEARCH

Neural Architecture Search (NAS) has recently become a topic of great interest. However, there is a potentially impactful issue within NAS that remains largely unrecognized: noise. Due to stochastic factors in neural network initialization, training, and the chosen train/validation dataset split, the performance evaluation of a neural network architecture, which is often based on a single learning run, is also stochastic. This may have a particularly large impact if a dataset is small. We, therefore, propose to reduce this noise by evaluating architectures based on average performance over multiple network training runs using different random seeds and cross-validation. We perform experiments for a combinatorial optimization formulation of NAS in which we vary noise reduction levels. We use the same computational budget for each noise level in terms of network training runs, i.e., we allow fewer architecture evaluations when averaging over more training runs. Multiple search algorithms are considered, including evolutionary algorithms, which generally perform well for NAS. We use two publicly available datasets from the medical image segmentation domain, where datasets are often limited and variability among samples is often high. Our results show that reducing noise in architecture evaluations enables finding better architectures by all considered search algorithms.

The contents of this chapter are based on the following publication: **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "Heed the noise in performance evaluations in neural architecture search". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM. 2022, pp. 2104–2112.

6.1. INTRODUCTION

6.1.1. NEURAL ARCHITECTURE SEARCH

Neural Architecture Search (NAS), i.e., the automated design of Neural Networks (NNs) architectures tailored for a specific task, has become a topic of great interest recently. The main reason for that is the growing number of ideas in NN design, many of which demonstrate great performance. However, with this growth, it is becoming more difficult to guess without running experiments which network would be the best for a given task. This makes automated network design a natural research topic, positioned in between deep learning and optimization algorithms.

Optimization algorithms used for NAS include Evolutionary Algorithms (EAs) [1, 2, 3], Bayesian optimization algorithms using performance predictors [4, 5], gradient descent-based methods [6, 7], reinforcement learning algorithms [8], and Local Search (LS) [9, 5]. Gradient descent-based methods use a so-called supernetwork, of which the structure is optimized using a gradient descent optimizer simultaneously with the network weights. However, it was shown in [10] that the performance of such methods is often suboptimal, in some cases not better than the most simple search approach - random search.

NAS can be extremely computationally expensive if the search relies on training numerous networks. To reduce the computational costs, part of the network trainings can be replaced by a computationally cheaper performance estimation made by performance predictors (also called surrogate models). However, developing a powerful performance predictor for NAS might be challenging and search space specific [11].

6.1.2. MEDICAL IMAGE SEGMENTATION

Segmentation is one of the major tasks in computer vision. Given an image, the task is to automatically perform a specific pixel-wise classification that outlines certain things in the image. Medical Image Segmentation (MIS) is a special case with input images being medical scans, such as Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) acquired from a region of interest determined by a medical expert. Examples of common MIS tasks are organ, tumor, and vessel segmentation. Designing a precise and fast MIS algorithm can be immensely beneficial for healthcare as potentially it can not only reduce the workload of physicians (for manual scan segmentation) but also make some medical procedures faster which may be beneficial for patients as well. In some cases, segmentation performed by a deep NN can demonstrate human-level performance [12]. Most of the proposed architectures are adaptations of U-Net [13]. The main idea of U-Net is an encoder-decoder structure that, firstly, extracts features from the image, and then, translates them into a segmentation. Despite the good performance of U-Net in general, due to variability in both image quality and expert input segmentations, finding the best network architecture for a specific task is still challenging, and therefore, NAS for MIS has high practical value.

6.1.3. NEURAL ARCHITECTURE SEARCH FOR MEDICAL IMAGE SEGMENTATION

Research on NAS for MIS has so far been less elaborate than on NAS for classification. However, several works have shown that automatically found networks perform better than the state-of-the-art manually designed ones [7, 3, 14, 15]. In general, the search can be performed on two levels: 1) configuration and combinations (also called *a cell*) of atomic operations in a network (e.g., number and kernel size convolutions, or activation functions); 2) network topology, i.e., defining the connections between cells, and input /output tensor dimensionalities for them. In [14] NAS-Unet was introduced, the main idea of which is to search for the configuration of cells in the fixed U-Net structure. The best found network demonstrated better performance than a standard U-Net. In [3] it was shown that a bilevel search of an U-Net-like network topology at the first level and the structure of cells at the second level, can find even better-performing networks than NAS-Unet. A simultaneous search of cell structure and network topology was performed in [7], also showing better performance than NAS-Unet. This suggests that network topology and the configuration of cells are connected. In [15] state-of-the-art performance was achieved on 10 MIS tasks from the Medical Segmentation Decathlon challenge [16] by a newly proposed method called nnUnet. However, it can be considered to be a semi-NAS method as, while the networks are automatically configured for each task, the number of options is very limited and the network construction is rule-based, determined by the dataset properties such as the resolution of the scans.

6.1.4. POTENTIALLY IMPACTFUL ISSUE: NOISE

NAS, just as any search task, needs a definition of a function that maps a solution (i.e., a candidate network architecture) to a performance score. It is common practice in NAS to evaluate a network (assign a performance score) using a fixed validation set. Moreover, in most cases, the evaluation is deterministic, meaning that only one random seed for the initialization of network weights and stochastic training components (i.e., applied augmentations and sampling of batches) is used.

In our own experiments with NAS, we have noticed that NN network performance can depend a lot on the chosen random seed and train/validation dataset split, especially (in a relative sense) when considering well-performing architectures. A demonstration of this for the case of MIS is given in Figure 6.1.1. The Spearman rank correlation between networks trained with two different random seeds is poor (< 0.1) when calculated for the top 20% of the networks. A similar result applies to networks trained with two different train/validation data splits. Moreover, the best network according to one random seed or one train/validation data split does not correspond to the best network when another seed or train/validation data split is used. This means that using a standard evaluation procedure (one seed and one train/validation data split) might lead to finding suboptimal networks, of which the true performance (e.g., on an independent test set or obtained by cross-validation with several random seeds if a test set is not available) is not as good as expected. Similar results are shown for the case of NAS for classification using the NAS-101 benchmark [17] in Figure 6.1.2.

In this chapter, we study whether increasing the network performance evaluation reliability by using cross-validation and multiple random seeds leads to a better quality

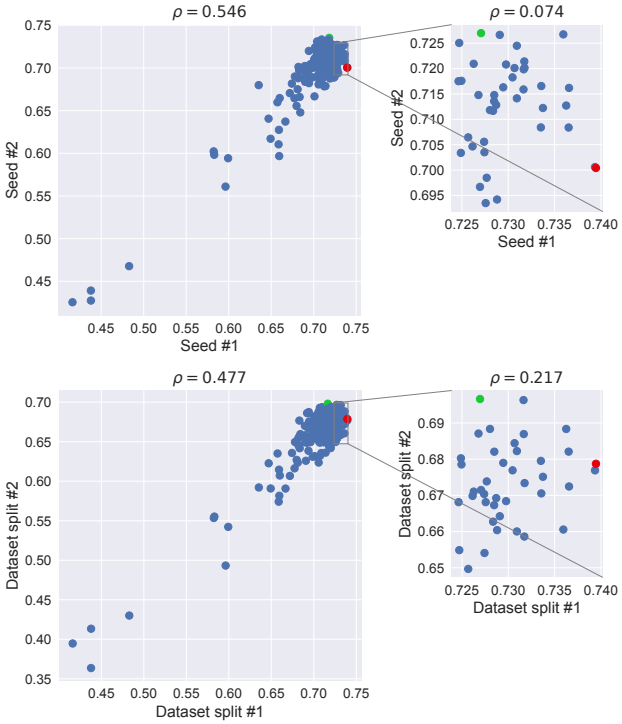


Figure 6.1.1. Differences in performance of NNs trained with different random seeds (upper plot) and different train/validation data splits (lower plot). Each dot represents one architecture. Zoomed-in plots correspond to the top 20% of the networks (that will likely be discovered in the later stages of the search process). Spearman rank correlation is denoted by ρ. The dataset is the Prostate segmentation dataset from the Medical Segmentation Decathlon [16]. The performance metric displayed in both axes is the Dice coefficient (see definition in Section 6.2.2). Red and green dots denote the best architectures according to each seed (top) and each train/validation data split (bottom). Networks in the plots are collected during a run of the SAGOMEA search algorithm (see Section 6.2.1) with a budget of 200 function evaluations.

of networks found by NAS. We use different performance evaluation setups as fitness functions for various search algorithms to understand whether more computationally expensive network evaluations lead to better generalization. To the best of our knowledge, this is the first time such a study is conducted. In contrast to just measuring network performance using different training random seeds (as, for instance, in NAS-101 [17], NAS-201 [18]), we study how more reliable performance evaluation setups affect the found network quality when measured in an *independent* performance evaluation. We focus on NAS for MIS as medical image datasets used for training segmentation models are often small (tens of scans) and the generalization problem, i.e., the problem of results transferability obtained for a particular training random seed, or validation subset, might have a more substantial impact on NAS.

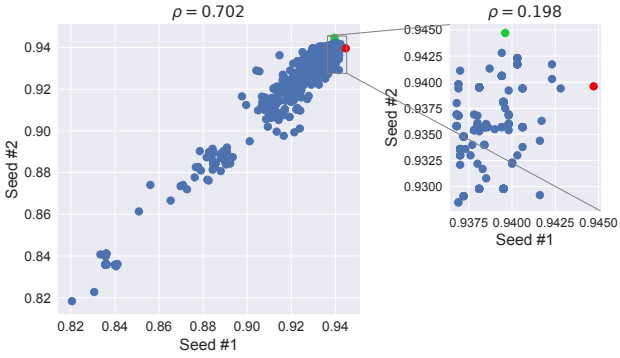


Figure 6.1.2. Differences in performance of NNs trained with different random seeds from the NAS-101 tabular benchmark (CIFAR-10 dataset). Each dot represents one architecture. The zoomed-in plot corresponds to the top 20% of the networks (that will likely be discovered in the later stages of the search process). Spearman rank correlation is denoted by ρ. The performance metric displayed in both axes is validation accuracy. Red and green dots denote the best architectures according to each seed. Networks in the plot are collected during a run of the local search algorithm (see Section 6.2.1) with a budget of 1000 function evaluations.

6.2. NAS METHOD

To use a NAS method, a search algorithm and a fitness function need to be defined. The choice of search algorithm depends on the chosen NAS task formulation (e.g., a combinatorial optimization paradigm). The definition of the fitness function depends on the chosen performance evaluation strategy and the selected segmentation quality metric.

6.2.1. SEARCH ALGORITHMS

The first considered search algorithm is Local Search (LS). It is a simple search approach, but it was shown to perform better in various NAS cases than random search and in some cases even be on par with advanced EAs [9, 5]. LS works by iterating over variables in random order and greedily choosing the best option for each variable. Evolutionary algorithms are powerful general-purpose search algorithms and have also successfully been applied to NAS, see, e.g., [19, 2]. Therefore, the next algorithm we consider is the parameterless version of the state-of-the-art EA Gene-pool Optimal Mixing Evolutionary Algorithm (P3-GOMEA) (Chapters 5,2 further referred to as GOMEA). P3-GOMEA is a model-based EA that attempts to detect and exploit linkage information during optimization. It was shown to perform better than other EAs on a range of problems (Chapter 5), and, importantly, it does not have a population size hyperparameter which needs to be tuned in many other EAs. Further, we consider surrogate-assisted GOMEA (SAGOMEA): a modification of GOMEA which uses a surrogate model for cheap fitness estimation. SAGOMEA was designed specifically for discrete optimization problems with computationally expensive fitness functions and has been shown to be efficient (Chapter 5). Finally, we use a Bayesian optimization algorithm called Tree Parzen Estimator (TPE). Specifi-

cally, we use its implementation in the Hyperopt optimization package [20]. Similarly to SAGOMEA, TPE was designed for problems with expensive fitness functions.

6.2.2. SEGMENTATION QUALITY METRICS

To use the above-described search algorithms, a fitness function needs to be defined. The Dice score is a commonly used metric for medical image segmentation quality evaluation. There exist other metrics to evaluate segmentation quality (Surface Dice coefficient, Hausdorff distance), but in this chapter, we focus on the most commonly used in literature Dice coefficient. Moreover, its calculation is computationally cheap and independent of the voxel spacing of scans. For the Dice coefficient calculation, a reference segmentation R and the predicted segmentation P need to be binary segmentation maps of dimensionality $C \times X \times Y \times Z$ (where C is the number of segmentation classes, and $X \times Y \times Z$ is the scan size), i.e., the value one in position (c, x, y, z) means that the voxel with coordinates (x, y, z) belongs to class c. We use the average Dice of multiple classes, which is defined by the formula $\frac{1}{C-1}\sum_{1}^{C}\frac{2|R_c \cap P_c|}{|R_c|+|P_c|}$. Note that the zero class which is usually the image background is not included in the Dice calculation.

6.2.3. NAS TASK FORMULATION

We formulate the NAS task as a combinatorial optimization problem (maximization) with the search space consisting of possible network architectures encoded with discrete variables and the fitness function being a network segmentation performance (in our case: the Dice coefficient). Ultimately, the goal of NAS is to find an architecture that performs well on unseen (during search) data. Therefore, it is a common practice in NAS, to divide the dataset into train, validation, and test parts. In each fitness function evaluation of NAS, a network is trained (from scratch) on the train subset, and its performance is evaluated on the validation subset. After obtaining the best-performing network according to the validation set, i.e., at the end of NAS, its performance is verified on the test subset. This way, it is checked whether overfitting to the specific validation subset took place. However, in the case of small medical datasets, performance on a single test subset might be not sufficiently indicative. Therefore, we do not use a separate test set. Instead, for the final evaluation procedure (to obtain the true performance measure), we retrain the networks from scratch using an averaged cross-validation performance on previously (during search) unseen cross-validation data splits and random seeds different from the ones used for training during optimization.

6.2.4. NETWORK PERFORMANCE EVALUATION

The most basic way to evaluate network performance (i.e., map a solution to a fitness score), is to calculate the Dice score on a single validation set using one random seed for network initialization and training. However, due to the differences between medical scans, the performance of networks may vary a lot depending on the validation dataset. Moreover, due to the stochastic nature of network training and initialization, performance may depend on the random seed used for weights initialization and training. This may affect NAS. To mitigate this problem, we investigate different possible evaluation procedures. The first considered procedure is a basic one that is mostly used in NAS literature: only one random seed and a fixed validation set (**Setup-1Fold**). The second proposed

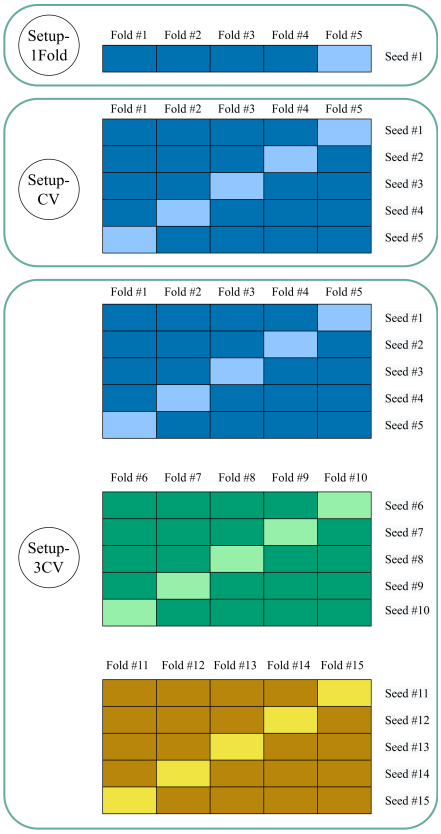


Figure 6.2.1. Different setups for evaluating architecture performance. Each row denotes one training. In each row, a lighter rectangle denotes the used validation subset, while darker ones constitute a training subset. The second and the third data splits used in Setup-3CV (with fold numbers 6-10, 11-15) denote different cross-validation data splits (this is also shown with different colors) than the one with the fold numbers 1-5. In total, Setup-1Fold requires one network training, Setup-CV 5, and Setup-3CV 15.

evaluation procedure uses 5-fold cross-validation (CV) instead of a single validation set (**Setup-CV**). In order to address both of the above-mentioned reasons for performance variance, different random seeds are used for network initialization and training in each of the cross-validation folds. The most computationally expensive considered evaluation procedure repeats 5-fold cross-validation using three different data partitionings into folds (**Setup-3CV**). These three evaluation procedures are shown schematically in Figure 6.2.1.

6.3. SEARCH SPACE

In choosing the search space for our experiments, we followed three main design principles: 1) The search space should contain networks that perform reasonably well, prefer-

ably better than a U-Net. 2) The search space should be large and contain diverse networks to make the search a non-trivial task. 3) The search space should not contain networks that are prohibitively computationally expensive to train. We decided to adopt the search space used in [3] while making it more versatile to meet all these criteria. Such a search space allows NAS formulation as a discrete combinatorial problem, is reasonably large, and in [3] promising results are shown. Instead of a bi-level search, we search simultaneously for the architecture topology and cells. Secondly, we allow different cells instead of repeating the same structure in all positions. Finally, to enlarge the search space and avoid the situation that the majority of networks are considered infeasible, we lift the architectural restrictions which were applied in [14]: in contrast to a fixed encoder-decoder structure, we allow all possible architectures from the topology space described below.

6.3.1. TOPOLOGY SEARCH SUBSPACE

The topology part of the search space determines the general structure of the network. It is shown in Figure 6.3.2. The network topology is defined by a sequence of N cells, which can be on different levels, i.e., $S = (l_0, l_1, \ldots, l_{N-1})$. At level i, feature maps have the dimensionality $\left(D * 2^i, \frac{W'}{2^i}, \frac{H'}{2^i}\right)$, where D is the number of channels of the network input (after the stem convolution is applied, see Section 6.3.3), and (W', H') is its spatial dimensionality. We allow only one level change between the consecutive cells: i.e., $|l_i - l_{i+1}| \leq 1$. Therefore, to encode the network topology S, it is sufficient to specify whether to increase the level (from l_i to $l_i + 1$) for the next cell (*downsampling*), decrease (*upsampling*), or keep it at the previous level (*normal*). The topology search space encoding is, therefore, a vector of size N: $\{0, 1, 2\}^N$, where 0, 1, and 2 encodes normal, downsampling, and upsampling cells respectively. Note that some vectors from such search space represent infeasible architecture topologies as 1) it is not possible to apply an upsampling operation from the first level, and 2) a downsampling operation cannot be applied from the maximum allowed level. The most straightforward way to fix a vector representing an infeasible architecture is to change infeasible downsampling or/and upsampling operations to normal ones.

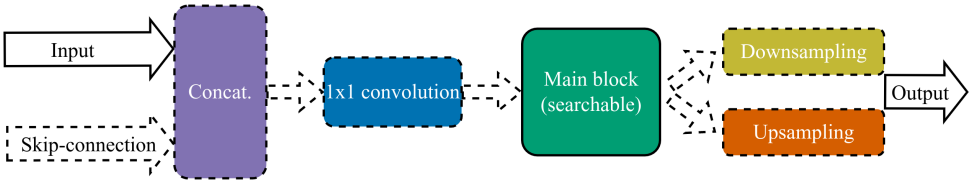


Figure 6.3.1. The network cell structure. The dashed parts are optional. Downsampling and upsampling operations depend on the network topology, while a skip-connection is always applied when possible (see Section 6.3.2). When a skip-connection is applied, the corresponding feature maps are firstly concatenated (Concat. block) with the input feature maps and then passed through a 1×1 convolution to keep the spatial dimensions equal to the input dimensions.

