Parameterless Gene-Pool Optimal Mixing Evolutionary Algorithms

Dushatskiy, Arkadiy; Virgolin, Marco; Bouter, Anton; Thierens, Dirk; Bosman, Peter A.N.

DOI
10.1162/evco_a_00338

Publication date
2024

Published in
Evolutionary computation

Citation (APA)
Dushatskiy, A., Virgolin, M., Bouter, A., Thierens, D., & Bosman, P. A. N. (2024). Parameterless Gene-Pool Optimal Mixing Evolutionary Algorithms. *Evolutionary computation*, *32*(4), 371-397. https://doi.org/10.1162/evco_a_00338

Parameterless Gene-Pool Optimal Mixing Evolutionary Algorithms

Arkadiy Dushatskiy arkadiy.dushatskiy@cwi.nl
Marco Virgolin marco.virgolin@cwi.nl
Anton Bouter anton.bouter@cwi.nl
Evolutionary Intelligence Group, Centrum Wiskunde and Informatica, Amsterdam, 1098XG, the Netherlands

Dirk Thierens d.thierens@uu.nl
Department of Information and Computing, Utrecht University, Utrecht, 3584CC, the Netherlands

Peter A. N. Bosman peter.bosman@cwi.nl
Evolutionary Intelligence Group, Centrum Wiskunde and Informatica, Amsterdam, 1098XG, the Netherlands; Department of Software Technology, Delft University of Technology, Delft, 2628XE, the Netherlands

https://doi.org/10.1162/evco_a_00338

Abstract

When it comes to solving optimization problems with evolutionary algorithms (EAs) in a reliable and scalable manner, detecting and exploiting linkage information, that is, dependencies between variables, can be key. In this paper, 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 several 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 nine black-box problems that can be solved efficiently only 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 GOMEA and CGOMEA significantly outperform the original GOMEA and DSMGA-II on most problems, setting a new state of the art for the field.

Keywords

Model-based evolutionary algorithms, linkage learning, optimal mixing, estimation-of-distribution algorithms, genetic algorithms.

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

Manuscript received: 17 November 2021; revised: 17 February 2023 and 22 May 2023; accepted: 23 June 2023.

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, but 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 long been 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 (Thierens, 1999; Pelikan et al., 1999). Especially in the domain of discrete variables that constitute a Cartesian search space, which is also the domain that this paper pertains to, many EAs employ a mixing operator that exchanges parts of solutions. Ensuring this mixing operator is linkage-friendly, that is, has a high probability of exchanging groups of genes that are highly dependent, can make the difference between obtaining efficient (low-polynomial) and inefficient (exponential) scale-up of the required running time to solve the problem (Thierens, 1999; Pelikan et al., 1999).

The relevance and importance of linkage processing is 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, including 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 (Wolpert and Macready, 1997). 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 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; that is, 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 (Thierens, 1999). 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, that is, fitness, evaluations. This process is commonly known as *linkage learning*, which is a key concept in this paper.

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 (Hart et al., 2005). 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, that is, 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 being 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., Pelikan and Goldberg, 2006; Bosman and Thierens, 2012a). 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) (Larrañaga and Lozano, 2001). 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 paper 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, for example, processing a hierarchy of weak and strong dependencies.

The main contribution of this paper can be summarized as presentation of the parameterless EA showing state-of-the-art performance in the field of discrete BBO. This paper joins all algorithmic information from our previous work that is needed to make this paper 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 new and old versions of GOMEA are compared and shown in comparison with other EAs, including the recent version of DSMGA-II (Chen et al., 2017) and the parameter-less population pyramid (P3) (Goldman and Punch, 2015). 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 paper is organized as follows. In Section 2, we discuss related work. In Section 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 4, followed by the results in Section 5. The paper ends with a discussion in Section 6 and conclusions in Section 7.

2 From Genetic Algorithms to Estimation of Distribution Algorithms 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 (Holland et al., 1992), that is, 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.

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 from. 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, nonoverlapping, deceptive trap functions (Deb and Goldberg, 1993).

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 (Kargupta, 1996). Algorithms in this family allowed genes to be reordered by explicitly encoding their location. Although eventual algorithms were able to avoid exponential scale-up, the overhead of reordering 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 typically reminiscent of genetic algorithms. This type of algorithm is known as the estimation-of-distribution algorithm (EDA) (Larrañaga and Lozano, 2001; Lozano et al., 2006). Effectively and efficiently estimating probability distributions that capture higher-order dependencies was still key, however, to avoid exponential scale-up on problems with nontrivial dependency structure. Initial attempts that used either univariately factorized probability distributions (e.g., PBIL, Baluja and Caruana, 1995, and cGA, Harik et al., 1999) that modeled every variable to be independent from every other variable, or bivariately factorized distributions that considered variable dependencies of at most order two (e.g., MIMIC, De Bonet et al., 1997; COMIT, Baluja and Davies, 1997; and BMDA, Pelikan and Mühlenbein, 1999) still fail to obtain polynomial

scale-up on the additively decomposable deceptive trap functions. Low-order polynomial scale-up is obtained only by EDAs that model higher-order dependencies (e.g., ECGA, Harik et al., 2006; LFDA, Mühlenbein and Mahnig, 1999; and BOA, Pelikan et al., 1999).