6.3.2. CELLS SEARCH SUBSPACE

The structure of a cell is shown in Figure 6.3.1. Each cell consists of its main block, optional downsampling or upsampling operations, and an optional skip-connection from one of the previous cells. As the necessity for downsampling or upsampling is encoded in the topology subspace, and the skip-connections presence is fixed (see Section 6.3.3), only block types need to be additionally encoded.

For each cell block, we consider five different options, four of which are non-trivial and one is an identity operation, the purpose of which is to act as a placeholder and allow more lightweight networks. The first type of block is a VGG block [21], which is also used in a standard U-Net [13]. In addition to this type of block, we consider three blocks that are used in classification networks that exhibit top performance: a block with a residual connection [22, 23] (ResNet block; a block that uses a concept of a depth-wise separable convolution [24] (Xception block); and a block used in one of the state-of-the-art classification networks, namely, Efficient-net [25]. We believe that the selected options can, firstly, contribute to a good performance in segmentation tasks, and, secondly, make the architectures in the search space diverse as different blocks can be used in different parts of the network.

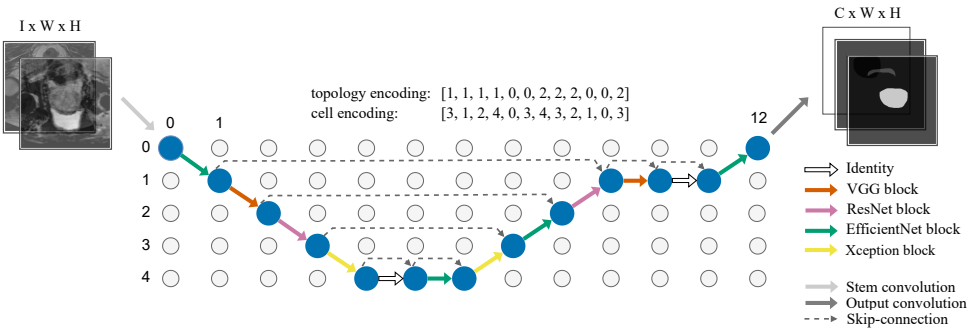


Figure 6.3.2. The scheme of network architectures in the proposed search space and an example U-Net-like architecture. Each circle represents feature maps (a three-dimensional tensor), the blue ones are used in the shown architecture. Feature maps at depth j (from 0 to 4) have 2^j smaller spatial resolution and 2^j more channels than the feature maps after applying stem convolution (in the left upper corner). Each architecture is defined as a sequence of $N = 12$ cells (represented by arrows). Each cell is defined by two discrete variables: its main block specification (five different options), and its type: normal (horizontal arrows), downsampling (downward arrows), or upsampling (upward arrows). Skip-connections are added using fixed rules (see Section 6.3.3). A stem convolutional block translates an input image to feature maps with 32 channels. An output convolutional block translates corresponding feature maps to segmentation masks.

After performing block operations, downsampling cells perform convolution with stride 3 and stride 2. In the upsampling cells, the upsampling operation is s transpose convolution with kernel size 3.

Similar to [3], skip-connections are added to the input of a cell in two cases: 1) The cell follows an upsampling cell. Then, a skip-connection is added from the previous cell

on the same level. 2) There is a cell before the previous one on the current level of input. Then, a skip-connection is added from that cell.

6.3.3. SEARCH SPACE DETAILS

Just as done in related literature [3, 6], we set the maximum network depth to $N = 12$. Note, however, that in contrast to [3], we allow the number of effective cells to be lower due to possible identity blocks. The maximum cell level is 5, this value was also used in [3]. In total, our search space encoding consists of $N = 12$ discrete variables which encode the network topology, and 12 variables that encode the corresponding block type for each cell in the network. The topology-related variables have a cardinality of 3, while the cell type variables have a cardinality of 5.

6.4. EXPERIMENTS

First, we compare different search algorithms in terms of performance on the given optimization problems. Secondly, we compare how the quality of the found architectures after independent re-training and re-evaluation depends on the used performance evaluation approach. Statistical tests (Wilcoxon test with Bonferroni correction, $\alpha = 0.05$) are conducted to verify the results. Then, we analyze the reasons for the observed differences. Finally, we compare the performance of the best found networks to commonly used handcrafted architectures.

6.4.1. EXPERIMENTAL SETUP

PERFORMANCE EVALUATION SETUPS

We consider three setups for the evaluation of neural network performance during the search (i.e., the fitness functions) as described in Section 6.2.4: 1) Training on one seed and using one dataset split (Setup-1Fold); 2) Using 5-fold cross-validation (Setup-CV); 3) Using 5-fold cross-validation on three different partitionings of the folds (Setup-3CV). For each of the setups and each of the algorithms (LS, GOMEA, TPE, SAGOMEA) we perform five runs per dataset. These runs differ in both the random seed used by the search algorithm and the seeds used for the network performance evaluation.

HANDLING INFEASIBLE SOLUTIONS

For all search algorithms, the same strategy of handling solutions that encode infeasible architectures is applied: before evaluating fitness, an architecture is checked for feasibility, and, if necessary, repaired as described in Section 6.3.1. Note that the algorithms receive a fitness score for infeasible solutions, i.e., the repaired genotype is only used for evaluation; it does not replace the original infeasible genotype. Such a choice was made in order to fairly compare different search algorithms without modifying them for infeasible solutions handling.

INDEPENDENT PERFORMANCE MEASURE

The final evaluation metric (also further referred to as *architecture quality*) is the performance of the architecture obtained after retraining from scratch with three different 5-fold cross-validations (basically, as in the Setup-3CV). Importantly, both random seeds

and cross-validation splits in this independent training and evaluation procedure do not overlap with the seeds and splits used during the search phase. While averaging only three cross-validations preserves some amount of noise in the score, we did not observe a substantial change in results if more than three cross-validations are used, and, therefore, we stick to the computationally cheaper procedure of using three different cross-validations.

COMPUTATIONAL BUDGET

For all setups we allocate an equal computational budget in terms of network trainings. We study performance under four different computational budgets: *T, 2T, 4T, 8T*. Due to computational time constraints, the largest considered budget of 8T comprises 3000 network trainings. Smaller budgets of T, 2T, 4T comprise 375, 750, 1500 network trainings correspondingly. With the Setup-1Fold, one fitness evaluation entails one network training. Thus, given, for instance, budget T, search algorithms perform T fitness evaluations in each optimization run. As the Setup-CV entails 5 network trainings in each fitness evaluation, search algorithms perform T/5 fitness evaluations given budget T (which equals to the same number of network trainings, namely, T). With the Setup-3CV, which is 15 times more computationally expensive than the first one, search algorithms can perform only T/15 fitness evaluations under budget T. While running search algorithms, we count only actual network trainings by storing all trained networks (after repair) along with their measured performance in an archive.

6.4.2. DATASETS

In this chapter, we perform experiments on two publicly available medical image segmentation datasets. The first segmentation task is prostate segmentation taken from the Medical Segmentation Decathlon challenge [16]. This dataset consists of 30 MRI scans (two scans with ids 18 and 32 were removed from the original dataset due to suspected label inconsistencies compared to the other scans) comprising two modalities: T2-weighted and ADC. The segmentation classes are background, prostate peripheral zone, and prostate transition zone. The second dataset is from the Automated Cardiac Diagnosis Challenge (ACDC) [26]. It contains MRI scans (in one modality) of 100 patients and the task is multi-class segmentation (left ventricular endocardium, myocardium and right ventricular endocardium). We keep only one scan per patient (from the diastole phase) to make the validation process easier, i.e., the total considered number of scans is 100. These datasets were chosen due to their diversity (were collected for different segmentation challenges, are focused on different organs) and relatively low scan resolution which allows for faster experiments.

6.4.3. PREPROCESSING AND TRAINING

We adopt the preprocessing resampling and voxel value normalization) and training setups used in the nnUnet framework [15] as it demonstrates state-of-the-art performance on various datasets. The loss function for training is a sum of soft Dice loss and cross-entropy. The optimizer is Stochastic Gradient Descent (SGD) with Nesterov momentum and weight decay. Momentum and weight decay values are 0.99 and, $3 \cdot 10^{-5}$ respectively. The initial learning rate is 0.01 and a polynomial decay learning schedule is used. To avoid

overfitting, data augmentations are used with magnitude and probability values adopted from the nnUnet. For computational efficiency reasons, in our main experiments for training we use patches of size 128 × 128 pixels which are randomly sampled from the original images. The training is performed for 40 epochs. This value was chosen as a trade-off between training time and network performance. In order to minimize noise coming from one of the performance factors that we do not focus on here, i.e., different performance scores after a slightly different number of training epochs, we average the segmentation prediction (which is then evaluated) over the last five epochs.

6.4.4. IMPLEMENTATION DETAILS

Neural network training and evaluation is implemented in Python using Pytorch. LS, GOMEA, and SAGOMEA are implemented in C++. The source code is provided at [1]. Experiments are conducted on a system with Nvidia A100 GPUs. One full network training (40 epochs) and evaluation takes ≈ 2 minutes. Therefore, one search run with the largest budget of 8T takes approximately 100 GPU hours.

6.5. RESULTS

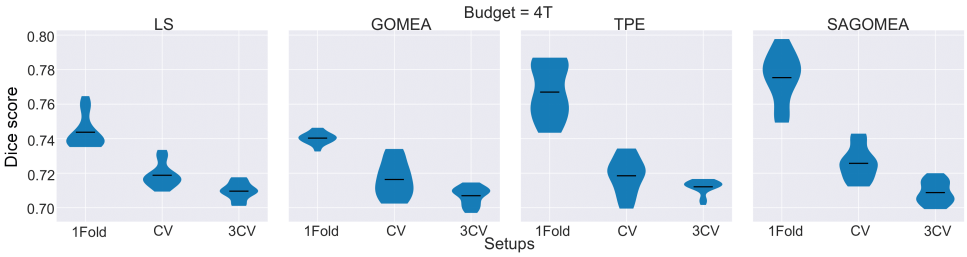


Figure 6.5.1. Comparing optimization performance of search algorithms in different setups with a computational budget of 4T (as described in Section 6.4.1). The image shows distributions (over five runs, five best architectures from each run) of search results. Note that these results take into account only fitness function values, not the final architecture's quality.

6.5.1. SEARCH PERFORMANCE

First, we study how different search algorithms perform on given optimization tasks. These results are shown in Figure 6.5.1 (Prostate dataset, budget 4T), and in Figure 6.5.4 (budgets T and 2T). All results are also provided in tabular form in Table 6.5.2. LS and GOMEA are approximately equal in optimization quality on all three setups. TPE finds solutions with better average fitness values than LS and GOMEA on all three setups. The difference is more substantial with Setup-1Fold. SAGOMEA outperforms TPE on Setup-1Fold and Setup-CV, however, it slightly underperforms on Setup-3CV. Statistical significance testing results are provided in Table 6.5.4. With a budget of 4T, TPE performs better than LS and GOMEA on Setup-1Fold and Setup-3CV with statistical significance. SAGOMEA performs better than LS, GOMEA, and TPE on Setup-1Fold and Setup-CV with

[1] https://github.com/ArkadiyD/Noise_in_NAS

statistical significance. These results demonstrate that algorithms that use surrogate models and were designed for expensive optimization problems have great potential in NAS. In further experiments (the second dataset and larger budget) we use only SAGOMEA and TPE algorithms as they demonstrate better ability to solve the considered NAS optimization task.

6.5.2. QUALITY OF FOUND NETWORKS

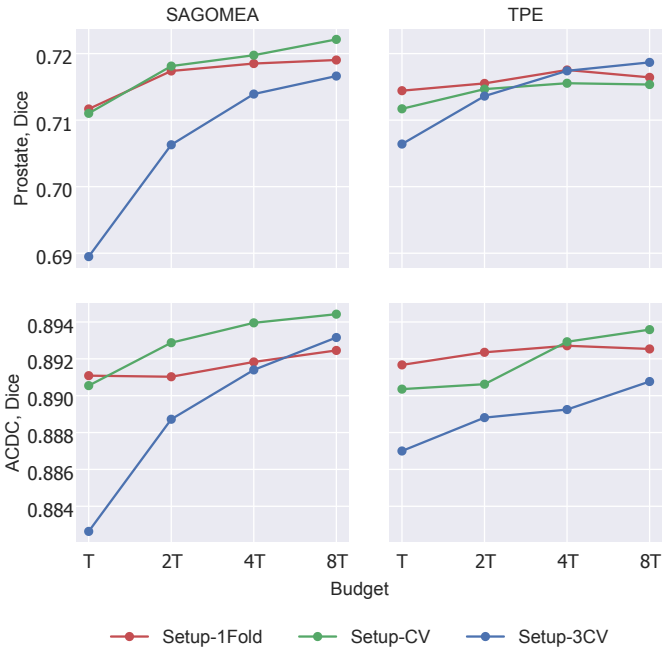


Figure 6.5.2. Main experimental results. The graphs show average (over 5 runs, 5 best architectures per run) architectures quality (obtained in the independent evaluation) under different computational budgets and different performance evaluation Setups.

Secondly, we study the differences in quality of the found architectures by different search algorithms with different performance evaluation setups and computational budgets. The quality of the networks is calculated independently of the search runs as described in Section 6.4.1. The results are shown in Figure 6.5.2; Figure 6.5.5 and in tabular form in Table 6.5.2. These results suggest that with enough computational budget for a search run, it is better (on average) to use more reliable, yet computationally more expensive performance evaluation setups as the fitness function. We see that as the budget increases, in most cases, Setup-CV starts to find better architectures than Setup-1Fold. We hypothesize that with an even larger budget, the most computationally expensive Setup-3CV might level to even results than Setup-1Fold and Setup-CV as it demonstrates steady improvement with the budget increase. On the contrary, the performance of Setup-1Fold seems to improve slower or, in some cases (for instance, with TPE, on both datasets),

even decline as the budget increases. Such behavior indicates overfitting to a specific train/validation split and random seed. Similar trends are observed for all considered search algorithms, which suggests that our findings do not depend on a specific search algorithm. Note that even with a budget of 8T, search algorithms with Setup-3CV are allowed to do only 200 function evaluations, which is not a big number even for expensive optimization algorithms. From a practical perspective, the Setup-CV seems to be a good trade-off between reliability and computational cost.

Statistical tests results are provided in Table 6.5.3. Setup-CV results in better quality architectures than Setup-1Fold with statistical significance for SAGOMEA on the Prostate dataset with budget T and on the ACDC datasets with budgets 2T, 4T, and 8T.

6.5.3. EXPLAINING PERFORMANCE DIFFERENCES

In order to better understand why Setup-1Fold is underperforming, we do the following experiment. We take two networks, one is among the best found networks for the Prostate dataset, and the second also performs reasonably well, but worse than the first one by ≈ 0.01 (in the independent evaluation). We do 30 different evaluations of these two networks (different seeds and cross-validation splits) and calculate in how many cases the first net would be preferred to the second one when different evaluation setups are used. These results are shown in Figure 6.5.3. When Setup-1Fold is used, the first net correctly shows better performance in only 17/30 cases (57%). This ratio goes up to 70% (21/30 cases) when Setup-CV is used. As expected, with Setup-3CV this ratio is even higher, namely to 29/30 cases or 97%. While there are subtle differences in some of the scores of Setup-3CV and the independent evaluation, their average values are reasonably close. These results show that using Setup-1Fold for fitness function calculation causes many situations when a suboptimal net is selected during the search, and, therefore, such an approach underperforms compared to more reliable Setup-CV and Setup-3CV (with enough computational budget). Due to low noise in fitness function scores, Setup-3CV should potentially lead to finding the best architectures if enough computational budget is available.

6.5.4. COMPARISON TO ALTERNATIVE NETWORK ARCHITECTURES

Though obtaining state-of-the-art networks is not the main goal of this chapter, we compare the performance of the found architectures to well-known U-Net-like architectures in order to ensure that the used search space contains well-performing architectures. The considered alternative architectures are the automatically configurable nnUnet [2], and three U-Nets with different encoders (Resnet-18, Efficientnet-b0, Efficientnet-b7) as implemented in the SegmentationModelsPytorch library [3]. Results are shown in Table 6.5.1. The best found architectures are visualized in Figure 6.5.6. For the Prostate dataset, all our NAS configurations managed to find better networks than the alternatives. For the ACDC dataset, our Setup-CV and Setup-3CV found better networks than all alternatives except the nnUnet. The performance of nnUnet is on par with our best Setup (3CV). Note however that in contrast to our networks, nnUnet does not use downsampling in the first convolution which allows to effectively process images in higher resolution.

[2] https://github.com/MIC-DKFZ/nnUNet
[3] https://github.com/qubvel/segmentation_models.pytorch

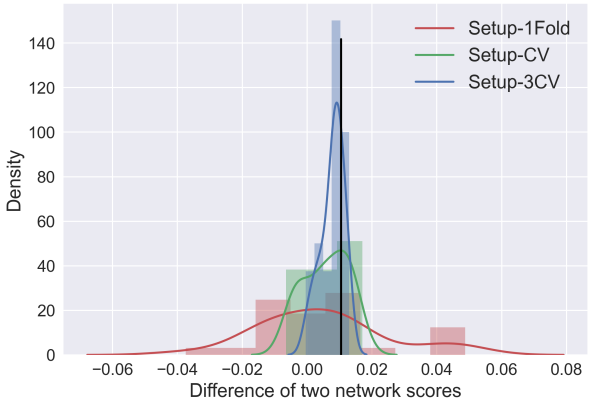


Figure 6.5.3. Histograms and fitted Kernel Density Estimation (KDE) of 30 evaluation score differences (different seeds, cross-validation splits) between two networks (score of the first net minus the score of the second net) for the three investigated Setups. The black vertical line shows the performance difference in the independent evaluation procedure. The samples to the left from zero mean that these two networks are wrongly ordered: the second one is better according to the corresponding Setup, while in the independent evaluation, the first one is better. The y-axis is normalized such that the area under the KDE curve is 1.

Architecture	Prostate, Dice	ACDC, Dice
ResNet-18-U-Net	0.690	0.893
EfficientNet-b0-U-Net	0.703	0.885
EfficientNet-b7-U-Net	0.707	0.895
nnUnet	0.699	**0.897**
Ours, Setup-1Fold (best)	0.723	0.894
Ours, Setup-CV (best)	0.723	0.896
Ours, Setup-3CV (best)	**0.726**	**0.897**

Table 6.5.1. Comparison of the best found architectures to alternative U-Net-like architectures. The numbers in the table are the final (independent) evaluation scores. Note that for each of our setups, for fairness of comparison, we report here the final evaluation score of the network found to have the best fitness according to the search procedure (among all runs of all search algorithms). The best-performing architectures according to two metrics are indicated with bold font.

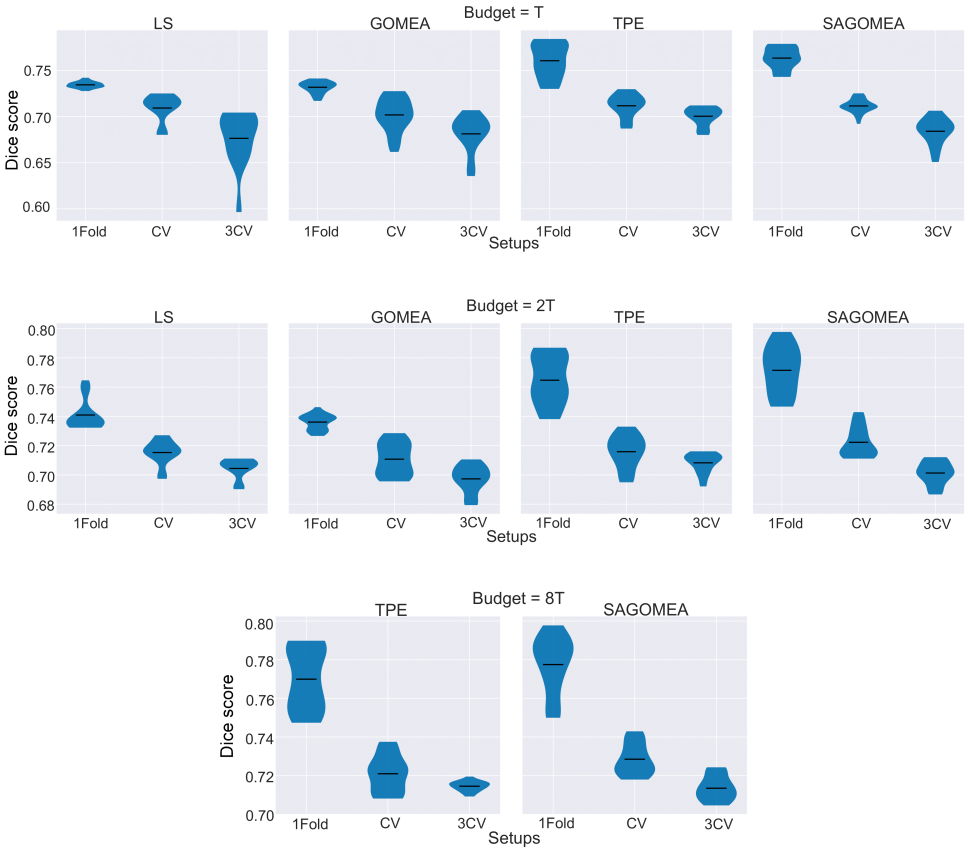


Figure 6.5.4. Comparing optimization performance of search algorithms on the Prostate dataset with different performance evaluation setups (i.e., fitness functions) with computational budgets T, 2T, and 8T. The image shows distributions (over five runs, five best architectures from each run) of the search results. Note that these results take into account only fitness function values, not the final architecture's quality.

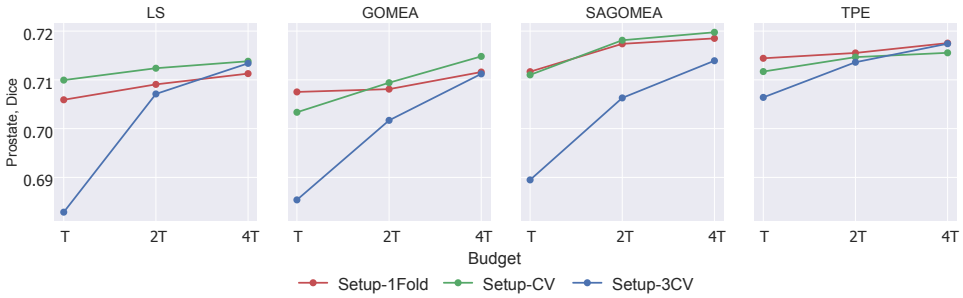


Figure 6.5.5. Experimental results for all search algorithms on the Prostate dataset. The graphs show average (over 5 runs, 5 best architectures per run) architectures quality (obtained in the independent evaluation) under different computational budgets (with up to 4T, results for budget 8T for SAGOMEA and TPE are provided in Figure 6.5.2) and different performance evaluation Setups.

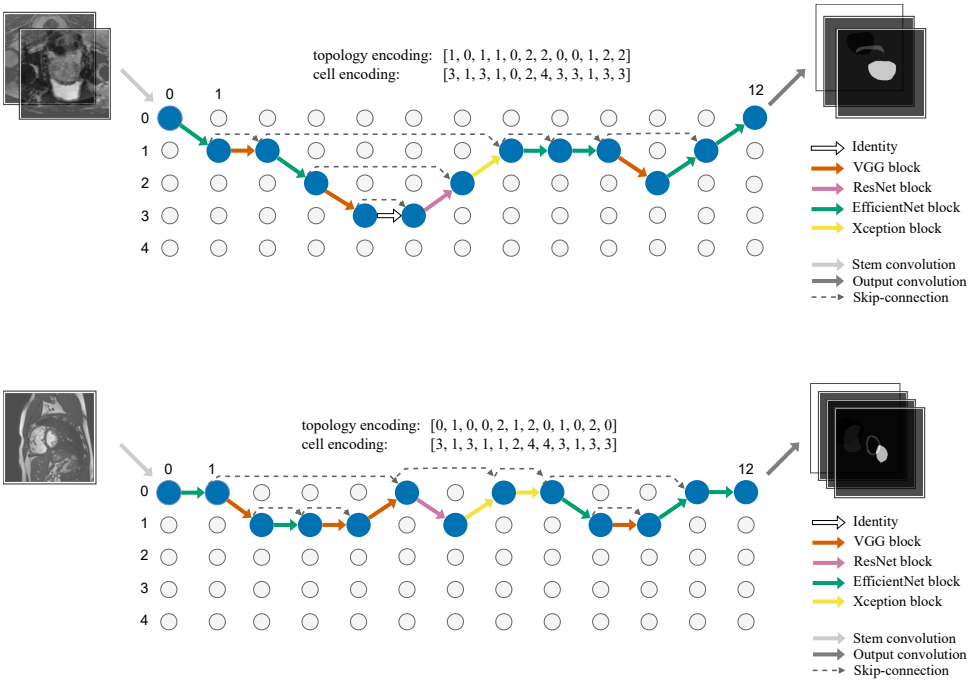


Figure 6.5.6. The best found network architecture for the Prostate (upper row) and ACDC (bottom row) datasets.

Dataset	Budget	Algorithm	Setup-1Fold	Setup-CV	Setup-3CV
Prostate	T	LS	0.734 / 0.706	0.709 / 0.710	0.676 / 0.683
		GOMEA	0.732 / 0.708	0.702 / 0.703	0.681 / 0.685
		TPE	0.761 / 0.714	0.712 / 0.712	0.708 / 0.706
		SAGOMEA	0.763 / 0.712	0.711 / 0.711	0.684 / 0.689
	2T	LS	0.741 / 0.709	0.715 / 0.712	0.704 / 0.707
		GOMEA	0.736 / 0.708	0.711 / 0.709	0.697 / 0.702
		TPE	0.765 / 0.716	0.716 / 0.715	0.708 / 0.714
		SAGOMEA	0.771 / 0.0.717	0.722 / 0.718	0.701 / 0.706
	4T	LS	0.744 / 0.711	0.719 / 0.714	0.710 / 0.713
		GOMEA	0.740 / 0.712	0.716 / 0.715	0.707 / 0.711
		TPE	0.767 / 0.718	0.718 / 0.716	0.712 / 0.711
		SAGOMEA	0.775 / 0.719	0.726 / 0.720	0.709 / 0.714
	8T	LS	0.744 / 0.711	0.719 /0.714	0.711 / 0.715
		GOMEA	0.740 / 0.712	0.716 / 0.715	0.707 / 0.711
		TPE	0.770 / 0.716	0.721 /0.715	0.714 / 0.719
		SAGOMEA	0.777 / 0.719	0.728 / 0.722	0.713 / 0.717
ACDC	T	TPE	0.895 / 0.892	0.891 / 0.890	0.888 / 0.887
		SAGOMEA	0.896 / 0.891	0.892 / 0.891	0.883 / 0.883
	2T	TPE	0.896 / 0.892	0.892 / 0.891	0.889 / 0.889
		SAGOMEA	0.896 / 0.891	0.895 / 0.893	0.889 / 0.889
	4T	TPE	0.897 / 0.893	0.894 / 0.893	0.890 / 0.889
		SAGOMEA	0.897 / 0.892	0.895 / 0.894	0.892 / 0.891
	8T	TPE	0.898 / 0.893	0.895 / 0.894	0.892 / 0.891
		SAGOMEA	0.898 / 0.892	0.896 / 0.894	0.894 / 0.893

Table 6.5.2. Experimental results for all search algorithms. All results are averaged over 5 runs and 5 best scores per run. The first number in each cell means the performance evaluation score (i.e., fitness function values) value. The second number in each cell shows means the architecture quality (obtained in the independent evaluation. Configurations (different datasets, computational budgets, and performance evaluation Setups) of all conducted experiments are included. The best-performing search algorithm in each experiment configuration and for each setup is shown in orange font color. The best-performing setup for each experiment configuration is shown in green font color.

Dataset	Budget	Algorithm	**Setup-CV** is better than **Setup-1Fold**	**Setup-3CV** is better than **Setup-1Fold**
Prostate	T	LS	0.086	1.000
		GOMEA	0.967	1.000
		TPE	0.810	1.000
		SAGOMEA	0.500	1.000
	2T	LS	0.082	0.879
		GOMEA	0.468	0.974
		TPE	0.702	0.856
		SAGOMEA	0.245	1.000
	4T	LS	0.198	0.584
		GOMEA	0.014	0.554
		TPE	0.926	0.553
		SAGOMEA	0.221	0.914
	8T	LS	0.198	0.221
		GOMEA	0.014	0.553
		TPE	0.644	0.026
		SAGOMEA	0.001	0.922
ACDC	T	TPE	0.999	1.000
		SAGOMEA	0.933	1.000
	2T	TPE	0.998	1.000
		SAGOMEA	0.000	1.000
	4T	TPE	0.458	0.999
		SAGOMEA	0.000	0.862
	8T	TPE	0.063	0.996
		SAGOMEA	0.000	0.169

Table 6.5.3. P-values of Wilcoxon tests which test hypotheses that one Setup finds networks with better quality (after independent evaluation) than another one. Statistically significant results at $\alpha = 0.05$ with Bonferroni correction ($m = 8$, for two used datasets and four search algorithms) are shown in blue font color. If a cell is empty, the corresponding experiment was not conducted. Values are rounded to three decimals.

Dataset	Budget	Setup	**TPE** is better than **LS**	**TPE** is better than **GOMEA**	**SAGOMEA** is better than **LS**	**SAGOMEA** is better than **GOMEA**	**SAGOMEA** is better than **TPE**
Prostate	**T**	1Fold	0.000	0.000	0.000	0.000	0.067
		CV	0.045	0.000	0.000	0.000	0.000
		3CV	0.000	0.000	0.183	0.674	1.000
	2T	1Fold	0.000	0.000	0.000	0.000	0.000
		CV	0.298	0.000	0.000	0.000	0.000
		3CV	0.000	0.000	0.862	0.003	0.998
	4T	1Fold	0.000	0.000	0.000	0.000	0.000
		CV	0.720	0.000	0.000	0.000	0.000
		3CV	0.004	0.000	0.874	0.205	0.949
	8T	1Fold	-	-	-	-	0.000
		CV	-	-	-	-	0.000
		3CV	-	-	-	-	0.831
ACDC	**T**	1Fold	-	-	-	-	0.057
		CV	-	-	-	-	0.001
		3CV	-	-	-	-	1.000
	2T	1Fold	-	-	-	-	0.604
		CV	-	-	-	-	0.000
		3CV	-	-	-	-	0.427
	4T	1Fold	-	-	-	-	0.817
		CV	-	-	-	-	0.001
		3CV	-	-	-	-	0.001
	8T	1Fold	-	-	-	-	0.074
		CV	-	-	-	-	0.000
		3CV	-	-	-	-	0.000

Table 6.5.4. P-values of Wilcoxon tests which test hypotheses that one search algorithm finds better solutions (by fitness values only, not in the independent evaluation) than another one. Statistically significant results $\alpha = 0.05$ with Bonferroni correction ($m = 6$, for two used datasets and three Setups) are shown in blue font color. If a cell is empty, the corresponding experiment was not conducted. Values are rounded to three decimals.

6.6. DISCUSSION

In this chapter, we focused on NAS for medical image segmentation. Due to computational cost reasons, we used a 2D segmentation paradigm and quite compact architectures. However, it was shown [15], that using a 3D segmentation approach (i.e., train on 3D volumetric patches instead of 2D patches), might be beneficial for performance. Moreover, increasing the resolution of the patches, removing the image downsampling in the stem convolution, and training for more epochs might potentially substantially increase the performance of the found networks. The computational cost of our experiments and the available computing capacity did not allow us to make such modifications, but we argue that the conducted experiments are well-suited for this study.

We used a natural, yet not necessarily the most efficient type of NAS: to evaluate the performance of each architecture, we trained it from scratch. However, there exist approaches aimed at reducing the computational costs of NAS. Two main classes of such approaches are learning curve modeling (predicting architecture performance from partial training) and supernetwork-based NAS (training one large network which has all networks in the search space as its subnetworks). For practical usage of NAS, using such methods can be beneficial as they might substantially reduce computational costs. In this chapter, we did not focus on such approaches. Furthermore, we observe that even without partial training (for a fewer number of epochs) network performance scores are quite noisy, and using partial training can only aggravate this problem. However, we believe that further studying noise in network performance evaluation with more advanced NAS techniques, including approaches based on supernetworks, is an interesting topic for further research.

6.7. CONCLUSION

In this chapter, we address the problem of stochasticity in the architecture performance evaluation score in Neural Architecture Search (NAS). We focused on NAS for medical image segmentation. To reduce the stochasticity of the network performance evaluation procedure, instead of a simple performance evaluation (training a network with one random seed and evaluating it on a fixed validation set), we proposed to use cross-validation with different random seeds for each fold or even three times repeated cross-validation with different data splits. We conduct experiments on two publicly available segmentation datasets. In our experiments, we allocated an equal computational budget to the three considered performance evaluation setups and studied differences in the performance of found networks when calculated independently of the conducted search runs. Results showed that more reliable performance evaluation setups lead to finding better-performing architectures if enough computational budget is provided. We believe the obtained conclusions can be valuable for both practical applications of NAS and the development of new NAS approaches.

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO). We thank SURF (www.surf.nl) for the support in using the Supercomputer Snellius.

REFERENCES

[1] Y. Ci et al. “Evolving search space for neural architecture search”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2021, pp. 6659–6669.

[2] D. So, Q. Le, and C. Liang. “The evolved transformer”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2019, pp. 5877–5886.

[3] Q. Yu et al. “C2FNAS: coarse-to-fine neural architecture search for 3D medical image segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4126–4135.

[4] H. Shi et al. “Bridging the gap between sample-based and one-shot neural architecture search with BONAS”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1808–1819.

[5] C. White, S. Nolen, and Y. Savani. “Local search is state of the art for NAS benchmarks”. In: *arXiv preprint arXiv:2005.02960* (2020).

[6] C. Liu et al. “Auto-DeepLab: hierarchical neural architecture search for semantic image segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 82–92.

[7] X. Yan, W. Jiang, Y. Shi, and C. Zhuo. “MS-NAS: multi-scale neural architecture search for medical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2020, pp. 388–397.

[8] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu. “GraphNAS: graph neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1904.09981* (2019).

[9] T. Den Ottelander, A. Dushatskiy, M. Virgolin, and P. A. N. Bosman. “Local search is a remarkably strong baseline for neural architecture search”. In: *Proceedings of the Evolutionary Multi-Criterion Optimization: 11th International Conference*. Springer. 2021, pp. 465–479.

[10] A. Yang, P. M. Esperança, and F. M. Carlucci. “NAS evaluation is frustratingly hard”. In: *arXiv preprint arXiv:1912.12522* (2019).

[11] C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter. “How powerful are performance predictors in neural architecture search?” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28454–28469.

[12] S. Nikolov et al. “Clinically applicable segmentation of head and neck anatomy for radiotherapy: deep learning algorithm development and validation study”. In: *Journal of Medical Internet Research* 23.7 (2021), e26151.

[13] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[14] Y. Weng, T. Zhou, Y. Li, and X. Qiu. “NAS-UNet: neural architecture search for medical image segmentation”. In: *IEEE Access* 7 (2019), pp. 44247–44257.

[15] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. In: *Nature methods* 18.2 (2021), pp. 203–211.

[16] M. Antonelli et al. “The medical segmentation decathlon”. In: *Nature communications* 13.1 (2022), p. 4128.

[17] C. Ying et al. “NAS-Bench-101: towards reproducible neural architecture search”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2019, pp. 7105–7114.

[18] X. Dong and Y. Yang. “NAS-Bench-201: extending the scope of reproducible neural architecture search”. In: *arXiv preprint arXiv:2001.00326* (2020).

[19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4780–4789.

[20] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. “Hyperopt: a Python library for model selection and hyperparameter optimization”. In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[21] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *Proceedings of the 3th International Conference on Learning Representations*. 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[23] R. Wightman, H. Touvron, and H. Jégou. “ResNet strikes back: an improved training procedure in timm”. In: *arXiv preprint arXiv:2110.00476* (2021).

[24] F. Chollet. “Xception: deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1251–1258.

[25] M. Tan and Q. Le. “EfficientNet: rethinking model scaling for convolutional neural networks”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114.

[26] O. Bernard et al. “Deep learning techniques for automatic MRI cardiac multi-structures segmentation and diagnosis: is the problem solved?” In: *IEEE Transactions on Medical Imaging* 37.11 (2018), pp. 2514–2525.

7

OBSERVER VARIATION-AWARE MEDICAL IMAGE SEGMENTATION BY COMBINING DEEP LEARNING AND SURROGATE-ASSISTED GENETIC ALGORITHMS

There has recently been great progress in medical image segmentation with deep learning algorithms. In most works, observer variation is acknowledged to be a problem as it makes training data heterogeneous, but so far no attempts have been made to explicitly capture this variation. Here, we propose an approach capable of mimicking different styles of segmentation, which potentially can improve the quality and clinical acceptance of automatic segmentation methods. Instead of training one neural network on all available data, we propose to train several neural networks on subgroups of data belonging to different segmentation variations separately. Because a priori it may be unclear what styles of segmentation exist in the data and because different styles do not necessarily map one-on-one to different observers, the subgroups should be automatically determined. We do this by searching for the best data partition with a genetic algorithm. Thus, each network can learn a specific style of segmentation from grouped training data. We provide proof of principle results for open-source prostate segmentation MRI data with simulated observer variations. Our approach provides an improvement of up to 23% (depending on simulated variations) in Dice and surface Dice values compared to one network trained on all data.

The contents of this chapter are based on the following publication: **A. Dushatskiy**, A. M. Mendrik, P. A. N. Bosman, and T. Alderliesten. "Observer variation-aware medical image segmentation by combining deep learning and surrogate-assisted genetic algorithms". In: *Medical Imaging 2020: Image Processing*. Vol. 11313. SPIE. 2020, pp. 296–306.

7.1. INTRODUCTION

7.1.1. BACKGROUND

Applying deep learning algorithms to automatically segment medical images is widely studied due to the significant advances and great success of deep learning algorithms in other fields. Fundamental improvement in segmentation performance was achieved by introducing encoder-decoder models, starting with the U-Net architecture [1]. The main idea of neural networks of this type is to use an encoder to obtain the compressed image features and then to upsample the compressed representation by a decoder. In all such architectures, the encoder contains multiple convolutional layers, while the decoder consists of either transposed convolutional or bilinear upsampling layers. These networks can be used for both binary and multi-class segmentation problems. The architecture of the encoder-decoder model has many modification possibilities, varying in number and types of layers [2, 3, 4]. Deep learning algorithms are widely applied to MRI and CT segmentation. The most commonly considered regions are brain, lungs, and lower abdomen, though there are works on upper abdomen and heart segmentation as well [5]. Despite good results of deep learning algorithms in terms of typical metrics, there is an issue of clinical acceptance of automatic segmentations to be tackled. For instance, it was shown, that a convolutional neural network can provide a high-quality automatic delineation of the prostate, rectum, and bladder with Dice scores of 0.87, 0.89, and 0.95 respectively, but for clinical usage 80% of these automatically made segmentations required manual correction [6].