The most advanced EDA in this line is commonly accepted to be the hierarchical Bayesian optimization algorithm (hBOA) (Pelikan and Goldberg, 2006). 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 it now satisfactorily solves the linkage problem and provides 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 $\Theta(\sqrt{\ell})$ (Pelikan, 2005). The proofs for different EDAs, for example, UMDA (Mühlenbein and Paass, 1996), are provided by Doerr and Neumann (2019).

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, that is, 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 a basis of the GOMEA framework. GOMEA was first introduced in 2011 (Bosman and Thierens, 2012a), posed as a broadened scope of the idea behind the original linkage tree genetic algorithm (LTGA) introduced in 2010 (Thierens, 2010). 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 led 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 variation, further reducing the overall required model-building complexity. As later versions of LTGA, including the one presented in this paper, 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

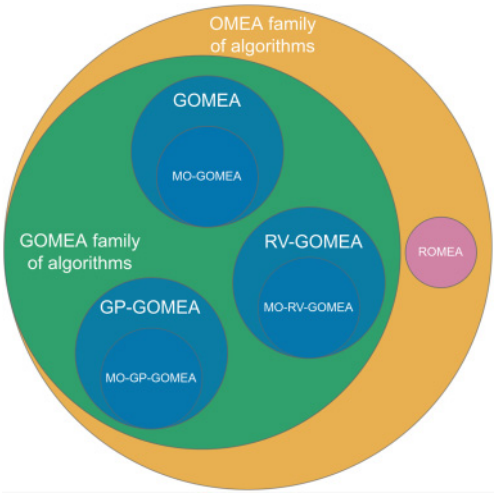

Figure 1: The set diagram of the key algorithms in the OMEA family of algorithms. ROMEA refers to the recombinative OMEA; (MO-)GOMEA refers to the (multiobjective) discrete GOMEA; (MO-)RV-GOMEA refers to the (multiobjective) real-valued GOMEA; (MO-)GP-GOMEA refers to (multiobjective) GOMEA for genetic programming.

linkage information have recently been proposed (Hsu and Yu, 2015; Chen et al., 2017; Goldman and Punch, 2015). These algorithms that can more generally be described as model-based EAs have also been successful at outperforming hBOA.

3 The GOMEA Family of EAs

The family of GOMEAs has been proved to show impressive performance on benchmarks and, importantly, real-world problems. For instance, Real-Valued Multi-Objective GOMEA (RV-MO-GOMEA) (Bouter et al., 2019) is now used for brachytherapy treatment planning optimization. This application received a Silver Humies award (Goodman, 2020), which highlights its practical value and outstanding, better-than-human performance. Another example is an adaptation of GOMEA for genetic programming (GP-GOMEA) (Virgolin et al., 2017). 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 (Virgolin et al., 2020). 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 (Bosman and Thierens, 2012a). Another subset is the recombinative OMEA (ROMEA) (Thierens and Bosman, 2011) family whereby mixing of solutions occurs only between two 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 (Bosman and Thierens, 2012a). For this reason, we focus particularly on GOMEA here. The graphical overview of the GOMEA family of algorithms is shown in Figure 1.

The main idea behind the OMEA framework is that linkages are identified using sets of variable indices (see Section 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 the general class 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 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.

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 $\mathcal{L} = \{0, 1, \ldots, \ell - 1\}$ 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 $\mathcal{L}$. 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 $\mathcal{F}$, that is:

$$\mathcal{F} \subseteq \wp(\mathcal{L}). \tag{1}$$

3.1.1 Linkage Tree (LT) Model

Though different ways to configure a FOS model by learning linkage from the population were introduced, we focus here on a so-called linkage tree (LT) model which demonstrated efficiency in solving various combinatorial optimization problems (Thierens and Bosman, 2011). An LT is a binary tree with $2\ell - 1$ vertices. LT leaves are singletons of problem variables, the root of a 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: $F^i = F^j \cup F^k$, $F^j \cap F^k = \emptyset$.