The common approach is to train one network on all available data without acknowledging the heterogeneous nature of medical data caused by inter- and intra-observer variation [7]. One of the recently introduced approaches aware of observer variation is the probabilistic U-Net [8]. However, the probabilistic U-Net does not aim at capturing observer variations. The main idea of the probabilistic U-Net is to use latent variables, which parameterize the space of the produced predictions. The goal of this model is to produce a set of segmentation predictions, covering the space of probable segmentation masks as much as possible. However, such an approach is different from producing several variants of segmentations, each corresponding to a particular segmentation style. Results on lung nodule segmentation show that the probabilistic U-Net produces a variety of possible masks, different in size, but it is virtually impossible to match the predictions with segmentation styles of particular observers [8]. Thus, this approach does not solve the problem of automatically producing several segmentation masks corresponding to different segmentation styles.

7.1.2. THE PROPOSED APPROACH

This chapter is focused on the task of automatic segmentation of medical images with deep learning algorithms while capturing observer variation. Though our approach can be extended to other sources of variation in data, e.g., different scanning devices, we focus on observer variation and, particularly, consider the following scenario:

1. In a dataset, there are multiple scans, one scan per patient.
2. Each scan is segmented by one observer, the name of the observer is unknown.

3. The observers may have different ways of segmenting organs (different styles of segmentation) as well as slight variations within one style of segmentation (i.e., intra-observer variation).

We propose, instead of training one neural network on all data, to separately train several neural networks on different subsets of data corresponding to different segmentation styles. These subsets are obtained by an optimization procedure. Such an approach has two major merits: (1) We improve the segmentation quality as each network is trained on (more) homogeneous data, and therefore it can more accurately learn each segmentation style rather than learning one average style. (2) Multiple variants of segmentation can be presented to a doctor, and it is more likely that they agree with one of the variants than with an average segmentation. These two points may well contribute to the ultimate goal of our work - increased acceptance of automatic segmentation in clinical practice.

7.2. METHOD

7.2.1. ALGORITHM OUTLINE

We assume that there are multiple segmentation styles. Thus, we need to partition N scans in disjoint subgroups, representing different styles of segmentation. We tackle this partitioning task by solving multiple optimization problems (details of the optimization procedure are provided in Section 7.2.3). To simplify the partitioning task, we propose to solve it hierarchically: at each step divide the scans into two subgroups, and recursively apply the partitioning algorithm. The partitioning algorithm steps are the following:

1. Make a partition of scans in two subgroups by solving an optimization problem.
2. Validate the results: calculate scores of holdout samples in leave-one-out cross-validation on the initial data (a mixture of segmentation styles), and on both found subgroups. If training on the subgroups provides a better score compared to the training on the mixture of data, step 1 is repeated for both subgroups. Otherwise, the partitioning algorithm is stopped.

The scheme of the proposed data partitioning algorithm is provided in Figure 7.2.1a. The algorithm can be terminated when the obtained number of subgroups reaches the assumed number of segmentation styles in a dataset.

7.2.2. SEGMENTATION QUALITY EVALUATION

In the literature on automatic segmentation methods, the Dice-Sørensen Coefficient (DSC) is a common metric for evaluation: $DSC = \frac{2|G \cap P|}{|G|+|P|}$, where G is the reference segmentation mask, P is the predicted mask [9]. However, it has drawbacks for automatic segmentation evaluation in clinical practice since (1) DSC values depend on organ size, (2) DSC values do not say anything about the contour part that requires manual correction. Thus, we also consider the $2D$ version of the surface Dice-Sørensen Coefficient [10] (SDSC), a recently proposed metric aiming at alleviating shortcomings of the DSC.

The SDSC indicates the percentage of the contour that deviates from the reference by more than τ mm (the margin width τ is a chosen constant). More formally, $SDSC = \frac{|S_G \cap B_P^\tau| + |S_P \cap B_G^\tau|}{|S_G| + |S_P|}$, where $S_G = \partial G$ and $S_P = \partial P$ are borders of corresponding masks (in the $3D$ case, borders of surfaces), B_G^τ, B_P^τ are borders of corresponding masks (surfaces) at a given tolerance τ, $B_G^\tau = \{x \in \Re^2 | \exists \sigma \in S_G, \|x - \xi(\sigma)\| \leq \tau\}$, $B_P^\tau = \{x \in \Re^2 | \exists \sigma \in S_P, \|x -$

$\xi(\sigma)\| \leq \tau\}, \xi : S \mapsto \Re^2$. We use a fixed value of threshold $\tau = 0.5mm$ (making it less than voxel size) as it allows to detect small differences between contours. For the optimization procedures, we use the SDSC only, in the final evaluation of the obtained partition we calculate both DSC and SDSC cross-validation scores to demonstrate that there is an improvement in terms of the DSC as well.

7.2.3. OPTIMIZATION PROCEDURE

We formulate the problem of partitioning a set of scans P ($|P|$=N) in two disjoint subgroups P_1 and P_2, $P_1 \cup P_2 = P$, $P_1 \cap P_2 = \emptyset$ as an optimization problem. The search space (space of solutions) is the space of binary vectors of length N. Binary vectors of size N determine the partitioning: a 0 in position i determines that the $i-$th scan belongs to P_1, a 1 determines that the $i-$th scan belongs to P_2.

The ultimate goal is to maximize the cross-validation scores of two networks trained on a data partition determined by a binary vector. However, the leave-one-out cross-validation approach has a major drawback: it is very computationally expensive (it requires N model trainings) and not scalable since the number of required model trainings is equal to the number of scans. Cross-validation with a fixed number of folds k is less computationally expensive (requires $2k$ model trainings), but our preliminary experiments showed that it is not a reliable method to evaluate a partition as the scores vary a lot across different cross-validation splits. Instead of a direct evaluation of a partition (i.e., a solution of the formulated optimization problem), we propose to evaluate partitions with a proxy function. The calculation of the straightforward objective $\boldsymbol{F}$ (normalized cross-validation score) is done as follows:

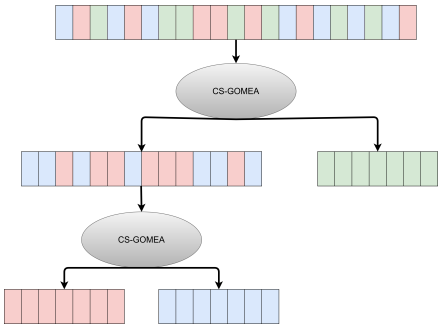
1. [Preliminary step]
   Perform leave-one-out cross-validation on P (contains a mixture of styles). For each scan, calculate its hold-out SDSC M_i.
2. [$\boldsymbol{F}$ calculation]
   (a) Perform leave-one-cross validation on P_1; calculate validation SDSC scores S_i; obtain relative scores $R_i = \frac{S_i}{M_i}$
   (b) Perform leave-one-cross validation on P_2; calculate validation SDSC scores S_i; obtain relative scores $R_i = \frac{S_i}{M_i}$
   (c) The direct function value is an average over collected relative scores R_i (of all scans): $F = \frac{1}{N}\sum_{i=1}^{N} R_i$

The calculation of the proposed proxy objective $\boldsymbol{G}$ is done as follows:

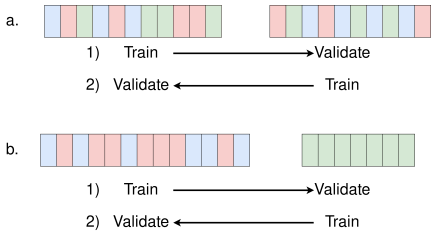
1. [Preliminary step]
   Perform leave-one-out cross-validation on P (contains a mixture of styles). For each scan, calculate its hold-out SDSC M_i.
2. [$\boldsymbol{G}$ calculation]
   (a) Train a neural network on P_1; calculate validation SDSC scores $S_i, i \in P_2$; obtain relative scores $R_i = \frac{S_i}{M_i}$
   (b) Train a neural network on P_2; calculate validation SDSC scores $S_i, i \in P_1$; obtain relative scores $R_i = \frac{S_i}{M_i}$
   (c) The proxy function value is an average over collected relative scores R_i (of all scans): $G = \frac{1}{N}\sum_{i=1}^{N} R_i$

The proxy function ***G*** is the objective, which is minimized by a binary optimization algorithm. The scheme of this function calculation is shown in Figure 7.2.1b. The intuition behind the proposed proxy function is the following: the more different the segmentation styles of two subgroups, the smaller the scores obtained in steps 2.1 and 2.2 of ***G*** calculation. Thus, we can minimize this proxy function instead of directly maximizing the cross-validation scores. Our experiments show, that it is a valid proxy function, i.e., its values correlate well with the target function. Experiments and results demonstrating this are described in Section 7.4. Moreover, the number of neural network trainings is fixed (only two trainings are required), independent of the number of scans.

For optimization, we use a surrogate-assisted genetic algorithm, namely, the convolutional neural network surrogate-assisted GOMEA (CS-GOMEA) introduced in Chapter 4. This algorithm was chosen as it is capable of finding good solutions within a limited number of function evaluations, and it outperforms competitors on a set of benchmark functions. In the current application, the algorithm requires ≈ 250 evaluations to converge, which is ≈ 5 times less than a state-of-the-art non-surrogate-assisted genetic algorithm (GOMEA).



(a) The recursive scheme of the partitioning algorithm. Different colors denote different segmentation styles. The example demonstrates the perfect partition found by a binary optimization algorithm (we use the surrogate-assisted genetic algorithm CS-GOMEA introduced in Chapter 4), resulting in three subgroups of scans delineated in three different segmentation styles.



(b) Two examples of data partitioning evaluated by the optimized objective function. Different colors denote different segmentation styles. The partitioning evaluation contains two model trainings. First, the scans from the first subgroup are used for training, and for the scans from the second subgroup, the validation scores are calculated (step 1). Then, the scans from the second subgroup are used for training and the scans from the first subgroup are used as validation set (step 2). The score of the objective function is supposed to be better in the example **b**, as the subgroups are more different from each other by contained segmentation styles.

Figure 7.2.1. The key components of the proposed data partitioning algorithm.

7.2.4. NEURAL NETWORK ARCHITECTURE AND TRAINING

We use the architecture of the Context Encoder Network (CE-Net) [11], which was recently introduced and demonstrated state-of-the-art results on medical image segmentation benchmarks. This architecture falls under the category of encoder-decoder models for segmentation tasks, introduced with the U-Net architecture. The encoder of this network is part of the ResNet-34 architecture, pretrained on the ImageNet dataset. To use the encoder with three input channels without any modifications on grayscale images, we simply copy the input image three times and pass the copies to the encoder.

For neural network training with back-propagation, we use the Adam optimizer [12]. The learning rate is 10^{-4} (lower than the default 10^{-3} to make the training more stable). The performance on the validation set is checked every training epoch, and the learning rate is decreased by a factor of 10 if the validation metric has not improved for 5 consecutive epochs. If the learning rate becomes smaller than 10^{-5}, the training is stopped. During training, we use random data augmentations to reduce overfitting: scale (by a factor from 0.7 to 1.3), shift (from -40 to 40 px in each direction), rotation (from -10 to 10 degrees), contrast and brightness adjustments (by a factor from 0.5 to 1.5). These augmentations are chosen according to possible scan appearance variations in the dataset. The batch size is 128. The loss function is the commonly used soft dice loss.

7.3. EXPERIMENTS

7.3.1. DATA

We use open data from the Medical Segmentation Decathlon [1] [13] containing MRI scans of the prostate of 32 patients with corresponding segmentation masks. Each segmentation mask represents pixel-wise labels of one of three classes: background, prostate central zone (CZ), and prostate peripheral zone (PZ). We do not distinguish between the prostate CZ and the prostate PZ and consider a segmentation task with 2 classes: prostate and background. To make a proper transformation of the initial $3D$ data to $2D$ slices, we perform a standard procedure of voxel spacing normalization. The resulting voxel spacing of our scans is (0.6mm, 0.6mm, 2mm). After normalization, each scan is zoomed in by a factor of two to decrease the number of background pixels and rescaled to (128px, 128px) size.

7.3.2. SIMULATED VARIATIONS

We simulated both global and local variations in segmentation styles by applying a transformation to the reference prostate segmentation mask. Global variations include erosion, dilation, and shift operations. Local ones are under/over-segmentation of the top/bottom (base/apex) parts of the prostate. The chosen variations transform the reference contours in prostate zones that have the largest inter-observer variation[14]: base and apex. Table 7.3.1 summarizes the applied variations. Each transformation has a numerical parameter (e.g., shift by t pixels), which is sampled from a normal distribution for each $2D$ slice. Different transformations simulate different segmentation styles, while the parameters of transformations determine the extent of variations within a style. For instance, we simulate that some scans are segmented by an observer who always over-

[1] http://medicaldecathlon.com

segments the upper part of the prostate. Other scans are segmented by an observer who under-segments this part (two different styles of segmenting). The magnitude (in pixels) of over- and under-segmentation is sampled from a Gaussian distribution, representing the variation within a style. In Figure 7.3.1 examples of applied variations are illustrated. Transformations are done once, stored on disk, and then all experiments are conducted using the obtained data.

	Global variations	**Local variations**
Prostate mask size is changed	erosion dilation	over-segmentation of top part under-segmentation of top part over-segmentation of bottom part under-segmentation of bottom part
Prostate mask size is unchanged	shift	-

Table 7.3.1. Overview of applied artificial variations.

To conduct an experiment with two types of variations, we divide all scans into two subgroups (each containing 16 scans). The first subgroup contains scans with an applied variation of the first type only, the second subgroup contains scans with the variation of the second type. After forming the subgroups with different variations, all scans are gathered together and form a mixture of segmentation styles. Then, we put aside a part of all scans (12 scans) for network pretraining. Our preliminary experiments have shown that pretraining is essential, as it makes the training procedure more stable and helps in solving the optimization task. The remaining scans (20) are used in the optimization algorithm. The scheme of experiments with two segmentation styles is demonstrated in Figure 7.3.2.

To demonstrate that our approach is not limited to only two styles, we conduct an experiment with three variations in the data as well. In the case of three segmentation styles, we use 11 scans for pretraining and solve a partitioning problem for 21 scans. In this case, there are three subgroups (each has seven scans), each containing scans with a particular type of variation. We make experiments with eight different combinations of two variations and one combination of three variations. The tested combinations of variations are listed in Table 7.4.1.

7.3.3. OBJECTIVE FUNCTIONS ANALYSIS

To investigate how the proxy objective function $\boldsymbol{G}$ described in 7.2.3 is connected with the cross-validation scores function $\boldsymbol{F}$ (direct objective), we do the following experiment. We select two mixtures of scans: one with large (erosion/dilation $\sim \mathcal{N}(10,4)$) and one with small (top over-/under-segmentation $\sim \mathcal{N}(5,1)$) artificial variations. We take the optimal solution S that represents the perfect partition in two subgroups: each subgroup contains scans with variations of one type only. Then, we generate solutions having Hamming distance to S from 1 to 10, generating five random solutions for each value of

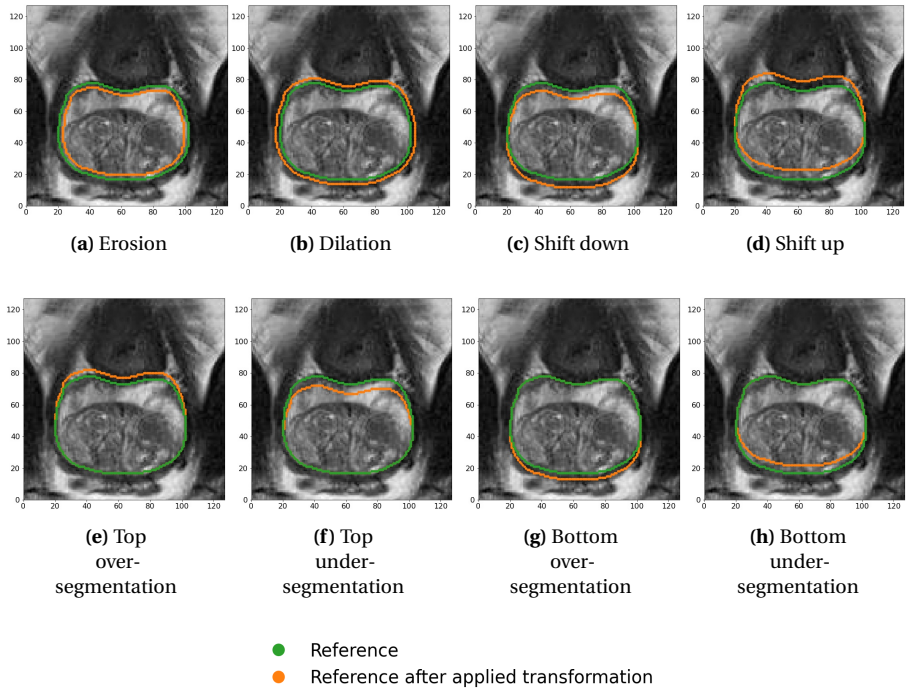


Figure 7.3.1. Variations applied to the reference masks demonstrated on the same slice. The original reference contour is plotted in orange, and the contour after applying the transformation is illustrated in green. The global variations are shown in the first row and the local ones in the second row. The magnitude of all transformations in these examples was sampled from the Gaussian distribution $\mathcal{N}(5,1)$.

the Hamming distance. For S and these generated solutions, we calculate values of $\boldsymbol{F}$ and $\boldsymbol{G}$.

7.4. RESULTS

The scatter plots demonstrating how values of the proposed proxy objective function $\boldsymbol{G}$ are related to values of the direct objective $\boldsymbol{F}$ (the experiment is described in Section 7.3.3) are shown in Figure 7.4.1. The correlation coefficient ρ shows a strong dependency between the two functions. Importantly, the point corresponding to S is the maximum of $\boldsymbol{F}$ and the minimum of $\boldsymbol{G}$ (among the sampled points). These results support the proposed idea of minimizing $\boldsymbol{G}$ instead of maximizing $\boldsymbol{F}$.

The quantitative results of the proposed segmentation approach are provided in Table 7.4.1. Our algorithm in all cases produces a partition of scans in subgroups, which indeed have different variations. In most cases, it produces such a grouping, that all scans in a subgroup have the same segmentation style. Consequently, a better performance (in terms of both DSC and SDSC) is achieved when training separately on the two found subgroups of scans compared to training on all of them in all conducted experiments.

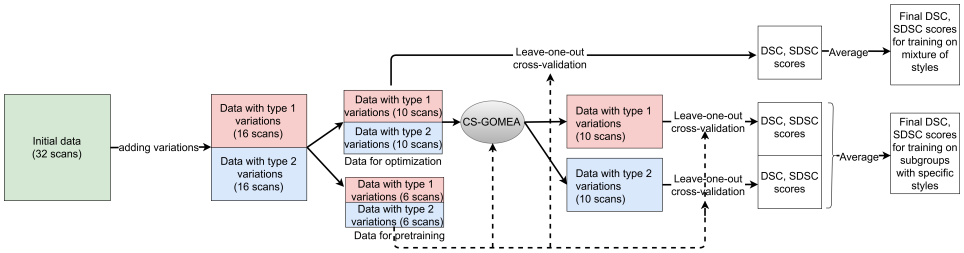


Figure 7.3.2. Experiment scheme for the case of two simulated variations. First, two styles of segmentation are simulated for two subgroups of scans. Second, a part of the scans is put aside for pretraining. All networks are pretrained on these scans (indicated by dashed arrows). Then, CS-GOMEA is used to find an optimal partition of the remaining scans. As a result, it produces two subgroups of scans, ideally each having scans with one segmentation style only. To check the improvement provided by the obtained partition, leave-one-out cross-validation is performed on both subgroups, and final DSC and SDSC scores are averaged over all scans. They are compared with the DSC and SDSC scores of the leave-one-out cross-validation of training on the mixture of styles. The results of the comparison are provided in Table 7.4.1.

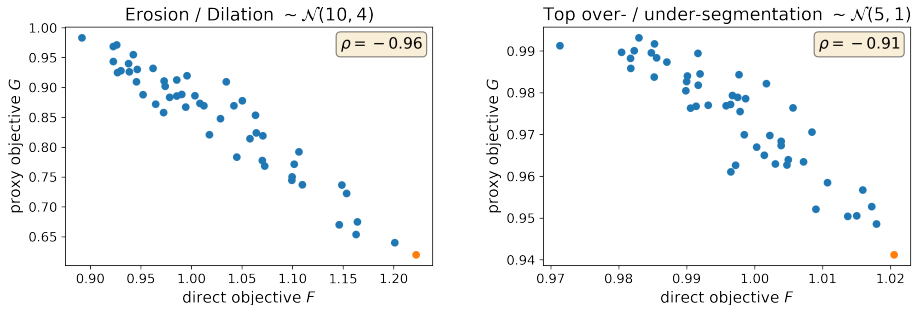


Figure 7.4.1. Comparison of values of the direct objective function F and the proposed proxy objective function G for two examples of mixtures of large (left plot) and small (right plot) artificial variations. The correlation coefficient is denoted as ρ. The point corresponding to the partition of samples in two subgroups having two different segmentation styles is visualized in orange.

This improvement is larger for the experiments with variations of a larger magnitude. Also, the improvement is larger for the experiments with global variations (erosion/dilation and global shifts), than for local ones, as the global variations transform the reference masks by a larger number of pixels. In the experiment with three different variations, the number of misclassified patients is larger and, consequently, the improvement is smaller compared to the experiment with only two of these variations (over-under-segmentation of the top part $\sim \mathcal{N}(10, 4)$).

The qualitative results are demonstrated in Figure 7.4.2. The images show that the contour produced by a network trained on a specific style of segmentation is closer to the reference contour than the one produced by a network trained on a mixture of styles. Moreover, it demonstrates that a neural network can learn large variations, not necessarily

Variations			Mixture		Specific styles		Improvement (%)	
Operation	**Size(px) ~**	**Mis.**	**DSC**	**SDSC**	**DSC**	**SDSC**	**DSC**	**SDSC**
Erosion/dilation	$\mathcal{N}(10,4)$	0	0.65	0.67	0.79	0.81	**22.91**	**20.49**
Erosion/dilation	$\mathcal{N}(5,1)$	0	0.78	0.79	0.83	0.84	**6.36**	**6.64**
Up/down shift	$\mathcal{N}(10,4)$	0	0.70	0.72	0.82	0.84	**17.43**	**17.69**
Up/down shift	$\mathcal{N}(5,1)$	0	0.80	0.81	0.84	0.86	**6.06**	**6.26**
Bottom over-/under -segmentation	$\mathcal{N}(10,4)$	0	0.76	0.78	0.83	0.85	**8.2**	**8.16**
Bottom over-/under -segmentation	$\mathcal{N}(5,1)$	1	0.82	0.83	0.83	0.85	**2.16**	**1.90**
Top over-/under -segmentation	$\mathcal{N}(10,4)$	0	0.78	0.81	0.83	0.84	**5.83**	**4.16**
Top over-/under-segmentation	$\mathcal{N}(5,1)$	3	0.83	0.85	0.84	0.86	**1.27**	**1.27**
Top over-/under-/bottom under -segmentation	$\mathcal{N}(10,4)$	3	0.76	0.78	0.79	0.81	**3.57**	**3.60**

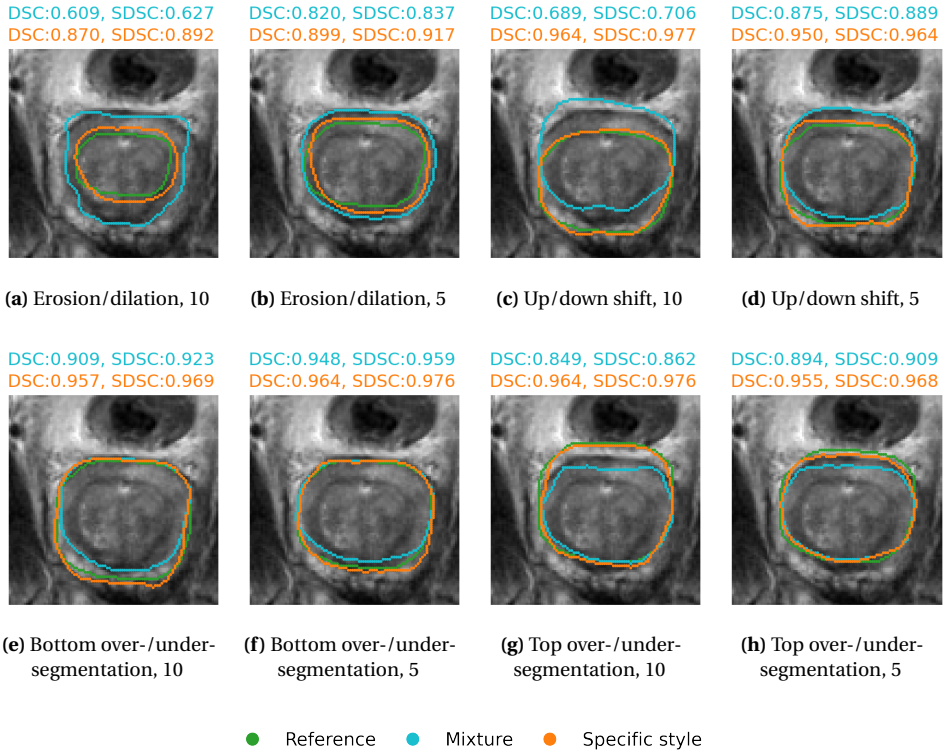
Table 7.4.1. Main results. In the mis. (misclassified) column the numbers of scans which were put in the wrong subgroup are given, based on how segmentation variations were created. DSC and SDSC scores are rounded to two decimal places.

aligned with the organ anatomy if such a variation is consistently present in the training data. For variations of a larger magnitude, the difference between contours produced by a network trained on the mixture of segmentation styles and contours produced by networks trained on scans having a particular segmentation style is more substantial. This is demonstrated by quantitative results (in Table 7.4.1) as well: the improvement gained by training networks separately is larger for the experiments with variations of a larger magnitude.

We found, that in our experiments, CS-GOMEA requires 250 evaluations to converge: 200 of them are performed in the warm-up stage, and 50 are used for the real search process. Function evaluations in the warm-up stage are performed in parallel. One function evaluation takes ≈ 3 minutes (we use four NVIDIA RTX 2080 Ti cards for training the neural networks). The total computational time of one experiment is ≈ 5 hours.

7.5. DISCUSSION

We propose a novel approach for automatic medical image segmentation with deep learning algorithms: instead of training one universal model for all available data we propose to train separate networks for each segmentation style. Such a technique has the potential to provide better segmentation quality and better acceptance of automatic methods in clinical practice, as it produces a set of predicted contours, from which a doctor can choose the preferred one.



(a) Erosion/dilation, 10 **(b)** Erosion/dilation, 5 **(c)** Up/down shift, 10 **(d)** Up/down shift, 5

(e) Bottom over-/under-segmentation, 10 **(f)** Bottom over-/under-segmentation, 5 **(g)** Top over-/under-segmentation, 10 **(h)** Top over-/under-segmentation, 5

Figure 7.4.2. Samples of a slice (zoomed in) from experiments with different variations. The DSC and SDSC scores in the first and second rows above each image correspond to scores in the case of training on the mixture of styles and training on the subgroups of scans obtained by the optimization algorithm, respectively. The reference contour (illustrated in green) represents a simulated style of segmentation: the initial reference contour after applying a transformation.

In this chapter, we use a modern CE-Net neural network. However, our approach is not limited to a particular neural network architecture. Moreover, it is not limited to deep learning-based segmentation methods. Potentially, if a new state-of-the-art algorithm for automatic segmentation appears, it can be integrated into the proposed method as long as it uses a general machine learning paradigm of learning from data. We provide proof of principle for our approach for prostate segmentation data on MRI scans. Nevertheless, the same concept could be applied to other organs and medical imaging types as well. Moreover, our approach is not limited to a particular optimization algorithm. However, considering the time-consuming nature of function evaluations of the optimization problem (one evaluation takes ≈ 3 minutes), it is important that the used optimization algorithm is capable of finding a good solution within a limited number of function evaluations.

In our experiments, we used a dataset of moderate size: it contains 32 scans. We suppose that applying our approach to larger datasets (more than 50 scans) can provide even better results, as it will allow the detection of more subtle variations. In the case of

a small dataset, the performance of training on a larger number of subgroups results in the number of scans in these subgroups not being sufficient to accurately learn a specific segmentation style. In the experiment with three different variations, the improvement is smaller compared to the experiment with only two of these variations, and it can be explained by the lack of data in the subgroups: seven scans is perhaps not enough to accurately learn a specific segmentation style.

Our algorithm shows good results on artificially generated variations. The main assumption of the approach is that there exist different consistent segmentation styles. That is a reasonable assumption, but in case the observers are inconsistent (the variations are random) the applicability of our approach is limited and needs further study. Another assumption is that the names of observers are unknown. Such a situation is not uncommon in clinical practice. Nevertheless, even if for each scan the name of the observer, who made the segmentation, is known, our algorithm has an added value. Different observers may have similar styles of segmentation, while one observer may have different styles, depending on extra information about a patient. Thus, the task of revealing subgroups of scans segmented in different styles is still non-trivial and our algorithm can be applied to solve it.

Applying the algorithm to real clinical data is left for future research. The potential of the algorithm regarding application to other sources of variation in the training data (e.g., different scanning devices) also needs to be studied in the future.

7.6. CONCLUSIONS

We have proposed a novel approach to medical image segmentation aimed at capturing observer variation in data. Specifically, to identify different segmentation styles automatically, we proposed to automatically divide scans into subgroups (each subgroup representing a specific style of segmentation) by solving an optimization problem using the surrogate-assisted genetic algorithm CS-GOMEA. In experiments with artificial segmentation variations added to the data (representing multiple distinguishable segmentation styles), our approach was found to be able to detect the subgroups automatically and build multiple segmentation models pertaining to these styles, resulting in a substantial performance increase of auto-segmentation. Our approach is designed to find the best possible split of data to improve overall segmentation quality. Thus, it can be applied to any dataset and will likely produce such subgroups that training on them separately provides better results. Moreover, the proposed approach can be used with any segmentation method and our approach produces a grouping that leads to better results. Finally, we note, that our approach can be extended to also consider other sources of variation in data (e.g., different scanning devices).

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO).

REFERENCES

[1] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[2] A. Chaurasia and E. Culurciello. "LinkNet: exploiting encoder representations for efficient semantic segmentation". In: *2017 IEEE Visual Communications and Image Processing (VCIP)*. IEEE. 2017, pp. 1–4.

[3] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440.

[4] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. "ENet: a deep neural network architecture for real-time semantic segmentation". In: *arXiv preprint arXiv:1606.02147* (2016).

[5] M. H. Hesamian, W. Jia, X. He, and P. Kennedy. "Deep learning techniques for medical image segmentation: achievements and challenges". In: *Journal of Digital Imaging* 32.4 (Aug. 2019), pp. 582–596.

[6] A. Amjad et al. "Evaluation of a deep learning-based auto-segmentation tool for online adaptive radiation therapy". In: *International Journal of Radiation Oncology, Biology, Physics* 105.1 (2019), E785–E786.

[7] M. H. Hesamian, W. Jia, X. He, and P. Kennedy. "Deep learning techniques for medical image segmentation: achievements and challenges". In: *Journal of Digital Imaging* 32.4 (Aug. 2019), pp. 582–596.

[8] S. Kohl et al. "A probabilistic U-Net for segmentation of ambiguous images". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6965–6975.

[9] A. P. Zijdenbos, B. M. Dawant, R. A. Margolin, and A. C. Palmer. "Morphometric analysis of white matter lesions in MR images: method and validation". In: *IEEE Transactions on Medical Imaging* 13.4 (1994), pp. 716–724.

[10] S. Nikolov et al. "Clinically applicable segmentation of head and neck anatomy for radiotherapy: deep learning algorithm development and validation study". In: *Journal of Medical Internet Research* 23.7 (2021), e26151.

[11] Z. Gu et al. "CE-Net: Context Encoder Network for 2D medical image segmentation". In: *IEEE Transactions on Medical Imaging* 38.10 (Oct. 2019), pp. 2281–2292.

[12] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *Proceedings of the 3th International Conference on Learning Representations*. Ed. by Y. Bengio and Y. LeCun. 2015.

[13] A. L. Simpson et al. "A large annotated medical image dataset for the development and evaluation of segmentation algorithms". In: *arXiv preprint arXiv:1902.09063* (2019).

[14] G. M. Villeirs et al. "Interobserver delineation variation using CT versus combined CT + MRI in intensity–modulated radiotherapy for prostate cancer". In: *Strahlentherapie und Onkologie* 181.7 (July 2005), pp. 424–430.

8

DATA VARIATION-AWARE MEDICAL IMAGE SEGMENTATION

Deep learning algorithms have become the golden standard for segmentation of medical imaging data. In most works, the variability and heterogeneity of real clinical data are acknowledged to still be a problem. One way to automatically overcome this is to capture and exploit this variation explicitly. Here, we propose an approach that improves on our previous work in this area and explain how it potentially can improve clinical acceptance of automatic segmentation methods.

In contrast to a standard neural network that produces one segmentation, we propose to use a multi-path U-Net network that produces multiple segmentation variants, presumably corresponding to the variations that reside in the dataset. Different paths of the network are trained on disjoint data subsets. Because a priori it may be unclear what variations exist in the data, the subsets should be automatically determined. This is achieved by searching for the best data partitioning with an evolutionary optimization algorithm. Because each network path can become more specialized when trained on a more homogeneous data subset, better segmentation quality can be achieved. In practical usage, various automatically produced segmentations can be presented to a medical expert, from which the preferred segmentation can be selected. In experiments with a real clinical dataset of CT scans with prostate segmentations, our approach provides an improvement of several percentage points in terms of Dice and surface Dice coefficients compared to when all network paths are trained on all training data. Noticeably, the largest improvement occurs in the upper part of the prostate which is known to be most prone to inter-observer segmentation variation.

The contents of this chapter are based on the following publication: **A. Dushatskiy**, G. Lowe, P. A. N. Bosman, and T. Alderliesten. "Data variation-aware medical image segmentation". In: *Medical Imaging 2022: Image Processing*. Vol. 12032. SPIE. 2022, pp. 759–765.

8.1. INTRODUCTION

This chapter is focused on the automatic segmentation of medical images with deep learning algorithms while capturing data variation. Fundamentally, our approach can detect and tackle any existing source of variation in the data, e.g., different scanning devices or different patient groups. Specifically, we focus on observer segmentation variation and, particularly, consider the following scenario: (1) In a dataset there are multiple scans, one scan per patient. (2) Each scan is segmented by one observer, the name of the observer is normally unknown. In contrast to other works which focus on medical image segmentation taking into account the observer variation [1, 2], our goal is not to model the data uncertainty and use a probabilistic model to produce a set of possible segmentations but to explicitly capture observer variation, and produce segmentation variants corresponding to different ways of segmenting residing in the dataset. Moreover, we do not assume that for each scan segmentations by multiple observers are available. We believe such a situation is quite common in clinical practice.

We envision a clinical usage scenario as follows. For each scan that needs to be segmented (e.g., for brachytherapy treatment planning), our neural network (previously trained) produces multiple organ segmentation variants. A medical expert can choose one of those for further clinical usage. Additional manual adjustments are possible but much less likely needed since there were multiple options in-line with the original variation in the data to choose from. Thus, the goal of the neural network is to produce segmentations such that at least one of them is close to the expert's segmentation understanding for that particular scan. We propose an approach to get more accurate and more diverse segmentation variants by using different data subsets during training. The best dataset partitioning is found automatically using an efficient optimization algorithm. Such an approach has two major goals: (1) Improve the segmentation quality as each net is trained on more homogeneous data and thereby more accurately learn each possible specific segmentation variant rather than learning just one, most probably an average, segmentation. (2) Multiple variants of segmentation can be presented and it is more likely that an observer agrees with one of the variants than with an average segmentation. These two points may well contribute to the ultimate goal of our work - increased acceptance and uptake of automatic segmentations in clinical practice.

To the best of our knowledge, the only similar approach was initially proposed by us in Chapter 7. However, in our previous work substantially large simulated segmentation variations were used, and the approach was mainly suited for only two types of variations. Here, we propose an efficient algorithm capable of capturing an arbitrary number of variations (defined by the user beforehand) and applying it to segmentation variations present in real clinical data.

8.2. METHOD

8.2.1. MULTI-PATH SEGMENTATION NETWORKS

Ultimately, in our approach, each segmentation variant is supposed to be learned by a separate segmentation network. In common U-Net-like segmentation architectures [3], which consist of an encoder and a decoder, an encoder learns compressed representations of image features, while the decoder translates them to a segmentation prediction.

It is reasonable to utilize the same encoder for different segmentation variants, while connecting it to multiple decoders because (1) We can expect that the image features required to properly predict variants are common. (2) A shared encoder improves computational efficiency.

We assume that our segmentation network consists of an encoder E and multiple decoders $D_1,\ldots,D_\alpha$. The i-th segmentation variant for input image x is produced as $D_i(E(x))$. The architecture scheme is shown in Figure 8.2.1. In each training epoch, each decoder is trained using the corresponding training data subset P_i. Note that the encoder is trained using the full dataset P. This training procedure is shown in pseudocode in Algorithm 8.2.1.

8.2.2. OPTIMIZATION PROCEDURE

We formulate the problem of partitioning a set of scans P ($|P|$=N) into α disjoint subsets $P_1,P_2,\ldots,P_\alpha$: $P_1 \cup P_2 \cdots \cup P_\alpha = P, P_1 \cap P_2 \cdots \cap P_\alpha = \emptyset$ as an optimization problem. The search space (space of solutions) is the space of vectors of length N over alphabet $\{1,2,\ldots,\alpha\}$, i.e., $\{1,2,\ldots,\alpha\}^N$. These vectors determine the partitioning: a value k in position i determines that the $i-$th scan belongs to P_k. For optimization, we use a recently introduced efficient surrogate-assisted evolutionary algorithm *SA-P3-GOMEA* (introduced in Chapter 5) that was specifically designed for discrete optimization problems with expensive function evaluations. In this optimization problem, one function evaluation means a full training and evaluation of a network with a suggested dataset partitioning. As this is a computationally expensive procedure, an efficient optimization algorithm might be used to reduce the overall computational costs.