3.1.2 Similarity Measures

An LT can be built in a bottom-up fashion using hierarchical clustering (Kraskov and Grassberger, 2009): 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 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 (Bosman and Thierens, 2012b). Here, we consider two of them that are most commonly used (e.g., Luong et al., 2018; den Besten et al., 2016; Goldman and Punch, 2015), 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.

3.1.3 Linkage Tree Filtering

It was shown in Bosman and Thierens (2013) that a full LT model (with $2\ell - 1$ vertices) may have redundant subsets that can be filtered out to increase mixing efficiency. Here we consider one particular case of filtering that was successfully applied in Goldman and Punch (2015). 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:

$$\forall F^i, F^j, F^k \in \mathcal{F} \text{ such that } F^i = F^j \cup F^k,$$
$$S(F^j, F^k) \leq 1 - \varepsilon. \tag{2}$$

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 pseudocode in Algorithm 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 (Thierens and Bosman, 2011) and ascending order of subsets size ($|F^i|$) (Goldman and Punch, 2015). 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, that is, 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.

Algorithm 1: GOM and the proposed CGOM operators. Lines in green font color are used in CGOM only.

```
Function (C)GOM(s, P, F, G, useEDS, useFI):
    input : current solution s, population P (of size n), FOS F, dependency graph G,
            hyperparameters useEDS, useFI
    output: evolved offspring o
    b ← o ← s // b is a backup solution
    changed ← false
    U ← ∅
    F ← orderFOS(F) // e.g., random permutation of FOS elements F^i
    for i ∈ {0, 1, ..., |F| − 1} do
        donorsList = randomPermutation({P_0, P_1, ..., P_{n−1}}) // P_i is the i-th
            individual
        for j ∈ {0, 1, ..., n − 1} do
            U ← U ∪ F^i
            d ← donorsList_j
            if ¬checkDonor(o, d, G, U) then
                continue
            o_{F^i} ← d_{F^i}
            if o_{F^i} ≠ d_{F^i} then
                evaluateAndUpdateElitist(o) // get fitness of o, update
                    elitist if necessary
                if (o ≠ elitist and o.fitness ≥ b.fitness) or (o = elitist and o.fitness > b.fitness) then
                    b_{F^i} ← o_{F^i}
                    changed ← true
                else o_{F^i} ← b_{F^i}
                break
            if ¬useEDS then break
    if useFI and (¬changed or o.NIS > 1 + log_10(n)) then FI (s, F, G, useEDS) // o.NIS
        is a counter of non-improving GOM iterations for this solution
        ("No Improvement Stretches")

    if o.fitness ≤ s.fitness then o.NIS ← o.NIS + 1
    else o.NIS ← o
    return o
Function FI(s, F, G, useEDS):
    input : current solution s, FOS F, dependency graph G, hyperparameter useEDS
    output: evolved offspring o
    b ← o ← s // b is a backup solution
    changed ← false
    U ← ∅
    F ← orderFOS(F) // e.g., random permutation of FOS elements F^i
    for i ∈ {0, 1, ..., |F| − 1} do
        U ← U ∪ F^i
        if ¬checkDonor(o, elitist, G, U) then continue
        if o_{F^i} ≠ elitist_{F^i} then
            o_{F^i} ← elitist_{F^i}
            evaluateAndUpdateElitist (o) // get fitness of o, update elitist
                if necessary
            if o.fitness > b.fitness then
                changed ← true
                break
            else o_{F^i} ← b_{F^i}
        if ¬useEDS then break
    if ¬changed then o ← elitist
    return o
Function checkDonor(o, d, G, U):
    input : offspring o, donor d, dependency graph G, currently used varaibles set U
    output: decision (false/true) if the donor can be used
    /* G(v) are variables linked with F^i                                    */
    for p ∈ G(v) ∩ U do
        if o_p ≠ d_p // variable p in o and d
            then return false
    return true
```

3.2.1 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. This modification is called *exhaustive donor search (EDS)*, following Goldman and Punch (2015).

3.2.2 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 Bosman and Thierens (2012a) to deal with convergence issues in MAXCUT. Namely, it can happen that the population starts to drift in fitness plateaus; that is, 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 is the overwrite action accepted. To prevent the FI phase from reducing 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.