To define the function to be optimized, resulting in the best partitioning, we explain how each partitioning is evaluated. After a net is trained using Algorithm 8.2.1, we use a fixed validation set V for evaluation. For each scan from V, the trained network produces α segmentation variants. For each variant, its quality is computed using a segmentation performance metric. The variant with the best score is selected for that scan. The final validation score is averaged over all scans in V. This score is the optimized (maximized) function value. Such an evaluation procedure simulates a usage scenario, such, that for each scan a medical expert is shown multiple segmentation variants and is interested in selecting the best one. To avoid overfitting on the validation set, we use an additional test set, which is left out during the optimization and is used only for obtaining the final results.

8.2.3. SEGMENTATION QUALITY EVALUATION

We consider common segmentation performance metrics: the Dice-Sørensen Coefficient (DSC), and the Surface Dice-Sørensen Coefficient [4] (SDSC). While the DSC measure considers only global segmentation quality accuracy without focusing on the accuracy near organ borders, the SDSC indicates the ratio of the contour that deviates from the reference by no more than τ mm (the tolerance value τ is a chosen constant). An SDSC value of one means no manual adjustment is needed if the acceptable deviation from the reference is τ mm. Inside the optimization procedure, we use the average of DSC and SDSC (with $\tau = 2mm$) to combine both global and local segmentation quality metric values.

8.2.4. SEGMENTATION NEURAL NETWORK ARCHITECTURE

Our approach is not limited to a particular segmentation network architecture. However, due to the computational complexity (many trainings inside the optimization loop), the used segmentation network has to be computationally efficient. Thus, we selected two simple, yet efficient (found in preliminary experiments) U-Net architectures from [5]. The selected networks differ in the implemented encoder network. The first one is based on the ResNet18 [6], while the second one has the VGG19 [7] encoder. Preliminary experiments demonstrated that VGG19-U-Net shows better performance than ResNet18-U-Net at the cost of a longer training time. While ultimately we are interested in using the best performing network, we show results on both selected networks in order to demonstrate the independence of our approach from a particular architecture.

Algorithm 8.2.1: Training routine for the proposed segmentation approach

input: The number of demanded segmentation variants α; dataset partitioning $P_1, \ldots, P_\alpha$; architecture encoder E, decoders $D_1, \ldots, D_\alpha$, the number of training epochs *nEpochs*

```
// X denotes an image, y is a corresponding reference segmentation, ŷ is a
   predicted segmentation
```

for *epoch* $\leftarrow 1, \ldots,$ *nEpochs* **do**
  for $i \in$ `randomPermutation`$([1, \ldots, \alpha])$ **do**
    for *batch of pairs* $(X, y) \in P_i$ **do**
      $\hat{y} \leftarrow E(D_i(batch))$
      $L \leftarrow$ `loss`$(y, \hat{y})$
      update weights of E and D_i by backpropagation using L

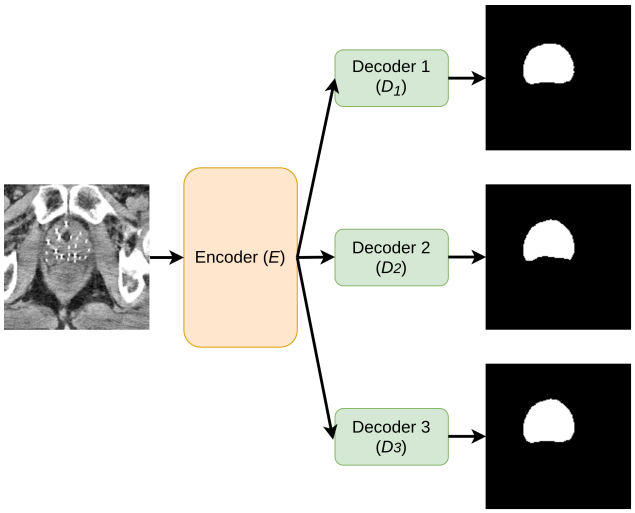


Figure 8.2.1. A schematic illustration of the used multi-path U-Net network. Multiple decoders (3 in this case) are attached to the shared encoder and produce three different segmentation predictions.

8.3. EXPERIMENTS

8.3.1. DATA

We use a real clinical dataset from Mount Vernon Cancer Centre (Northwood, UK) of 196 prostate cancer patients treated with external beam radiation treatment and brachytherapy. For each patient, we have one CT scan with the corresponding organ segmentations that were used for brachytherapy treatment planning purposes. In the preliminary preprocessing step, 13 scans with severe visual artifacts are filtered out. From the remaining 183 scans, 40 and 43 scans were left out for validation and test sets, respectively. The remaining 100 scans can be used for partitioning optimization and training. We consider a segmentation task with two classes: prostate and background. The initial $3D$ data is transformed into $2D$ slices. Each scan consists of 18 slices on average (range: 10 to 27 slices). In order to balance the number of background and prostate pixels, each slice is cropped to the 150mm × 150mm central region of the image. The pixel spacing of the processed slices is 1.17mm × 1.17mm.

8.3.2. EXPERIMENTAL SETUP

To show that our approach works with datasets of different sizes, we perform experiments with 50 and 100 patients in the training part ($N = 50, 100$). We set the number of evaluated partitionings in each optimization run in the case of $N = 50$ to 2000. This value was chosen as a tradeoff between computational time and the quality of the found partitionings. As one epoch of training with 100 scans is approximately two times longer than the epoch of training with 50 scans, we set the number of evaluated partitionings in the case of $N = 100$ to 1000. We study the results when various numbers of partitions are used: $\alpha = 2, 3, 4$. As an additional baseline, we add a standard single-path network ($\alpha = 1$). As both the neural network training and the used optimization algorithm are naturally stochastic, we repeat an experiment three times for each value of α, and each used network architecture (ResNet18-U-Net or VGG19-U-Net). All results are reported on the test set, which guarantees no overfitting (on the validation set). The reported values are averaged over the three runs.

Our experiments aim at showing that our approach results in better segmentation performance in comparison to the standard training approach, i.e., with no data partitioning (all network paths are trained on the full training dataset).

8.4. RESULTS

The main results are shown in Table 8.3.1. The proposed approach shows better performance than a corresponding multi-path network trained without dataset partitioning by approximately 1% in DSC coefficient when $N = 50$. More substantial differences of up to 5% and 1.5% can be observed in SDSC metric with $\tau = 2mm$ and, $\tau = 4mm$ respectively. Note that the smaller τ value means less tolerance to the deviations from the reference segmentation. Furthermore, our approach with α values of 2, 3, and 4 shows better performance than a standard single-path network $\alpha = 1$. In the case of $N = 100$, the improvements of our approach are smaller ($1.5 - 2.5\%$ in SDSC, $\tau = 2mm$, $0.5 - 1\%$ in DSC), though it still managed to find an improvement over the baselines in all cases. We hypothesize, that running optimization for longer might result in finding better partition-

			No partitioning			Our approach			Improvement (%)		
Network	***N***	***α***	**DSC**	**SDSC**		**DSC**	**SDSC**		**DSC**	**SDSC**	
				2mm	4mm		2mm	4mm		2mm	4mm
ResNet18-U-Net	50	1	0.863	0.670	0.917	-	-	-	-	-	-
	50	2	0.864	0.675	0.920	0.871	0.701	0.931	**0.88**	**3.92**	**1.15**
	50	3	0.863	0.672	0.918	0.872	0.703	0.934	**1.04**	**4.52**	**1.68**
	50	4	0.865	0.677	0.922	0.874	0.711	0.936	**1.14**	**5.14**	**1.58**
	100	1	0.864	0.676	0.919	-	-	-	-	-	-
	100	2	0.866	0.683	0.922	0.870	0.695	0.928	**0.55**	**1.87**	**0.71**
	100	3	0.867	0.686	0.923	0.872	0.702	0.930	**0.67**	**2.41**	**0.83**
	100	4	0.868	0.691	0.924	0.873	0.708	0.930	**0.64**	**2.51**	**0.64**
VGG19-U-Net	50	1	0.868	0.683	0.924	-	-	-	-	-	-
	50	2	0.868	0.686	0.926	0.876	0.713	0.938	**0.95**	**3.94**	**1.24**
	50	3	0.868	0.688	0.927	0.877	0.719	0.939	**1.00**	**4.58**	**1.27**
	50	4	0.870	0.692	0.929	0.878	0.723	0.942	**1.02**	**4.58**	**1.38**
	100	1	0.871	0.700	0.929	-	-	-	-	-	-
	100	2	0.872	0.701	0.933	0.874	0.708	0.936	**0.31**	**1.01**	**0.29**
	100	3	0.873	0.702	0.934	0.875	0.712	0.938	**0.33**	**1.38**	**0.41**
	100	4	0.871	0.699	0.932	0.875	0.718	0.937	**0.47**	**2.73**	**0.62**

Table 8.3.1. Main results. The "No partitioning" results denote a training procedure in which all network paths are trained with all samples from the training dataset (see Section 8.3.2) with the number of paths equal to the corresponding α ($\alpha = 1$ means that a single-path network is used). N denotes the total number of patients in the training dataset (before partitioning). SDSC results are demonstrated for two tolerance values. DSC and SDSC scores are rounded to three decimal places. The bold font indicates the improvements obtained with our method in each row.

ings, but such an experiment was not conducted due to limited available computation resources. We separately study the effect of our approach on the segmentation quality in three parts of the prostate: the middle part (*mid-gland*), the upper part (*base*), and the lower part (*apex*). Finally, we visualize the obtained segmentations.

As shown in Figure 8.4.2, the main improvement comes from the base of the prostate (on average, approximately 1.2% in DSC, 2.6% in SDSC with $\tau = 2mm$). A smaller improvement is observed in the apex, while in the mid-gland part, it is the most subtle. Naturally, our approach can result in larger improvements if the data variation is larger. As illustrated in Figure 8.4.1, the difference between the produced segmentation variants is largest in the base and apex parts. The quality of produced segmentations is the highest in the mid-gland part, however, our approach leads to an improvement in segmentation quality in all parts of the prostate. These results are in line with studies, e.g., [8] that show that the inter-observer variation is the largest in the base, smaller in the apex, and the smallest in the mid-gland.

We note that the improvement of our approach compared to a training procedure with no data partitioning is larger for larger α. This means that producing more segmentation variants helps to make at least one of the variants closer to the reference segmentation, though the data amount for training each of the decoders is smaller.

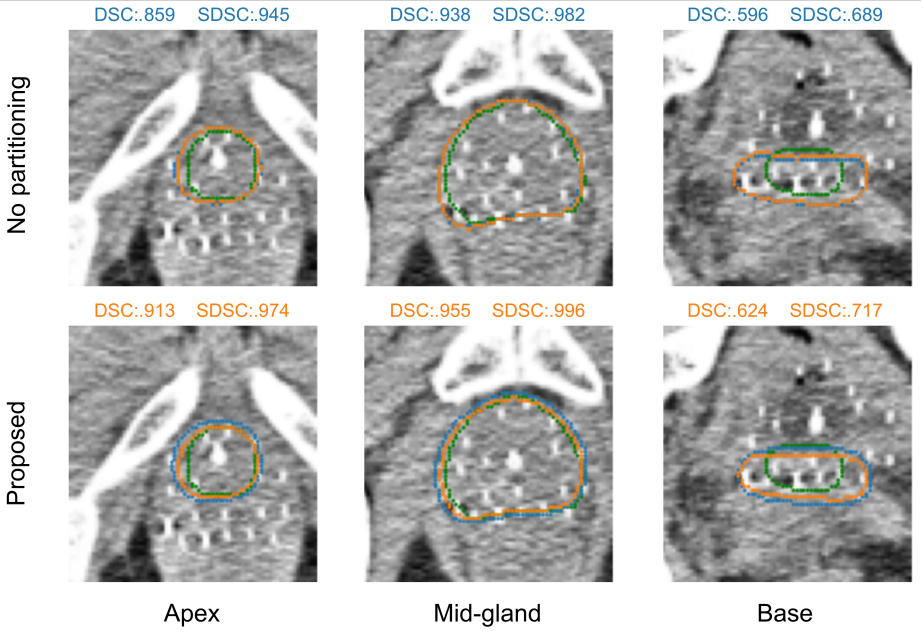


Figure 8.4.1. Results demonstration in case of $\alpha = 2$ for one of the patients. The reference segmentation is shown in green, the orange and the blue ones show two produced segmentation variants. The upper row corresponds to the training with no partitioning (see Section 8.4), and the bottom row corresponds to the training using the best-found partitioning. Scores for the selected (best among the two variants) segmentations are shown with a corresponding color. The proposed method produces more accurate segmentations. All images are zoomed in by a factor of two.

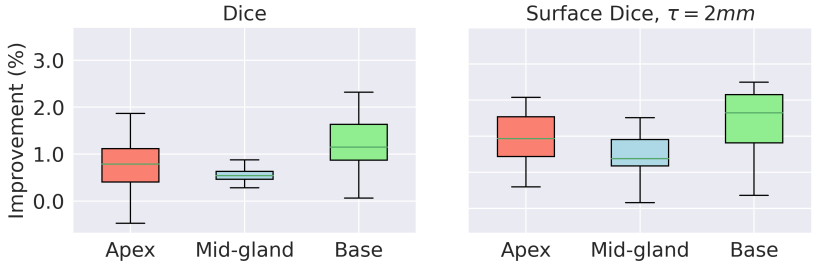


Figure 8.4.2. Improvement of the proposed approach for different prostate parts. Values are aggregated over all experimental setups with $N = 50$ (two types of nets, three values of α, three net initializations, see Section 8.3.2). Dice and Surface Dice with $\tau = 2mm$ performance metrics are shown. The lines inside the boxes represent the median values; the lower and upper whiskers represent the 5-th and the 95-th percentiles, respectively.

8.5. DISCUSSION

We applied the proposed segmentation approach to prostate segmentation on CT scans. However, our approach can be applied to other organs and image modalities as well.

Moreover, it is not limited to binary segmentation and can be naturally extended to a multi-organ segmentation task by corresponding modification of the segmentation network and segmentation quality metric (to a multi-class one).

In the interpretation of the results, we suggested that our algorithm managed to capture and utilize the observer segmentation variation present in the data. Such suggestion was supported by the fact that our algorithm showed the most substantial improvement in the base of the prostate - the same part which is known to have the largest inter-observer segmentation variation. However, it might be possible that there are other sources of variation in the data such as the presence of patients of different age groups, the use of varying imaging device settings, or different inclusion criteria of the seminal vesicles in the prostate delineated volume due to different degrees of prostate cancer spread into the seminal vesicles. Our algorithm does not distinguish between types of variation, and it only performs the partitioning to maximize segmentation quality. Making our approach insightful to end users (e.g., explicitly showing different types of variation in data) and validating the practical added value (i.e., reducing required time for scan delineation) is an interesting and important question for further research.

As mentioned in Section 8.4, a larger number of produced segmentation variants leads to better performance of our approach. However, for practical usage, many segmentation variants might not be convenient as a medical expert needs time to select one of them. We suggest that the selection of α is done according to the user preferences as a trade-off between the diversity of segmentation variants and the available time to assess them. Furthermore, we believe that a specialized graphical user interface needs to be developed and used in order to allow a medical expert to navigate through the suggested segmentation variants quickly and intuitively.

8.6. CONCLUSIONS

We have proposed a novel approach to medical image segmentation aimed at capturing variation in the data. Specifically, we proposed to achieve this by training a multi-path U-Net-like segmentation network such that each path is trained on a specific data subset. The best partitioning into subsets is found automatically by solving an optimization problem using the surrogate-assisted evolutionary algorithm SA-P3-GOMEA. In experiments with real clinical data consisting of CT scans with prostate segmentations, our approach was found to be able to detect the subsets of data automatically and produce multiple segmentation predictions, resulting in a performance increase of auto-segmentation compared to not using data partitioning. Analysis of the achieved improvement showed that most improvement was achieved for the base of the prostate. This corresponds to the literature reported findings that the largest inter-observer segmentation variation occurs at the base of the prostate. This is indicative that our approach is indeed capable of capturing observer variation in prostate segmentation.

ACKNOWLEDGMENTS

This work is part of the research programme Commit2Data (project number 628.011.012), which is financed by the Dutch Research Council (NWO).

REFERENCES

[1] S. Kohl et al. “A probabilistic U-Net for segmentation of ambiguous images”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[2] C. Rupprecht et al. “Learning in an uncertain world: representing ambiguity through multiple hypotheses”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3591–3600.

[3] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[4] S. Nikolov et al. “Clinically applicable segmentation of head and neck anatomy for radiotherapy: deep learning algorithm development and validation study”. In: *Journal of Medical Internet Research* 23.7 (2021), e26151.

[5] P. Yakubovskiy. *Segmentation Models*. https://github.com/qubvel/segmentation_models. 2019.

[6] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[7] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *Proceedings of the 3th International Conference on Learning Representations*. 2015.

[8] S. Montagne et al. “Challenge of prostate MRI segmentation on T2-weighted images: inter-observer variability and impact of prostate morphology”. In: *Insights into Imaging* 12.1 (2021), p. 71.

9

LIMITATIONS, DISCUSSION, AND CONCLUSIONS

This chapter concludes the thesis and is structured as follows. First, the answers to the research questions are given, including proposed solutions, achieved results, and limitations of the performed studies. Then, the general discussion considering all chapters of this thesis is presented. Finally, possible directions of future work are deliberated.

9.1. ANSWERS TO THE RESEARCH QUESTIONS

RESEARCH QUESTION 1

What is the best configuration of parameterless GOMEA for solving unconstrained combinatorial optimization problems?

In Chapter 2 we studied the effects of various design choices on the performance of GOMEA and how to best combine them to obtain the best-performing version of GOMEA. Such design choices are: the use of local search as part of GOMEA; whether to use a procedure to filter the linkage tree; whether to apply tournament selection before linkage learning; the metric used to build the Linkage Tree: mutual information or normalized mutual information; whether to use forced improvements; what FOS ordering to use in GOM (random or sorted by the element size). Moreover, we proposed a modified GOM (conditional GOM or CGOM) operator which takes into account weak dependencies between subsets of variables. Finally, we studied the performance of GOMEA with different automatic population management schemes. Experiments on a set of common benchmark functions showed that the obtained GOMEA version with CGOM as the variation operator outperforms previously published GOMEA versions (with GOM as the variation operator) as well as other modern model-based EAs such as DSMGA-II [1] and the originally proposed P3 algorithm [2] (not to be confused with the P3 population management scheme, the P3 algorithm proposed in [2] has other differences from previously

published GOMEA versions in addition to the P3 scheme). These results hold for both a single-population setup (when the optimal population size is obtained by use of bisection) and automatic population sizing. P3 was found to be the best-performing scheme for automatic population sizing. We found out that the two most impactful design choices are exhaustive donor search (which ensures that the GOM/CGOM operator produces a new candidate solution at each iteration) and the use of a hill climber (a single-iteration of local search which is applied to every new solution before it undergoes GOM). The best linkage tree building algorithm uses normalized mutual information as the variables similarity metric, and linkage tree filtering to remove redundant variable subsets.

The main limitation of this study is the use of only standard benchmark functions in the experiments instead of real-world optimization problems. However, we would like to point out that finding a collection of representative real-world problems is not trivial. Moreover, there are recent works [3] which aim at matching real-world problems to benchmark functions by using exploratory landscape analysis. This way, the performance of an algorithm (relative to the alternative algorithms) on a new real-world optimization problem might be approximately derived from its performance on the found to be similar functions from the benchmark set.

RESEARCH QUESTION 2

Is it possible to efficiently create a perfectly accurate surrogate model in a black-box optimization scenario (and then use it to solve an optimization problem)?

In Chapter 3 we proposed to use the Walsh decomposition as the surrogate model with learnable coefficients. These coefficients can be learned such that the Walsh decomposition values perfectly correspond to the values of the original fitness function. In contrast to other works considering Walsh decomposition as the surrogate model [4, 5], we aimed at not approximating the Walsh coefficients, but rather finding their *exact values*. We showed that for a special class of functions the proposed algorithm for learning exact Walsh coefficients has complexity $\mathcal{O}(\ell log \ell)$ in terms of fitness evaluations (ℓ is the problem size). This class of functions is the class of pseudo-Boolean *k-bounded* functions with *a linear number of subfunctions*: 1. $f(x) : \{0,1\}^\ell \mapsto \mathbb{R}$; 2. Function $f(x)$ is decomposable into subfunctions: $f(x) = \sum_i^M g_i(x)$, each of $g_i(x)$ comprises not more than k variables; 3. The number of subfunctions M is linear in ℓ: $M = \mathcal{O}(\ell)$. Examples of such functions are concatenated deceptive traps, NK-landscapes, and some instances of NP-hard problems MAX3SAT (with $\mathcal{O}(\ell)$ number of clauses) and Max-Cut (sparse graphs).

After obtaining a perfectly accurate Walsh decomposition, we used GOMEA (in fact, an older version than the GOMEA obtained in Chapter 2) to solve the original optimization problem by fully replacing the real fitness evaluations with the surrogate ones (computed by the Walsh decomposition-based surrogate function). This modified optimization task is essentially gray-box as it is known which variables appear together in the subfunctions. Thus, a powerful feature of GOMEA - *partial evaluations* could be used to speed-up the optimization process.

Due to its excellent scalability - $\mathcal{O}(\ell log \ell)$ in terms of real fitness evaluations, the proposed optimization approach outperformed standard GOMEA (without using any

surrogate model) by a large margin as well as alternative approaches to Walsh coefficients learning [6, 4].

The proposed approach considers the black-box optimization scenario and, therefore, focuses on learning problem structure as it is not known a priori. However, it can be argued that many real-world problems are, in fact, gray-box. Therefore, the linkage learning part is not needed to solve them. We believe, nevertheless, that among real-world problems there are some essentially black-box ones, for instance, Neural Architecture Search (NAS). Hence, efficient black-box optimization algorithms are still highly demanded. The key limitation of the proposed algorithm is the assumption that the function is separable with subfunctions having a limited number of variables k ($k << \ell$), and moreover, a linear number of subfunctions. Modification of the algorithm to allow working with other types of functions, for instance, by using a partial instead of a full Walsh decomposition, might be a required crucial step towards wider practical applicability.

RESEARCH QUESTION 3

How can GOMEA be adapted to effectively find high-quality solutions for expensive combinatorial optimization problems and show state-of-the-art performance on various problems (including real-world ones) with up to hundreds of binary and higher cardinality discrete variables?

In Chapter 4, we proposed the first version of surrogate-assisted GOMEA called *CS-GOMEA*. That version is based on the previously available version of GOMEA [7], which uses the Interleaved Multistart Scheme (IMS) [8, 9] to replace manual population size tuning. We proposed to use a small Convolutional Neural Network (CNN) as the surrogate model. The main motivation for it was the efficiency of CNNs in the scenarios when the same patterns appear in different spatial locations of the input data. With regard to surrogate modeling of fitness functions, this means that the same subfunctions which take different variables can be efficiently approximated, i.e., the same learnable weights of the CNNs can be used for that. An example of such a function is the concatenated deceptive trap: the same subfunctions (deceptive traps) are applied to different variables of the solution. We showed that the use of a CNN allowed for efficient surrogate modelling on rather small data samples in case of the optimization of problems with up to hundreds of variables comprising repeating subfunctions. In the experiments on commonly used benchmark functions such as deceptive traps [10], NK-landscapes [11], and hierarchical if-and-only-if, CS-GOMEA outperformed standard GOMEA and Bayesian optimization algorithms SMAC [12] and Hyperopt [13].

This first version of the surrogate-assisted GOMEA (i.e., CS-GOMEA) was limited to binary variables only. The assumption of the existence of repeating subfunctions in the objective function which makes the usage of the CNN surrogate effective, can also be considered to be a limitation in applying the algorithm to real-world optimization problems, as repeating subfunctions are not necessarily present in them.

Based on the study performed in Chapter 2, we proposed a more efficient and more generic version of surrogate-assisted GOMEA in Chapter 5 called *SAGOMEA*. That version

includes improvements shown to be beneficial for the performance of GOMEA in Chapter 2 such as linkage tree filtering with normalized mutual information as the variables similarity metric, and, importantly, P3 for automatic population management. Moreover, we aimed at applying it to real-world problems. As an example of such a problem we chose a machine learning problem, namely, finding the best dataset partitioning in order to get the best-performing ensemble of models. Each model in an ensemble is trained on its own data subset. As this problem does not have a clear linkage structure in contrast to benchmarks considered in Chapter 4, instead of using a CNN as a surrogate model, we considered multiple types of surrogate models (Support Vector Regression (SVR), random forest, multilayer perceptron, gradient boosting) and compared their performance. Interestingly, the use of SVR resulted in the best performance of SAGOMEA on the considered optimization problem. In general, our algorithm includes automatic surrogate model tuning, and, therefore, the chosen model is adapted to a specific optimization problem. Another key improvement of SAGOMEA compared to the algorithm presented in Chapter 4 is the ability to handle discrete problems with variables that are not necessarily binary. In the conducted experiments, problems had variables with a cardinality up to 10. In comparison to the well-known SMAC and Hyperopt algorithms, as well as non-surrogate-assisted GOMEA and local search, SAGOMEA showed the best performance.

The main design principle of SAGOMEA is relying on surrogate fitness evaluations most of the time during the search process, and performing a real fitness evaluation when the surrogate fitness of the solution is relatively high. The variation operator of SAGOMEA is GOM. By design, GOM creates many potential offspring solutions for each solution using the linkage model and evaluates them. The number of potential offspring in the case of a full linkage tree is $2\ell - 1$ (ℓ is the number of variables). This means that $2\ell - 1$ potential fitness evaluations are performed for each solution in the population in each generation. In the case of the filtered linkage tree, this number is likely lower, but still quite large ($\mathcal{O}(\ell)$). It might happen that many of these offspring solutions have a high surrogate fitness (because they are indeed highly fit solutions, or, perhaps, because the surrogate model is inaccurate in the vicinity of the current solution), which means performing a lot of real fitness evaluations. Performing many real fitness evaluations is undesirable for the expensive optimization scenario and, moreover, many of them might be redundant because the solutions are similar to each other. This limitation of the effectiveness of SAGOMEA might be, potentially, tackled by limiting the number of iterations inside GOM, or the number of performed real evaluations of a solution's offspring.

RESEARCH QUESTION 4

Does the innate stochasticity of fitness evaluations in Neural Architecture Search represent a significant hurdle for optimization algorithms, and, if yes, can it be alleviated?

In Chapter 6, we showed that fitness functions in NAS contain a substantial amount of noise. We considered a single-objective scenario in which only model quality (e.g., accuracy in the case of a classification task) is optimized and used as the fitness function.

The stochasticity of the fitness function was demonstrated by calculating the rank correlation between performance scores of the same architectures but: 1) Trained with two different random seeds (including different initializations of the weights); 2) Validated on different data subsets. The obtained rank correlation values were especially low (<0.2) for the top 20% of the architectures discovered by search algorithms. This result holds for both classification (CIFAR-10 dataset, tabulated results from the NAS-101 benchmark [14]) and medical image segmentation. For the medical image segmentation task, two open-source datasets (prostate segmentation and heart segmentation tasks) were used. The search was performed using four different search algorithms, two general-purpose ones (GOMEA and local search) and two expensive optimization algorithms (Hyperopt and SAGOMEA). To reduce the amount of noise in fitness evaluations, we proposed to use cross-validation, optionally repeated multiple times to reduce the noise even further. We studied a trade-off between a larger amount of computationally cheaper, but more noisy fitness evaluations and a fewer number of more robust fitness evaluations (which use the cross-validation technique). Such a trade-off arises when a fixed computational budget (in terms of consumed hours for training networks) is available. Results demonstrated that with a large enough computational budget, more robust fitness evaluations lead to finding better-performing architectures (according to an independent evaluation). This finding is independent of the used search algorithm.

The main limitation of the performed study is its basic NAS setup: in each fitness evaluation, an architecture is trained from scratch. For large datasets, such as ImageNet, such an approach might be prohibitively computationally expensive. An efficient NAS procedure using stochastic gradient descent [15] was proposed in order to reduce the total number of required network training to just one. Though stochastic gradient descent was shown to be a suboptimal search algorithm [16], it is often used in practice due to its computational efficiency. Studying the stochasticity of fitness functions in more advanced NAS approaches might be important for a better understanding of the practical considerations of their usage.

RESEARCH QUESTION 5

Can deep learning and optimization techniques be combined to establish a novel medical image segmentation approach that is capable of explicitly capturing and exploiting observer variation in the dataset?

We proposed the following approach to medical image segmentation, which is capable of capturing and exploiting observer variation in a dataset. Assuming that a dataset consisting of scans along with their segmentations collected in a hospital might be highly heterogeneous (e.g., containing scans segmented by different observers), it might be beneficial to train a collection of deep learning models on homogeneous subsets of such a dataset. After training, a collection of segmentations (one by each network) can be automatically produced.

However, it is not clear, a priori, which scans should be grouped together to obtain the best results. We proposed to consider this dataset partitioning task as a black-box

optimization problem. The goal of the optimization is to find the subsets which correspond to the best performance when used to train neural networks on them. One fitness evaluation, therefore, entails the training of a collection of networks on data subsets. This is a computationally expensive task (in our experiments, one fitness evaluation takes several minutes), hence, algorithms designed for expensive optimization should be used. For solving this optimization problem, we used the surrogate-assisted GOMEA algorithms we developed.

In the proof-of-principle of the proposed approach presented in Chapter 7 we demonstrated that our novel approach works in the case of artificially generated segmentation variations that simulate different styles of segmenting an organ (e.g., always over- or under-segmenting it). We used a publicly available prostate segmentation dataset from the Medical Segmentation Decathlon. For that study, we used the version of surrogate-assisted GOMEA that works with binary variables only (CS-GOMEA from Chapter 4). Thus, partitioning into more than two subsets was not straightforward and was proposed to be done in a hierarchical way, i.e., partitioning in two parts recursively. Our algorithm successfully revealed segmentation variations in the data (up to three different variations were used) and obtained a partitioning of the data into homogeneous groups. Multiple segmentation networks trained on the obtained data subsets corresponded to different segmentation styles, ultimately resulting in producing predictions that match simulated variations much closer than a traditional deep learning approach of training only one network on the full dataset.

In Chapter 8, a more advanced version of this approach was proposed. Namely, the following improvements were added: 1) Direct partitioning of the data into an arbitrary number of subsets. This became possible as the newer version of surrogate-assisted GOMEA was used (SAGOMEA from Chapter 5); 2) Also a more efficient training setup was used: a multi-head U-Net instead of several standalone networks. Thus, a shared part of such a U-Net (the encoder) can be trained on all available data, while the decoders of it are trained on separate data subsets, hence, enabling specialization to different variations in the data.

We see two limitations of the proposed approach. The first potential issue is the growing computational demand when the number of scans in the dataset, i.e., the number of variables in the optimization problem, is increasing. In hospitals, real clinical datasets are usually of moderate size (up to a few hundred scans), however, datasets gathered across multiple hospitals might contain up to tens of thousands of scans [17]. For a dataset of such size, the partitioning optimization problem (with such a number of variables) becomes extremely difficult and substantial computational effort might be required to find better than randomly sampled solutions. Though it might be overcome by applying partitioning hierarchically, such a usage scenario needs a separate study. The second limitation is that the algorithm by design does not distinguish between observer- and data-variation. Though training models on homogeneous scans (for instance, each obtained subset might contain scans made with a particular scanning device settings) can be beneficial for the model performance, the data partitioning found by the algorithm is not guaranteed to correspond to different groups of observers. To make the model focus only on observer variation, the dataset should preferably be homogeneous in all other aspects (e.g., used scanning devices and their settings).

RESEARCH QUESTION 6

Is the proposed observer variation-aware medical image segmentation method capable of producing high-quality and diverse segmentation variants on a real clinical dataset?

The more advanced version of the variation-aware segmentation algorithm proposed in Chapter 8 was applied to a real clinical dataset from the Mount Vernon Cancer Centre, United Kingdom consisting of CT scans used for external beam radiotherapy and brachytherapy treatments. In our study, we used only scans used for brachytherapy. The considered segmentation task was binary prostate segmentation (the two classes were prostate and background). The dataset size was 196 scans. The partitioning algorithm was applied with different numbers of subsets: 2, 3, and 4. Interestingly, the improvement obtained by the proposed approach slightly increases when more data subsets are used, which indicates that there is no problem of lack of data for training subnetworks on data subsets. Importantly, the performance improvement varies a lot for different parts of the prostate: it is larger for the base and apex parts of the prostate and smaller for the mid-gland part. From the literature, it is known that inter-observer variation is the largest for these base and apex parts and much smaller for the mid-gland. Thus, it might indicate that our algorithm managed to capture observer variation in the dataset, and therefore, showed the largest improvement for the parts which are known to be prone to larger inter-observer variation.

9.2. GENERAL DISCUSSION

In this thesis, different modifications made to GOMEA were shown to result in better performance than alternative approaches (other modern model-based EAs and BO algorithms) on optimization benchmarks. In some chapters of this thesis, the developed EAs were tested on sets of established benchmark functions, which include the deceptive traps and hierarchical-if-and-only-if (HIFF). Such functions were designed specifically to test particular properties of EAs (e.g., linkage learning efficacy, ability to tackle hierarchically composed problems). While these benchmarks are indeed useful to develop algorithms that are better at solving these problems and to quantitatively measure progress in the field, it does not necessarily translate to better performance on real-world problems. We also compared the developed EAs against BO algorithms and concluded that surrogate-assisted EAs have great potential and in many cases outperform BO algorithms. It should be noted, however, that a comprehensive comparison of EAs against BO algorithms is, in general, missing in the current literature as they are usually tested on completely different sets of problems (due to historical reasons and perhaps lack of interaction between different research communities). The creation and acceptance of a benchmark consisting of diverse, real-world, challenging problems by the optimization community (not limited to the EA community only) would likely accelerate the progress in the EA field (likely in other optimization fields as well) and allow to better compare EAs against other types of algorithms, for instance, BO algorithms and, potentially, show their added value. It is

noteworthy however that the creation of such a benchmark is challenging as there is no universal method of finding interesting problems from the perspective of optimization complexity as well as representativeness of real-world problems.

NSGA-II and CMA-ES are quite popular choices for solving optimization problems. However, they are not necessarily the best-performing EAs for, correspondingly, multi-objective and single-objective real-valued search domains (for instance, compared to the GOMEA family of algorithms in certain cases, especially, in a GBO setting). It would be interesting to figure out what the main reasons are for that and what can be done to increase the use of GOMEA algorithms by practitioners. When facing an optimization problem to solve, a practitioner is probably interested in an algorithm that: 1) Was shown to perform well across a variety of problems, and importantly, does completely fail on some problems; 2) Does not require hyperparameter tuning on a particular problem; 3) Is easy-to-use: this entails available open-source implementation, preferably having a Python wrapper (the algorithm itself might be well implemented in C/C++); 4) Ideally parallelizable and/or GPU-accelerated to enable large-scale optimization. The main steps in a wider GOMEA family of algorithms adoption is then probably a better availability of easy-to-use, well-documented source codes of the algorithms. GOMEA versions should also be evaluated and, if necessary, improved, on scenarios that occur in real-world optimization problems. For instance, noisy fitness functions (they are present in many real-world problems, especially if they are related to ML), might be a serious problem for GOMEA due to its local search type behavior of accepting improving moves.

More than 1000 papers have been published on NAS in recent years [18]. However, it should be noted that the recent state-of-the-art architectures for computer vision were either designed manually (ConvNeXt [19], CLIP [20]), using a simple grid search over possible design options (EfficientNet [21]), or using random search (EfficientNetV2 [22]). Thus, at this point it remains unclear whether NAS can result in finding state-of-the-art architectures in the domains where there are many attempts to design new architectures manually. For domains and problems with no strong manually designed baselines available, NAS was shown to have an added practical value [23] in some cases. The most challenging and fundamentally problematic part about NAS appears to be defining a good search space: it should not be too constrained to allow for finding some novel architectures, yet a too large search space makes effective search computationally infeasible. The currently used popular search spaces (for instance, DARTS [15]) allow finding only slightly better-performing architectures on particular datasets (which can be nevertheless valuable), but not a discovery of a next break-through idea in a neural network design such as attention [24]. Using NAS on an arbitrary dataset to obtain some (even subtle) performance improvement is also not straightforward as the main problem in using NAS (and hyperparameter optimization techniques) for small-scale datasets is, based on experiments in this thesis, a poor correlation between the results obtained for the validation and test data sets. This problem is especially problematic for medical datasets.

Using a larger search space for NAS should be accompanied by an efficient search algorithm (probably more complex than, for instance, a local search). This is also not sufficiently addressed in the current literature on NAS. The rapid progress in DL over the last few years can be to some point attributed to the existence of established benchmarks

(besides algorithmic advances and huge improvements in GPU performance): (relatively) small-scale CIFAR-10/100 benchmarks and the large-scale benchmark ImageNet. These benchmarks are (were at some point) challenging and allowed to highlight the advances in algorithms for DL which can be transferable to other tasks and datasets. For instance, the Residual Network (ResNet [25]) was first shown to significantly improve the performance on ImageNet and CIFAR-10/100, and then its main idea of the skip-connection was also shown to be beneficial for different computer vision tasks, and datasets. In NAS, however, the same goal (i.e., existence of challenging benchmarks which encourages the development of novel, better-performing algorithms) was not achieved with the introduction of benchmarks for NAS. The existing benchmarks (for instance, NAS-Bench-101 [14], NAS-Bench-201 [26]) are rather easy from the optimization perspective: the gap in performance between simple local search (or even random search) and more advanced search algorithms (BO-based, EAs) is subtle, if it exists at all. Creating more complex (open-ended) search spaces and benchmarks can boost the development of efficient search algorithms for NAS and this field in general.

In this thesis, a lot of work involved the usage of image segmentation networks, for instance, the U-Net. Many conducted experiments led to the conclusion that likely the best framework for practical usage of DL for medical image segmentation is the nnU-Net [27]. To better understand how its components work in detail, it is better, to use and modify the functions from its repository separately (instead of just running it as a black-box segmentation pipeline on a particular dataset). An interesting finding is that downsampling the scans to a lower resolution (to reduce the amount of required computational resources to train a network) might be rather detrimental to the segmentation quality. Instead, if training on scans in the original resolution is not computationally feasible, a technique from the nnU-Net should be used: training a network on (randomly) extracted patches of a smaller size from the scans (without downsampling them).