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; that is, 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 mutual exclusive subsets. That is, consider the NK-landscapes (Pelikan et al., 2009) 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 to and inspired by recently introduced conditional linkage models for the real-valued GOMEA (RV-GOMEA) (Bouter et al., 2020). 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 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; that is, 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

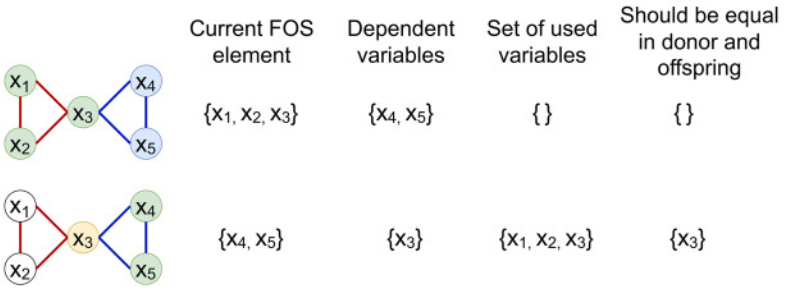

Figure 2: A minimal CGOM working example. Fitness function f is a function of five variables and can be decomposed into two subfunctions: $f(x_1, x_2, x_3, x_4, x_5) = f_1(x_1, x_2, x_3) + f_2(x_3, x_4, x_5)$. This is shown by the different colors of edges. Suppose that after filtering the LT FOS contains two elements: $\{x_1, x_2, x_3\}$ and $\{x_4, x_5\}$. Variables from the current FOS element are colored in green; variables which are dependent with it are colored in blue; variables which are dependent with it and are already used (i.e., are taken into account by CGOM) are colored in yellow. Green-colored genes are pooled to an offspring from only those donors which have the yellow-colored genes equal to what is found in the current offspring.

set of variables V such that (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 genes 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. The CGOM differences as compared to GOM in terms of pseudocode are highlighted in Algorithm 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, that is, 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. The hyperparameter λ is tunable; its range of possible

[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. The average value is intuitive, easy to use, and less sensitive to outliers.

Algorithm 2: 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 $\mathcal{P}$
  $\mathcal{P} \leftarrow$ `createPopulation` (n, *useHC*)
  while $\neg terminationCriterionSatisfied$ **do**
    $\mathcal{P} \leftarrow$ `doOneGeneration`($\mathcal{P}$)
  return $\mathcal{P}$

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

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

Function `learnDependencies` ($\mathcal{F}, \mathcal{P}$) **:**
  input : FOS $\mathcal{F}$, population $\mathcal{P}$
  output: dependency graph $\mathcal{G}$
  $\mathcal{G} \leftarrow \{\emptyset\}_{i=0}^{|\mathcal{F}|-1}$
  $\mathcal{S} \leftarrow$ `calculateSimilarityMatrix`($\mathcal{P}$) `// e.g., Mutual Information (pairwise)`
  for $i \in \{0, 1, \ldots, \mathcal{F}-1\}$ **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**
        $\mathcal{G}^i \leftarrow \mathcal{G}^i \cup \{j\}$
  return $\mathcal{G}$

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).

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.

3.4.1 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, Ochoa et al., 2010) works by flipping bits of a solution in random order and greedily accepting improving changes. EHC (also called *best-improvement local search*, Ochoa et al., 2010) 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 Hsu and Yu (2015), and EHC was used in Goldman and Punch (2015). The pseudocode for considered hill climber algorithms is listed in Supplementary Algorithm 1.

3.4.2 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 Hsu and Yu (2015) and Chen et al. (2017). We remark that with this option, the selection is disregarded after the linkage model is learned; that is, 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 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.

3.5 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 nontrivial. Methods to scale the population size automatically over time are therefore extremely useful and convenient in practice. In this paper, we consider two well-known population size-free schemes.

First, we consider the interleaved multistart scheme (IMS), which was heavily inspired from the work by Harik and Lobo (1999) on parameterless GAs. The IMS has been shown to be easy to use and can be naturally applied to almost any EA in various optimization domains (Luong et al., 2018; Lin and Yu, 2018; Dushatskiy et al., 2019; Virgolin et al., 2017).

The IMS consists of evolving multiple populations simultaneously, in an interleaved fashion. Its pseudocode with recursive implementation is listed in Supplementary Algorithm 2. In the beginning, a single population is initialized, typically of a very small size (e.g., 2). After $\mathcal{M}_{IMS}$ 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 $\mathcal{M}_{IMS}$ generations more. When the larger population has performed $\mathcal{M}_{IMS}$ generations, an even larger population is initialized, and so on. Our implementation of the IMS uses an initial population of size 2,

Algorithm 3: 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
```

exponential growth whereby each new population is twice the size of the previous, and $\mathcal{M}_{IMS} = 4$, as in Harik and Lobo (1999). 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.

The other schemes that we consider are the parameterless population pyramid (P3) (Goldman and Punch, 2015), and its further modification, multiple insertion pyramid (P3-MI) (den Besten et al., 2016). The difference between the two is explained below and pseudocode is provided in Algorithm 3.

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, offsprings 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. 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 Goldman and Punch (2015), different population growth functions were studied. In this work, 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 value 1; in other words, one new solution is created and evolved in each iteration.

4 Experiments

4.1 Benchmark Problems

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

4.1.1 Concatenated Deceptive Traps

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 $s = 4$ ($Trap_5^4$).

4.1.2 Bimodal Separable Deceptive Trap

The bimodal symmetric concatenated trap functions (Deb et al., 1993) 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 nonoverlapping subfunctions are given by

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

4.1.3 NK-Landscapes

The NK-landscapes with maximum overlap (also called NK-S1 landscapes, Pelikan et al., 2009) with subfunctions of size $k = 5$ are interesting because of overlapping subfunctions which are different depending on the position in the 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 subfunction positions.

4.1.4 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:

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

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

4.1.5 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) and x_i, x_j are solution values in the corresponding positions; that is, 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 four neighbors, forming a torus. Edge 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 instance is further referred to as MAXCUT Dense, and they are known to be NP-hard problems. For MAXCUT Dense, we use edge weights values in [0, 1000] sampled uniformly.

4.1.6 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 spin-glass instances we used, each variable interacts with up to four neighbors in a 2D grid.

4.1.7 MAXSAT

Finally, we consider the MAXSAT problem. Particularly, we consider unweighted MAX-3SAT uniform random instances (Chen et al., 2017). 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 a 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 $\approx 4.3\ell$.

Table 1: Considered hyperparameters of single-population GOMEA. In bold, the best settings found by the experiment described in Section 4.3.

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

4.2 Sizes of Problems

We frame our experiments in terms of scalability; that is, 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.

4.3 Finding the Best Settings for Single-Population GOMEA

We summarize different possible choices of single-population GOMEA components in Table 1. Because 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.

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 each 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 Chen et al., 2017) to prevent convergence problems. As it might happen that the smallest population size that allows for solution of a problem instance does not require the fewest function evaluations, during the bisection procedure we keep track of the population size that allows for solution of 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 GOMEA version is further referred to as $GOMEA^{best}$.

4.4 Adding 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 threhsold 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 4.3. This best performing CGOMEA version is further referred to as $CGOMEA^{best}$.

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 (Thierens and Bosman, 2011), and against the most recent single-population DSMGA-II version (Chen et al., 2017). The optimal population sizes for all algorithms are determined using bisection.

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 as described in Section 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 that 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 and DSMGA-II with IMS (Lin and Yu, 2018). 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 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.

4.7 Statistical Testing

To test the statistical significance of performance differences between algorithms, we use the two-step approach following Derrac et al. (2011): first, we use the 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). Signficance level is set to 0.05.

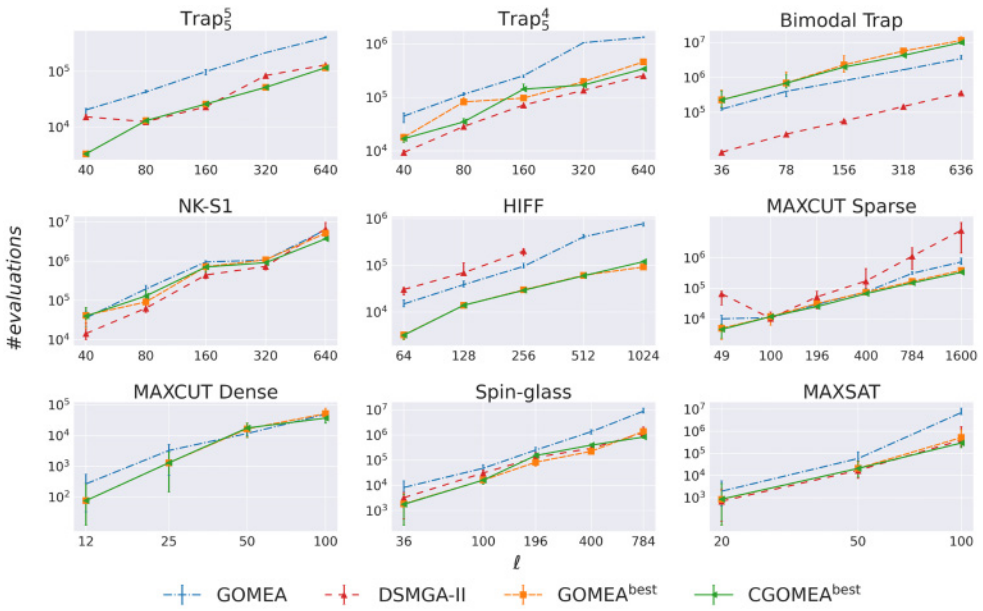

Figure 3: Scalability of single-population EAs in terms of function evaluations required to find an optimum. Points show median values of 50 runs. Bars show 3rd and 48th order statistics (92% confidence interval). If an algorithm fails to find the global optimum of a problem instance in all 50 runs, the corresponding point is not shown. GOMEA refers to the previously published version (Bosman and Thierens, 2012a).

4.8 Implementation Details

All GOMEA variants and the P3 algorithm are implemented in C++.[2] The 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.

5 Results

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* 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 1.

With the best hyperparameter settings, *GOMEA^{best}* has better performance than the previously published GOMEA version on seven out of nine considered problems. These results are shown in Table 4, and scalability plots are presented in Figure 3. On the MAXSAT and HIFF problems the improvement is approximately of an order of mag-

[2]Source code is available at: https://github.com/ArkadiyD/BinaryGOMEA

[3]https://github.com/brianwgoldman/FastEfficientP3/

[4]https://github.com/tianliyu/DSMGA-II-TwoEdge

nitude. Two 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 trap 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 nontrivial problems efficiently and it is less likely to fail to solve a problem. However, we see that in four out of nine 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 ($p\text{-}value = 0.03$). P-values for all comparisons are provided in Supplementary Table 1.

Although 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 aggregated performance of all algorithms that use this design choice regardless of all other options they use. These results are shown in Supplementary Figure 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 the 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 simpler hill climber and no hill climber at all. This is in line with earlier reported results (Bosman and Thierens, 2011). Using the filtered linkage tree built with the normalized mutual information measure slightly improves the performance on some problems from the benchmark set, although 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 of 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.

5.2 CGOMEA Performance

We take the 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 3. 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 the 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 that provides the best average 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.

Table 2: Considered hyperparameters of parameterless GOMEA. In bold, the best settings found by the experiment described in Section 4.6.

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

		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

As shown in Table 4 and in scalability plots in Figure 3, $CGOMEA^{best}$ outperforms $GOMEA^{best}$ on seven out of nine considered problems. On $Trap_5^5$ $CGOMEA^{best}$ perform on par with $GOMEA^{best}$. Only on the HIFF problem does CGOM perform slightly worse. Moreover, $CGOMEA^{best}$ performs better than DSMGA-II on five problems, and there are two problems (HIFF and MAXCUT Dense) that 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 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 Supplementary Figure 2. 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 is CGOMEA 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.

Table 4: Results of single-population (left table) and parameterless (right table) EAs. For single-population EAs, best population sizes are found with bisection. Ranking per problem shown through color gradient from green (best, i.e., the fewest median number of function evaluations) to red (worst, i.e., 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: $\mathbf{G} = \mathbf{GOMEA}$; $\mathbf{D{-}II} = \mathbf{DSMGA{-}II}$; $\mathbf{G^B} = \mathbf{GOMEA^{BEST}}$; $\mathbf{CG^B} = \mathbf{CGOMEA^{BEST}}$; $\mathbf{P3} = \mathbf{P3}$; $\mathbf{GP3^B} = \mathbf{GOMEA{-}P3^{BEST}}$; $\mathbf{CGP3^B} = \mathbf{CGOMEA{-}P3^{BEST}}$.

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

5.3 Parameterless EAs

Results of experiments with parameterless EAs are presented in Table 4 and in scalability plots in Figure 4. 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* based on *normalized mutual information*, *randomly shuffled FOS*, and *P3* scheme, as highlighted in Table 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 recalculated from scratch (Goldman and Punch, 2015). 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 4.3), is further referred to as *CGOMEA-P3^{best}*.

First, we see that DSMGA-II with IMS scheme was not capable of solving all problems in the experimental setup due to its issues with time efficiency.

CGOMEA-P3 ^{best} performs better than *GOMEA-P3^{best}* on five problems out of nine. 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 Sparse are subtle. Compared to the P3 algorithm (Goldman and Punch, 2015) *CGOMEA-P3^{best}* performs better on all problems except HIFF. *GOMEA-P3^{best}* performs better than P3 on six problems.

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.

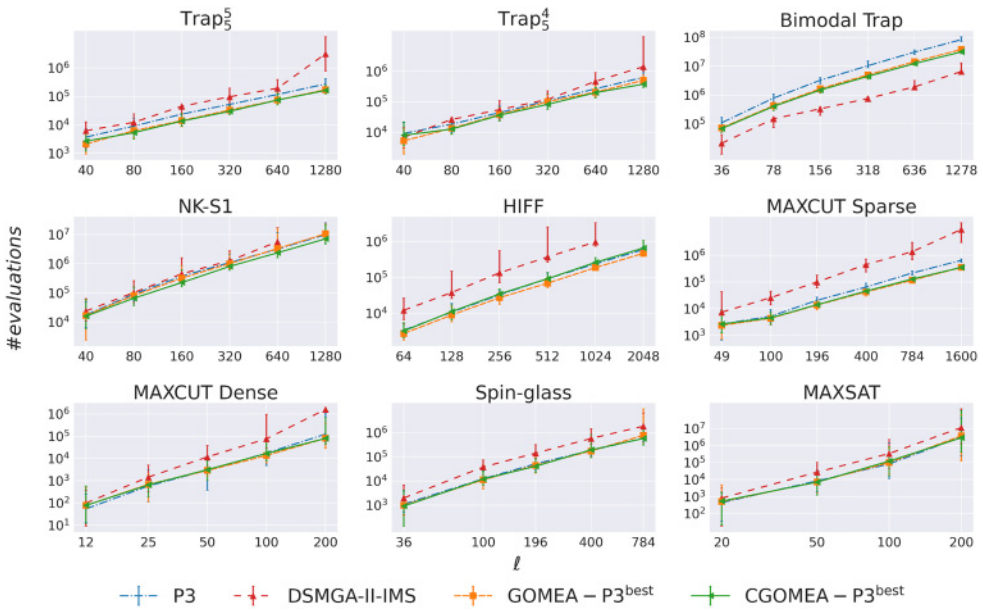

Figure 4: Scalability of parameterless EAs in terms of function evaluations required to find an optimum. Points show median values of 50 runs (48 runs for MAXSAT problem with $\ell = 200$). Bars show 3rd and 48th (46th for MAXSAT problem) order statistics (92% confidence interval). If an algorithm fails to find the global optimum of a problem instance in all 50 runs (48 runs for MAXSAT problem), the corresponding point is not shown.

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 Supplementary Table 2.

As shown in Supplementary Figure 3, P3 versions of GOMEA and CGOMEA scale similarly to P3, though they are slower. Scalability in terms of 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.

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 which are actually independent and missing some truly existing ones. Potentially, a more accurate approach to learning dependencies can further improve CGOM performance. Moreover, it may be interesting to apply the 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 a direct and concise modelling 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 specifically designed for GBO, such as Chicano et al. (2017).

The considered model-based EAs relied on entropy-based information measures to learn dependencies between variables. We notice that the performance on Bimodal Trap can 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 (Przewozniczek and Komarnicki, 2020; Dushatskiy et al., 2021). 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 paper, 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. Although this benchmark set includes well-known combinatorial optimization problems, problems arising in practical tasks may have properties (such as fitness landscape and dependencies structure) which are very different from all common benchmark functions. Although 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 an active field of research (van der Blom et al., 2020). We hypothesize, however, that the state-of-the-art benchmark problems in the field of EAs for binary optimization that we used are 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 keep 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 that utilizes graphical processing units (GPUs) has been added to the real-valued GOMEA for the GBO case (Bouter and Bosman, 2022). 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.

7 Conclusion

In this paper, 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 nine benchmark problems considered diverse and non-trivial in a single-population EA experimental setup where we assess scalability of the algorithms of required resources 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 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 program Commit2Data with 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

Baluja, S., and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In *Proceedings of Machine Learning*, pp. 38–46.

Baluja, S., and Davies, S. (1997). *Combining multiple optimization runs with optimal dependency trees*. Technical Report. Carnegie-Mellon University, Pittsburgh PA, Department of Computer Science.

Bosman, P.A.N., and Thierens, D. (2011). 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*, pp. 663–670.

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

Bosman, P.A.N., and Thierens, D. (2012b). On measures to build linkage trees in LTGA. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pp. 276–285.

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

Bouter, A., Alderliesten, T., Pieters, B. R., Bel, A., Niatsetski, Y., and Bosman, P.A.N. (2019). GPU-accelerated bi-objective treatment planning for prostate high-dose-rate brachytherapy. *Medical Physics*, 46(9):3776–3787. 10.1002/mp.13681

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

Bouter, A., Maree, S. C., Alderliesten, T., and Bosman, P.A.N. (2020). 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*, pp. 603–611.

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

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

De Bonet, J. S., Isbell, C. L., Jr., and Viola, P. A. (1997). MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, pp. 424–430.

Deb, K., and Goldberg, D. E. (1993). Analyzing deception in trap functions. In *Foundations of genetic algorithms*, Vol. 2, pp. 93–108. Elsevier.

Deb, K., Horn, J., and Goldberg, D. E. (1993). Multimodal deceptive functions. *Complex Systems*, 7(2):131–154.

den Besten, W., Thierens, D., and Bosman, P.A.N. (2016). The multiple insertion pyramid: A fast parameter-less population scheme. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pp. 48–58.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18. 10.1016/j.swevo.2011.02.002

Doerr, B., and Neumann, F. (Eds.). (2019). *Theory of evolutionary computation: Recent developments in discrete optimization*. Springer.

Dushatskiy, A., Alderliesten, T., and Bosman, P.A.N. (2021). A novel approach to designing surrogate-assisted genetic algorithms by combining efficient learning of Walsh coefficients and dependencies. *ACM Transactions on Evolutionary Learning and Optimization*, 1(2). 10.1145/3453141

Dushatskiy, A., Mendrik, A. M., Alderliesten, T., and Bosman, P.A.N. (2019). Convolutional neural network surrogate-assisted GOMEA. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 753–761.

Goldman, B. W., and Punch, W. F. (2015). Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary Computation*, 23(3):451–479. 10.1162/EVCO_a_00148

Goodman, E. (2020). Human-competitive results awards—"Humies" 2019—announces winners at GECCO. *SIGEVOlution*, 12(3):3–5. 10.1145/3381343.3381344

Harik, G. R., and Lobo, F. G. (1999). A parameter-less genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 258–265.

Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297. 10.1109/4235.797971

Harik, G. R., Lobo, F. G., and Sastry, K. (2006). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In M. Pelikan, K. Sastry, and E. Cantú-Paz (Eds.), *Scalable optimization via probabilistic modeling*, pp. 39–61. Springer.

Hart, W. E., Krasnogor, N., and Smith, J. E. (2005). Memetic evolutionary algorithms. In *Recent advances in memetic algorithms*, pp. 3–27. Springer.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press.

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

Kargupta, H. (1996). The gene expression messy genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 814–819.

Kraskov, A., and Grassberger, P. (2009). MIC: Mutual information based hierarchical clustering. In F. Emmert-Streib and M. Dehmer (Eds.), *Information theory and statistical learning*, pp. 101–123. Springer.

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

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

Lozano, J. A., Larrañaga, P., Inza, I., and Bengoetxea, E. (Eds.). (2006). *Towards a new evolutionary computation: Advances on estimation of distribution algorithms*. Studies in Fuzziness and Soft Computing, Vol. 192. Springer. 10.1007/3-540-32494-1

Luong, N. H., La Poutré, H., and Bosman, P.A.N. (2018). Multi-objective gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start scheme. *Swarm and Evolutionary Computation*, 40:238–254. 10.1016/j.swevo.2018.02.005

Mühlenbein, H., and Mahnig, T. (1999). FDA—A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376. 10.1162/evco.1999.7.4.353

Mühlenbein, H., and Paass, G. (1996). From recombination of genes to the estimation of distributions I. binary parameters. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pp. 178–187.

Ochoa, G., Verel, S., and Tomassini, M. (2010). First-improvement vs. best-improvement local optima networks of NK landscapes. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pp. 104–113.

Pelikan, M. (2005). Hierarchical Bayesian optimization algorithm. In *Hierarchical Bayesian optimization algorithm*, pp. 105–129. Springer.

Pelikan, M., and Goldberg, D. E. (2006). Hierarchical Bayesian optimization algorithm. In M. Pelikan, K. Sastry, and E. Cantú-Paz (Eds.), *Scalable optimization via probabilistic modeling*, pp. 63–90. Springer.

Pelikan, M., Goldberg, D. E., Cantú-Paz, E., et al. (1999). BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 1, pp. 525–532.

Pelikan, M., and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In *Advances in Soft Computing*, pp. 521–535.

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

Przewozniczek, M. W., and Komarnicki, M. M. (2020). Empirical linkage learning. *IEEE Transactions on Evolutionary Computation*, 24(6):1097–1111. 10.1109/TEVC.2020.2985497

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, 7(4):331–352. 10.1162/evco.1999.7.4.331

Thierens, D. (2010). The linkage tree genetic algorithm. In *Proceedings of the International Conference on Parallel Problem Solving from Nature, Part I*, pp. 264–273.

Thierens, D., and Bosman, P.A.N. (2011). Optimal mixing evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 617–624.

van der Blom, K., Deist, T. M., Tušar, T., Marchi, M., Nojima, Y., Oyama, A., Volz, V., and Naujoks, B. (2020). Towards realistic optimization benchmarks: A questionnaire on the properties of real-world problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 293–294.

Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P.A.N. (2017). 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*, pp. 1041–1048.

Virgolin, M., Wang, Z., Alderliesten, T., and Bosman, P.A.N. (2020). Machine learning for the prediction of pseudorealistic pediatric abdominal phantoms for radiation dose reconstruction. *Journal of Medical Imaging*, 7(4):046501. 10.1117/1.JMI.7.4.046501

Wolpert, D. H., and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82. 10.1109/4235.585893