Another finding about applying DL to medical image segmentation from the experiments performed in this thesis is that apparently, the impact of the training pipeline composition (data preprocessing, data augmentations, training hyperparameters) is typically larger than the impact of the choice of neural network architecture. This conclusion was also reached in the nnU-Net work and based on observed performance on various segmentation benchmarks (diverse image modalities, organs to segment, and segmentation task difficulty), outperforming more complex architectures (used with apparently suboptimal other components of the training pipeline). When applying a deep neural network to segmenting a real medical dataset (not a benchmark dataset) there is also a question about the amount and quality of the data. Based on the work conducted in this thesis, to obtain a reasonably well-performing segmentation model, at least 50 labeled (with sufficient quality) scans are needed. Works that claim obtaining clinically applicable (achieving expert-level quality) segmentation models required a significant amount of labeled data: in [28] 663 scans were used for training, in [29] more than 14000 scans. In [17] it was shown that using self-supervision techniques is beneficial, but the used unlabeled datasets need to be even larger (for some tasks, more than 100000 scans were used). From a practical point of view, if the goal is to achieve the maximal possible performance on a particular segmentation task and GPU resources are limited, it appears best to invest more resources into improving data quality and, ideally, collecting additional high-quality

data (if possible) rather than investing them into neural network tuning experiments such as NAS. However, the computational resources should be sufficient to perform training in high resolution in 3D.

9.3. FUTURE WORK

An interesting future work direction regarding surrogate-assisted GOMEA is extending it to multi-objective optimization problems. Such an extension is potentially interesting for NAS (two objectives are usually model quality vs. consumption of resources) and multi-objective hyperparameter search (two objectives can be, for instance, model quality and model fairness [30]. Model fairness is a measure of bias in model predictions originating from a sensitive attribute, e.g., gender or race. Minimizing such bias has been recently increasingly viewed as a crucially important property of ML models. There are some works on multi-objective BO [30], and it would be interesting to study whether surrogate-assisted EAs can outperform them, similarly to the single-objective case. A straightforward application of single-objective SAGOMEA to multi-objective problems might simply use scalarized objectives, but it might happen that such an approach underperforms compared to genuinely multi-objective algorithms, which are capable of optimizing the solutions along the entire Pareto front simultaneously. Thus, a deeper modification of surrogate-assisted GOMEA would be required, for instance, the usage of separate surrogate models for each objective and the integration of a domination-based technique for ranking solutions.

As it was shown in Chapter 6 for NAS, noise in the fitness function often plays a significant role in real-world optimization problems. However, as demonstrated in Chapter 5, when applying GOMEA to noisy fitness functions the performance deteriorates dramatically. The same problem holds for the surrogate-assisted version of GOMEA. The main reason for that is that by design GOM greedily accepts improving solution changes and as such, was not designed with noisy fitness evaluations in mind. In the presence of noise, spurious improvements due to one lucky sample from the noisy fitness function cannot be reversed, and the search process might divert to local optima with a larger probability. For the wider applicability of GOMEA and surrogate-assisted GOMEA to real-world optimization problems (including NAS), it is important to adapt them to better deal with noisy fitness functions. There are multiple ways to achieve this. For instance, a general approach is repeating and averaging fitness evaluations to improve the solution fitness estimate. A more specific modification is to fundamentally change the GOM operator, specifically, its greedy acceptance mechanism, for instance, replacing it with probabilistic acceptance or using predicted fitness variance in the case of surrogate-assisted GOMEA.

Automated medical image segmentation algorithms have the potential to further improve the radiation treatment workflow. Replacing manual scan segmentation by automatic segmentation can make the whole process faster and reduce the workload of clinicians. Because the current state of AI and legal regulations do not allow yet a fully automatic procedure, the proposed variation-aware approach has a particular benefit. A clinician can choose the segmentation variant among multiple automatically produced ones which requires the least number of manual correction actions. The usage of advanced optimization and other AI methods [31, 32] for the subsequent radiation treatment planning part has already been shown to have an added value in clinical usage. Radiation

treatment planning requires target (region of interest to treat) segmentations, as well as segmentations of the surrounding organs at risk. In this thesis, we focused on prostate segmentation only (which is both a target volume and an organ at risk in the case of brachytherapy). One of the main reasons for that is the incompleteness of organs at risk (e.g., rectum) segmentations in the clinical data. Often only parts of organs at risk are segmented in the case of brachytherapy, since the part of the organs that is far away from the implant will receive so little dose that it will not influence treatment planning. Moreover, the relative sizes of segmented parts are usually patient-dependent and therefore induce inconsistencies in the dataset. However, producing segmentations for all organs at risk is essential for the integration of an automatic segmentation method into the radiation treatment workflow, and therefore handling such inconsistencies should be considered as a necessary future work direction. The second potential work direction pursuing the goal of making radiotherapy faster (by spending less time on the organ delineation part), more accurate, and less labor-intensive, is connecting automatic segmentation with already automatic treatment planning optimization such as BRIGHT [33]. Alternatively, to using manually created segmentations as it is being used now, BRIGHT might potentially use automatically produced ones (after a check by a clinician and a manual correction if necessary). BRIGHT was already shown to be able to optimize treatment plans robustly [34] and deal with slight variations in the interpolation of manual segmentations needed to create full 3D organ shapes. Automatically produced different segmentation variants can be used for further development of more robust BRIGHT versions.

Related to the above-discussed practical application considerations of automatic segmentation methods and future work directions, is the consideration of method explainability. In general, the explainability of AI methods is important for wider acceptance and application in many domains, but especially, in healthcare. Improving the explainability aspect of the proposed variation-aware segmentation approach should be two-fold. Firstly, with a deep learning model we can not only produce organ segmentations, but we can also in some form derive why certain areas are (not) segmented. For instance, it can be done (but not limited to) by highlighting the areas on the scan that the model paid the most attention on [35] (note however that any such technique should be used with keeping in mind the interpretability gap [36]). Explainability of an image segmentation method might also be used as a tool to check that the model acts reasonably, and, for instance, is not overfitting on some details of the image [37]. Secondly, the results of the observer variation-aware segmentation method can be better explained if the found differences between groups of scans and/or their segmentations in different dataset partitions are shown. Thus, the differences between the groups of scans in the partitioning can be demonstrated and linked to the differences in the predicted segmentation variants. To sum up, improving the explainability aspect of the proposed medical image segmentation method is an important future work direction.

REFERENCES

[1] S.-H. Hsu and T.-L. Yu. “Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: DSMGA-II”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2015, pp. 519–526.

[2] B. W. Goldman and W. F. Punch. “Fast and efficient black-box optimization using the parameter-less population pyramid”. In: *Evolutionary Computation* 23.3 (2015), pp. 451–479.

[3] A. Thomaser, A. V. Kononova, M.-E. Vogt, and T. Bäck. “One-shot optimization for vehicle dynamics control systems: towards benchmarking and exploratory landscape analysis”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2022, pp. 2036–2045.

[4] S. Verel, B. Derbel, A. Liefooghe, H. Aguirre, and K. Tanaka. “A surrogate model based on Walsh decomposition for pseudo-Boolean functions”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 181–193.

[5] F. Leprêtre, S. Verel, C. Fonlupt, and V. Marion. “Walsh functions as surrogate model for pseudo-boolean optimization problems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 303–311.

[6] R. B. Heckendorn and A. H. Wright. “Efficient linkage discovery by limited probing”. In: *Evolutionary Computation* 12.4 (2004), pp. 517–545.

[7] P. A. N. Bosman and D. Thierens. “Linkage neighbors, optimal mixing and forced improvements in genetic algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2012, pp. 585–592.

[8] G. R. Harik and F. G. Lobo. “A parameter-less genetic algorithm.” In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vol. 99. 1999, pp. 258–267.

[9] N. H. Luong, H. La Poutré, and P. A. N. Bosman. “Multi-objective Gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start scheme”. In: *Swarm and Evolutionary Computation* 40 (2018), pp. 238–254.

[10] K. Deb and D. E. Goldberg. “Sufficient conditions for deceptive and easy binary functions”. In: *Annals of Mathematics and Artificial Intelligence* 10.4 (1994), pp. 385–408.

[11] M. Pelikan, K. Sastry, D. E. Goldberg, M. V. Butz, and M. Hauschild. “Performance of evolutionary algorithms on NK-landscapes with nearest neighbor interactions and tunable overlap”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2009, pp. 851–858.

[12] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Sequential model-based optimization for general algorithm configuration”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 507–523.

[13] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. “Hyperopt: a Python library for model selection and hyperparameter optimization”. In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[14] C. Ying et al. “NAS-Bench-101: towards reproducible neural architecture search”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2019, pp. 7105–7114.

[15] H. Liu, K. Simonyan, and Y. Yang. “DARTS: differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).

[16] A. Yang, P. M. Esperança, and F. M. Carlucci. “NAS evaluation is frustratingly hard”. In: *arXiv preprint arXiv:1912.12522* (2019).

[17] S. Azizi et al. “Robust and efficient medical imaging with self-supervision”. In: *arXiv preprint arXiv:2205.09723* (2022).

[18] C. White et al. “Neural architecture search: insights from 1000 papers”. In: *arXiv preprint arXiv:2301.08727* (2023).

[19] Z. Liu et al. “A ConvNet for the 2020s”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11976–11986.

[20] A. Radford et al. “Learning transferable visual models from natural language supervision”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2021, pp. 8748–8763.

[21] M. Tan and Q. Le. *EfficientNet: rethinking model scaling for convolutional neural networks*. PMLR, 2019.

[22] M. Tan and Q. Le. “EfficientNetV2: smaller models and faster training”. In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2021, pp. 10096–10106.

[23] R. Tu et al. “NAS-Bench-360: Benchmarking neural architecture search on diverse tasks”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 12380–12394.

[24] A. Vaswani et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* 30 (2017).

[25] K. He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[26] X. Dong and Y. Yang. “NAS-Bench-201: extending the scope of reproducible neural architecture search”. In: *arXiv preprint arXiv:2001.00326* (2020).

[27] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. In: *Nature methods* 18.2 (2021), pp. 203–211.

[28] S. Nikolov et al. “Clinically applicable segmentation of head and neck anatomy for radiotherapy: deep learning algorithm development and validation study”. In: *Journal of Medical Internet Research* 23.7 (2021), e26151.

[29] D. Fauw et al. “Clinically applicable deep learning for diagnosis and referral in retinal disease”. In: *Nature Medicine* 24.9 (2018), pp. 1342–1350.

[30] F. Karl et al. “Multi-objective hyperparameter optimization-an overview”. In: *arXiv preprint arXiv:2206.07438* (2022).

[31] D. L. J. Barten et al. “Towards artificial intelligence-based automated treatment planning in clinical practice: a prospective study of the first clinical experiences in high-dose-rate prostate brachytherapy”. In: *Brachytherapy* 22.2 (2023), pp. 279–289.

[32] E. Y. Han et al. “Clinical implementation of automated treatment planning for whole-brain radiotherapy”. In: *Journal of Applied Clinical Medical Physics* 22.9 (2021), pp. 94–102.

[33] D. Barten et al. “PD-0821 Artificial Intelligence based planning of HDR prostate brachytherapy: first clinical experience.” In: *Radiotherapy and Oncology* 161 (2021), S653–S655.

[34] M. C. van der Meer et al. “Robust optimization for HDR prostate brachytherapy applied to organ reconstruction uncertainty”. In: *Physics in Medicine & Biology* 66.5 (2021), p. 055001.

[35] A. Singh, S. Sengupta, and V. Lakshminarayanan. “Explainable deep learning models in medical image analysis”. In: *Journal of Imaging* 6.6 (2020), p. 52.

[36] M. Ghassemi, L. Oakden-Rayner, and A. L. Beam. “The false hope of current approaches to explainable artificial intelligence in health care”. In: *The Lancet Digital Health* 3.11 (2021), e745–e750.

[37] M. T. Ribeiro, S. Singh, and C. Guestrin. “"Why should I trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 1135–1144.

ACKNOWLEDGEMENTS

Being at the end of my Ph.D. journey, I would like to thank all the people who have helped me make it thus far.

First of all, I would like to thank my supervisors: Peter and Tanja. From them, I got inspired by the beauty of the evolutionary computation field and also learned what it means to do good scientific research in general. I appreciate their open-mindedness and willingness to let me explore and experiment with diverse research ideas. After our weekly meetings, I felt reassured about the research direction I was heading in, even in the moments when I had felt lost and confused before them. This is sometimes taken for granted but I also want to acknowledge the hard work of Peter and Tanja and their dedication to building the research environment which provides many opportunities.

I want to thank our project partners from Amsterdam UMC: Bradley, Henrike, Karel, and Jan for their enthusiasm about our project, invaluable help with collecting the data, performing the study of our method, and a lot of feedback and great ideas. Secondly, I thank our project partners from the Amsterdam eScience Center, especially, Adriënne who helped me at the beginning of the project to deep dive into medical imaging field. I also want to thank Gerry Lowe and his team from the Mount Vernon Cancer Centre who helped us to collect data and conduct the first real-world study of our proposed medical image segmentation method.

Then, I want to thank fellow Ph.D. students from the former Life Sciences and Health (now called Evolutionary Intelligence) group at CWI and Amsterdam UMC. One of the first people I met at CWI (and in the Netherlands) were Marco, Anton, Stef, Hoang, Marjolein, and Ziyuan. I wish to thank them for accommodating me when I started my Ph.D. and making me feel like home in the group. It was a lot of fun and inspiration in discussing science and sharing ideas during coffee breaks. Beside that, playing ping-pong matches while experiments were running was also very brain-refreshing and energizing. I want to thank Monika for insightful conversations about medical imaging and deep learning and for being always optimistic about my papers under review, sometimes more than I was myself. Then, I particularly thank my paranymphs who have also become good friends along my journey. I wish to thank Marco for being always incredibly supportive and ready to listen and help with anything. I thank Alex who joined our group when I was already at a late stage of my Ph.D. but he definitely brought a lot of fun into my Ph.D. journey through (but not limited to!) hours-long conversations about AutoML, new research ideas, and life.

Last but not least, I cannot thank enough my family. I can always feel the support from my parents and grandparents, even when they are far away. Knowing that you always believe in me, love me, and care, is something I am absolutely lucky to have. And finally, I thank the love of my life, Vera. One of the things I have learned during my Ph.D. about myself is that hard work and scientific achievements are not fulfilling on their own. You make my life complete, all joyful moments greater and all tough ones easier to overcome.

CURRICULUM VITÆ

Arkadiy was born in Stupino, Moscow Region (Russia) on May 27th 1995. He obtained his bachelor's degree in Applied Mathematics and Computer Science at the Higher School of Economics (Moscow), faculty of Computer Science. The specialization of his study was Algorithms and Data Science. The idea of the bachelor's thesis was to combine Genetic Programming and ensembling techniques for creating an automatic trading strategy. He obtained his Master of Science degree at the Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics. The degree specialization was Supercomputer Systems and Applications. The Master's thesis topic was analysis of a Convolutional Neural Network computation time on a Graphical Processing Unit and creating a predictive model for computation time estimation for new devices.

During his university study, he worked as a Data Scientist Intern at Agima Digital Agency and Habidatum in Moscow in 2015; as a junior researcher at Wunderfund in Moscow in 2015-2017; and as a junior researcher at LG Electronics R&D Lab in Moscow in 2017. In September 2018 he started his doctoral research at Life Sciences and Health group of the Dutch National Research Institute for Mathematics and Computer Science (Centrum Wiskunde & Informatica) in Amsterdam, the Netherlands. Netherlands eScience Center also participated in his research project.

His main research topics of interest are Evolutionary Computation and Deep Learning. He has expertise in designing and applying Evolutionary Algorithms in the domain of combinatorial optimization problems with computationally expensive fitness evaluations. He has experience in using Deep Learning techniques for medical image segmentation, along with designing new segmentation methods consisting of Evolutionary Algorithms and Deep Learning combinations. He is also interested in AutoML (Neural Architecture Search and hyperparameter optimization) in the realm of Deep Learning. During his doctoral research at Centrum Wiskunde & Informatica, Arkadiy has also collaborated with the Amsterdam University Medical Centres (location AMC) and Mount Vernon Cancer Centre (the UK) in order to include real-world medical data in his research and get insights from clinicians about their vision of the contribution of Deep Learning to healthcare.

LIST OF PUBLICATIONS

12. A. Chebykin, **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "Shrink-Perturb improves architecture mixing during Population Based Training for neural architecture search". In: *Proceedings of the European Conference on Artificial Intelligence (accepted for publication)*. 2023

11. **A. Dushatskiy**, A. Chebykin, T. Alderliesten, and P. A. N. Bosman. "Multi-objective Population Based Training". In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2023, pp. 8969–8989.

10. **A. Dushatskiy**, M. Virgolin, A. Bouter, D. Thierens, and P. A. N. Bosman. "Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms". In: *Evolutionary Computation* (June 2023), pp. 1–28.

9. **A. Dushatskiy**, P. A. N. Bosman, K. A. Hinnen, J. Wiersma, H. Westerveld, B. R. Pieters, and T. Alderliesten. "Deep learning-based segmentation considering observer variation - evaluation in prostate MRI for BT ". In: *Radiotherapy and Oncology* 182:S406-S407 (2023).

8. **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "Heed the noise in performance evaluations in neural architecture search". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM. 2022, pp. 2104–2112.

7. M. M. Bosma, **A. Dushatskiy**, M. Grewal, T. Alderliesten, and P. A. N. Bosman. "Mixed-block neural architecture search for medical image segmentation". In: *Medical Imaging 2022: Image Processing*. Vol. 12032. SPIE. 2022, pp. 193–199.

6. **A. Dushatskiy**, G. Lowe, P. A. N. Bosman, and T. Alderliesten. "Data variation-aware medical image segmentation". In: *Medical Imaging 2022: Image Processing*. Vol. 12032. SPIE. 2022, pp. 759–765.

5. **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "A novel approach to designing surrogate-assisted genetic algorithms by combining efficient learning of Walsh coefficients and dependencies". In: *ACM Transactions on Evolutionary Learning and Optimization* 1.2 (2021), pp. 1–23.

4. **A. Dushatskiy**, T. Alderliesten, and P. A. N. Bosman. "A novel surrogate-assisted evolutionary algorithm applied to partition-based ensemble learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2021, pp. 583–591.

3. T. Den Ottelander, **A. Dushatskiy**, M. Virgolin, and P. A. N. Bosman. "Local search is a remarkably strong baseline for neural architecture search". In: *Proceedings of the Evolutionary Multi-Criterion Optimization: 11th International Conference*. Springer. 2021, pp. 465–479.

2. **A. Dushatskiy**, A. M. Mendrik, P. A. N. Bosman, and T. Alderliesten. “Observer variation-aware medical image segmentation by combining deep learning and surrogate-assisted genetic algorithms”. In: *Medical Imaging 2020: Image Processing*. Vol. 11313. SPIE. 2020, pp. 296–306.

1. **A. Dushatskiy**, A. M. Mendrik, T. Alderliesten, and P. A. N. Bosman. “Convolutional neural network surrogate-assisted GOMEA ”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 753–761.

SIKS DISSERTATION SERIES

2016
01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UVA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions

31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TUE), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks

12 Xixi Lu (TUE), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TUE), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction

11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VU), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OUN), Learning visually grounded and multilingual representations

2020
01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TUE), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation-Methods for Long-Tail Entity Recognition Models
12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context

	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
	31	Gongjin Lan (VU), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
	18	Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
	19	Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
	20	Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems

21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (LIACS), Robust rules for prediction and description
24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

---

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research
17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation

20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
25 Anna L.D. Latour (LU), Optimal decision-making under constraints and uncertainty
26 Anne Dirkson (LU), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

---

2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques

10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals