Solving OAS

Solving order acceptance and scheduling with modified black box approaches

A. Guijt

TU Delft

Cover images by NASA, Mike Petrucci, Micheile Henderson, Alexandru Acea and JJ Ying via Unsplash.

Solving OAS

Solving order acceptance and scheduling with modified black box approaches

by

A. Guijt

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday 16th of October at 15:00.

Project duration:	November 12, 2018 – October 16, 2019	
Thesis committee:	Dr. M.M. de Weerdt,	TU Delft, supervisor
	Prof.dr. P.A.N. Bosman,	TU Delft
	Dr. A. Panichella,	TU Delft

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

TU Delft

Preface

While the problem itself was set in stone, what I wanted to add to the pre-existing literature was initially something completely different: a survey of black-box approaches, how to get them to work and how they perform. In the end this turned out a bit differently. Rather than a survey the focus ended up being on a single approach; and rather than a simple how to get it working the end result ended up improving beyond the state-of-the-art for the problem.

Analysing the results and figuring out the hints to where an approach works well and where it does not, was probably one of the most enjoyable parts of my thesis and was the key to obtain the afore-mentioned result. Using the insights to alter the approaches and actually having the results improve was a special cause of excitement.

I would like to especially thank dr. Mathijs de Weerdt for his time, the discussions and the feedback on my thesis. Furthermore I would like thank Prof.dr. Peter A. N. Bosman for the approach I expanded upon. My gratitude extends to both of them as well as dr. Annibale Panichella for being members of my thesis committee.

A special thank you to Lei He, the PhD student that introduced me to the OAS problem and the literature surrounding it, including the paper of the current state-of-the-art approach.

Finally I would like to thank my father and mother and my younger brother for their support and listening ear. Without this support from all of you I probably wouldn't be where I am now.

A. Guijt
Delft, September 2019

Contents

1 Introduction

Most scheduling problems in literature focus on completing all tasks, minimizing a measure related to time. Hall and Sriskandarajah [14] for example list a few objectives: makespan, lateness, idle time, cycle time, and weighted variants.

Many real world applications are however restricted on time. Working and operational hours are often limited, with some operations being expensive or impossible to perform outside of the available time. An example at the Dutch National Railways – the NS – is train shunting scheduling as described by van den Broek [28], a train needs to be checked, maintained and ideally also be cleaned before it can be used to transport passengers again. Hence all of these tasks have to be completed before its scheduled departure time. For such a scenario the problem becomes a decision problem - ”Does a schedule, that can be performed within the available time, exist?”.

Conversely, the real world may be more flexible in other ways. Orders may be rescheduled to a different point in time, or refused entirely for example. And in the case for train shunting, cleaning is not necessarily required. Not all tasks need to be performed within this limited time window, a selection needs to be made. This forms the basis of order acceptance and scheduling.

Time is limited — performing all orders is likely impossible — but profit should be maximized. This problem is a combination of multiple NP-hard problems. In particular, it is a generalization of Knapsack — weight is replaced with the concept of time and its corresponding scheduling problem — it is therefore not surprising that this problem is indeed NP-hard. The problem used as the basis for this problem – Order Acceptance and Scheduling with Sequence-Dependent Setup Times – expands upon this with the addition of sequence-dependent setup times. This makes the problem strongly NP-hard according to Oğuz et al. [20], as a restriction — the problem without selection of orders — is strongly NP-hard according to Lawler [18]. A mapping from Travelling Salesperson to Order Acceptance and Scheduling with Sequence Dependent Setup Times reaffirms this hardness and can be found in Appendix A. In the remainder of this thesis I will refer to this problem as the OAS problem.

Oğuz et al. [20] proposed this approach for make-to-order systems, but similar traits can be found in problems such as in Satellite Scheduling or the Travelling Repairman problem [2]:

The sequence-dependent setup times can for example not only model cleaning and setting up, but also travelling time for the Travelling Repairman problem and time to target the region of interest for Satellite Scheduling. The processing times model the time it takes to perform an action, whether it is performing a repair or taking a photograph. Release times, due dates and deadlines model availability windows, some people may want repairs to be performed in the morning, and a satellite can only take a picture of something in view. Selecting orders is especially important for satellite scheduling due to smaller overlapping time windows, and the impossibility of performing a task after its time window. And in reality a Travelling Repairman has working hours, barring the issue of working in someone’s house while they are absent or asleep.

The OAS problem as proposed by [20] has been the topic of some papers [9, 10, 24], and while they draw inspiration from prior literature for other problems, the initial state-of-the-art was a basic tabu search approach in [9].

A state-of-the-art approach for one problem may not necessarily work well on another problem. The structure of the problem is used to improve performance and this comes at the cost of specificity: while

some assumptions and approaches may be transferable, some may be too problem specific.

An example of this is that of the constructive greedy heuristic for initial solutions. For the OAS problem the greedy heuristic now needs to account for time windows. Applying such an approach to another problem is hence not trivial and requires adaptation, even if the underlying kind of solution — a permutation — is the same.

Conversely black-box approaches exist. As they approach the underlying problem as a black-box they are designed to be generally applicable, and therefore not problem specific. These metaheuristics often include adaptive components, which allow it to optimize a variety of different problems.

Subramanian and Farias have looked into the application of techniques such as ant colony, particle swarm and discrete differential evolution on a related problem – single machine total weighted tardiness scheduling with sequence-dependent setup times – in [26]. The techniques applied are all discrete in nature, with no continuous or specifically adjusted approaches for permutations; and the underlying problem does not deal with selection and hard deadlines.

The use of adaptive metaheuristics that use a model — such as a linkage model — is not commonly applied to the OAS problem in literature, while such metaheuristics have shown to be successful for a variety of problems [27] including scheduling [1, 7].

The use of a black-box approach may not necessarily require significant effort either, due to the availability of open source implementations, for example those of Feldt for Julia programming language [11], or research code. The application of such an approach then involves replacing the black-box with another; this new black-box will then evaluate the solution with respect to this new problem.

Creating a black-box that represents the new problem may however be more difficult. The input the black-box requires may not be the same as that of the approach optimizing the black-box. An approach usually has a fixed type of input it works with. The most common input types are continuous and discrete values, but other kinds of input such as the permutations exist as well. If there is a mismatch between black-box and approach — as is the case between the most common forms of input and the OAS problem, which requires a permutation — a compatibility layer – encoding – needs to be introduced to remedy this mismatch.

Given the complexity of the OAS problem and the general success of applying black box techniques to such complex problems and related problems, there is a distinct lack of investigation into the application of black-box approaches to the OAS problem itself. Continuous approaches have not been tried, yet could very well work for the OAS problem as keys correspond to their ordering in time. Furthermore, 'learning' approaches have not been tried either and show notable improvements when applied as well.

The current benchmark set of instances for the OAS problem is not yet solved to optimality. For many instances there still exists a gap between what metaheuristics can accomplish [10] and the upper bounds found by exact methods [24]. Given the complexity of the OAS problem, as well as the improvements obtained with advanced black-box approaches, the aim of this thesis is to utilize these approaches to obtain such improvements to close the gap between the quality of solutions obtained through a metaheuristic and upper bounds.

1.1. Research Questions

The goal of this thesis is to answer the question:

> **Can black-box approaches be utilized to obtain better performing metaheuristics for the Order Acceptance and Scheduling with Sequence Dependent Setup Times Problem?**

This question can be separated into a select few sub-questions:

- **How can instances for the OAS problem be encoded such that black-box approaches can solve them?**

- **What black-box approaches can be used?**
  There are too many techniques to cover (and analyse) in literature. As such a selection of techniques are described in Section 2.4. These techniques were selected on the basis on availability, preferably open source and well tested implementations, general performance on other problems and variety.

- **What instances warrant inclusion for testing?**
  Apart from the standard benchmark set for the OAS problem, many more real-world situations are subproblems. More benchmark sets — that are different from each other — are needed to assess performance in more cases. How does each approach perform in various realistic situations? How unrealistic do instances need to be before an approach starts performing badly?
- **What aspects of an approach make it perform well?**
- **And what aspects of an approach make it perform *badly*?**
- **How can current approaches be significantly improved upon?**
  For example by fixing the aforementioned issues that make an approach perform badly.
- **How well do these techniques perform compared against prior art from literature?**

1.2. Thesis Outline

Chapter 2 of the thesis will explain general terms, background, and the black box approaches used within this thesis. Chapter 3 will proceed to express the benchmark instances and the experimental setup related to these instances. Chapter 4 contains the results of these experiments and an analysis is performed, showcasing the strong and weak points of each approach. Chapter 5 takes these results and uses them to improve the aforementioned approaches further. This thesis ends in Chapter 6 with some concluding remarks and future work.

2

Background and Definitions

This chapter provides the definition of the OAS problem, the topic of this thesis, the approaches used within this thesis, as well as the required transformations to make them work on the problem itself. Definitions for important common terms such as black-box optimization and the quality metric used are stated. Furthermore some additional background is provided for the OAS problem and black-box solving approaches in general, with special attention for permutation problems.

Among the approaches used is a white-box approach that is the current state-of-the-art. As comparing a white-box approach, with access to the internals of a function, to a black-box approach without such access is difficult — the latter approach has a significant disadvantage — some information is also provided through the use of hints to close this information gap. These hints are based on literature, and make use of transforming the objective function by including completion time — as used in the state-of-the-art — and using the release times and deadlines for better initial solution generation.

2.1. Problem Definition

An instance of OAS is characterized by a number n of orders i. Each order has a release time r_i, a processing time p_i, and a due date d_i which represents the time after which a penalty is incurred. Additionally orders have a reward for completion e_i, and a penalty of being late w_i, together with the due date this specifies a deadline $\bar{d}_i = d_i + \frac{e_i}{w_i}$. The due date is the point after which the reward for completing the order is reduced: after d_i you lose w_i out of e_i profit per time unit leaving $\frac{e_i}{w_i}$ time units until all profit is lost. Finally every order has a sequence-dependent setup time s_{ji}, which starts after the release time of an order i and is dependent on the preceding order j. This preceding order is possibly the fake order o or 0: the setup time is then incurred for being the first order to be performed.

Assuming all orders are performed before their deadline, given a sequence of orders π, the completion time of an order at position i is then defined inductively:

$$t_i(\pi) = \max(r_{\pi(i)}, t_{i-1}) + s_{(\pi(i-1),\pi(i))} + p_{\pi(i)} \tag{2.1}$$

The objective is then to find the sequence of orders π that maximizes the total profit f:

$$f(\pi) = \sum_{i=1}^{|\pi|} e_{\pi(i)} - max(0, t_i - d_{\pi(i)}) \tag{2.2}$$

A few solutions were proposed in literature. A MIP model was proposed in [20], a Tabu Search metaheuristic was proposed in [9]. A Lagrangian relaxation and a column generation approach were proposed in [24], together with an Iterated Local Search metaheuristic. Finally a mixture of a genetic algorithm and local search called HSSGA was proposed in [10]. In Section 2.4 the analysed metaheuristic approaches will be discussed in more detail.

2.2. Black Box Optimization

Black Box Optimization is a general method for optimizing functions. The general task is find the parameters x_i that maximize or minimise the objective given only the ability to call a function $f(x_1, \ldots, x_n)$ that returns the objective value. This function f is also referred to the black box, and can be any computational procedure.

Unlike methods which require the gradient of a function — such as Gradient Ascent — or a metaheuristic for a single problem — such as Lin-Kernighan [19] for the Traveling Salesperson Problem — black-box techniques are quite general. This generality allows for a wide variety of applications; there is no problem the technique itself is coupled to. This generality also comes at a cost: the performance of a black box optimizer depends significantly on the black box in question. Dimensionality, time-budget and properties of the function itself such as smoothness affect or the kind of structure the performance of the optimizer [16]. Therefore care has to be taken with both designing a function that encodes the problem that you want to optimize, and the choice of an approach to optimize the function in the first place (including parameters!).

2.3. Encoding

In order to be able to use black-box approaches to optimize solutions, we first need to be able to encode solutions themselves in a convenient form first.

Implicit Selection The first paper to propose a metaheuristic – Tabu Search – used permutations rather than encoding a solution as a variable length sequence [9]. By ignoring an order that ends up completing past their deadline the selection itself is implicit in the ordering itself, without losing the ability to encode the optimal solution.

This first step allows encoding the solution as a permutation, and the key step that allows approaches operating directly on permutations such as the current state-of-the-art approach for the OAS problem — HSSGA — to work.

Encoding Permutations Yet encoding as a permutation is not sufficient to allow black-box approaches that expect continuous input to work.

Similar to solving a permutation problem by approaching it as a discrete optimization problem, naively encoding the permutation by ceiling floats is not an option. The chance that a random selection of these numbers ends up encoding a valid permutation with no duplicates is very small. This probability is given in Formula 2.3 and tends towards zero as n increases.

$$P(\text{valid permutation} \mid \text{naive}) = \prod_{i=1}^{n} \frac{n-i}{n} = \prod_{i=1}^{n} \frac{i}{n} = \frac{n!}{n^n} \tag{2.3}$$

Duplicate orders do also occur in the use of the standard local search, mutation and crossover operations, requiring the use of specialized operators such as those defined by Chaurasia in [10].

Random Keys Rather than encoding the permutation naively, the approach used in literature uses the concept of “Random Keys” [5]. The vector in this case consists of n values in $[0, 1]$, the encoded permutation is then the order of indices that sorts this vector. For example $[0.0, 0.9, 0.4, 0.6]$ encodes the permutation $[1, 3, 4, 2]$.

This approach is relatively simple: the decoding procedure is finding the permutation that sorts the list. This can be done through application of a sorting routine where indices are sorted by their corresponding keys.

Another nice trait is that every order is encoded in exactly the same manner, the meaning of a particular value does not change from order to order.

This encoding was already used within Permutation GOMEA — one of the methods to be discussed — in order to be able to mix permutations without issues, and now also used to make other approaches such as Generating Set Search and Adaptive Differential Evolution applicable to the OAS problem.

2.4. Solvers

There exist many black-box approaches for solving both discrete and continuous problems. As stated in Chapter 1, Subramanian and Farias have already investigated some standard approaches such as ant-colony and particle swarm optimization, as well as a discrete differential evolution approach [26] on a problem related to OAS. Kolda [17] discusses optimization by direct search in general, alongside some nice properties of such an approach. Differential Evolution [25] is one of the more well known continuous optimization techniques with many variants [22, 30] developed to deal with the importance of parameter configuration. Other approaches include approaches based on and/or in the same spirit as Evolution Strategies [6] such as Natural Evolution Strategies [29] which uses a gradient and CMA-ES [15].

Approaches specifically meant for optimizing black-boxes with permutations as input are harder to find. Most approaches take already existing black-box approaches and encode the permutation itself — for example using random keys, as explained in Section 2.3. One such approach is Permutation GOMEA [7], an adaptive approach based of the discrete version of GOMEA [27] with some modifications to make it work better on permutation problems.

Specifically targeting permutations can yield notable performance improvements according to Campos, Laguna and Martí [8], they show that their black-box approach for permutation problems outperformed even commercial implementations at the time.

This section describes the selected black-box optimizers, (modified) specialized metaheuristics and a simple baseline approach.

Some other approaches were considered but were left out due to their inability to perform well, even on smaller problems. For example a Natural Evolution Strategies [6, 11] approach almost always prematurely converged on a solution that was far off the optimum, even on smaller problems. This is possibly due to the estimated gradient of the encoded problem being zero in almost all cases. While a gradient of zero is a marker for an optimum for most continuous functions, this is not the case for this encoded function: a small delta in all likelihood does not change the underlying permutation, and the resulting fitness hence stays constant.

The approaches after the selection procedure are varied: both population-based, and single-solution-based search approaches are represented, as well as the state-of-the-art and various general black-box approaches.

2.4.1. Generating Set Search

Generating Set Search is a variant on Compass Search where the set compass directions is required to be a generating set [17]. The implementation used is provided by the BlackBoxOptim.jl package by Feldt [11] for the Julia programming language.

GSS bears a resemblance to simulated annealing: searching is done by taking steps with a certain step size. This step size roughly corresponds to the idea of temperature, high in the beginning and decreasing over time. Step size starts at a default value — usually half the diameter of the search space — and increases upon a successful improvement by a factor of Γ and when no improvement is made the step size decreases by a factor of Φ again.

The directions in which steps are taken are required to form a generating set $\mathcal{G}$: the entire search space $\mathcal{S}$ can be reached by combining the directions with positive weights.

Each direction $\vec{d} \in \mathcal{G}$ is then combined with the current step size δ to obtain the step that needs to be added to the current solution $\vec{s}$. This solution is then evaluated using the objective function or black-box f. If the new solution is better[1] — $f(\vec{s}+\delta\vec{d}) < f(\vec{s})$ — then the solution is replaced, otherwise the solution remains the same.

When the step size has decreased beyond that of a minimum Δmax, the algorithm restarts from a random point with the default step size.

As the objective function requires a permutation π and not a vector of continuous numbers $\vec{s}$, the vector $\vec{s}$ is using Random Keys as explained in Section 2.3 which are then decoded into a permutation π before evaluation.

[1] Here "better" is used in the context of continuous optimization, where minimization is the norm. Traditionally a better solution has a smaller objective value in case of continuous optimization.

2.4.2. Adaptive Differential Evolution

Differential Evolution by Price and Storn [25] is a genetic algorithm for optimizing black-box functions. As with the Generating Set Search, the implementation used is provided by the BlackBoxOptim.jl package by Feldt [11] for the Julia programming language.

The population is initialized with uniformly distributed random points. Mutation and crossover to create a new individual are performed according to parameters F and CR using randomly selected solutions $\vec{a}, \vec{b}, \vec{c}$ for every solution $\vec{x}$ in the population. A new solution is then created according to the Equation 2.4. If the new solution is an improvement the current solution is replaced.

$$\vec{y}_i = \begin{cases} \vec{a}_i + F \cdot (\vec{b}_i - \vec{c}_i) & \text{with probability } CR \\ \vec{x}_i & \text{otherwise} \end{cases} \tag{2.4}$$

This formula corresponds to the strategy DE/rand/1/bin which will be used within this thesis. The alternative formulation DE/best/1/bin for example uses the best solution in the place of $\vec{a}$.

The approach is adaptive as meta-optimization is taking place for the parameters F and CR — the difference scale factor and crossover probability respectively — in order to obtain better results. Each of these parameters is taken from a random distribution and is associated with the solution vector itself. If no improvement for a solution vector occurs the values are redrawn, while they are kept if an improving solution was found.

As with GSS, here too the solution $\vec{y}$ to be evaluated is a vector of random keys and decoding needs to take place before obtaining the permutation π.

2.4.3. GOMEA

Genetic Optimal Mixing Evolutionary Algorithm, or GOMEA for short, is an adaptive evolutionary algorithm that applies state-of-the-art linkage information learning on its population to guide its search [27]. For this thesis in particular a variant specifically modified to account of the use of random keys — as explained in Section 2.3 — through the use of re-encoding and rescaling [7].

These modifications were initially tested on Permutation Flowshop, another permutation problem different from Order Acceptance Scheduling, and make GOMEA diversify more.

Its performance stems from the use of a the mixing operator in combination with linkage learning. This learning results in a model that is referred to as a Family of Subsets, which groups correlated – linked – variables together. Various forms of this model exist, such as the Univariate factorization in which the model simply consists of each variable on its own, or the Neighbours factorization in which each element occurs in a subset with each of its neighbours.

A more involved type of model is the linkage tree, which extracts dependency information from the population into a dependency matrix — using Formula 2.7[7] — and performs UPGMA[2] hierarchical clustering through a fast implementation such as the one described in [13].

The Dependency Matrix consists of a few parts, δ_1 and δ_2.

The first of the two – δ_1 – is the inverse entropy of the probability that one key appears before another in the same solution.

$$\delta_1(i,j) = 1 - S\left(\frac{1}{n}\sum_{k=0}^{n-1}\begin{cases}1 & \text{if } r_i^k < r_j^k \\ 0 & \textit{otherwise}\end{cases}\right) \tag{2.5}$$

Where r_i^k is the random key of order i of population member k and S is entropy defined as:

$$S(x) = \begin{cases} 0 & \text{if } x \leq 0 \text{ or } x \geq 1 \\ x \cdot \log_2(x) + (1-x) \cdot \log_2(1-x) & \textit{otherwise} \end{cases}$$

And the latter of the two – δ_2 – is the inverse of the squared average distance.

$$\delta_2(i,j) = 1 - \frac{1}{n}\sum_{k=0}^{n-1}(r_i^k - r_j^k)^2 \tag{2.6}$$

[2] The new "dependency-strength" after merging are the weighted mean of the original where the size of each subset is used as weight. Unlike normal hierarchical clustering the implementation in GOMEA maximizes as Formula 2.7 indicates a similarity measure rather than a distance measure.

The final value in the dependency matrix is the multiplication of these two values, where a high value indicates higher dependency and thus should be merged earlier in the linkage tree.

$$D(i,j) = \delta_1(i,j) \cdot \delta_2(i,j) \tag{2.7}$$

Each of these models be combined and filtered, resulting in models such as Neighbours and Linkage Tree and filtered variants where larger subsets are removed.

For each solution mixing is performed in each generation of the algorithm. The mixing goes over all subsets in the FoS model, and copies over the values of the subset of variables from a random donor from the population to obtain a new solution. The new solution is then evaluated to check if this operation does not worsen the fitness of the solution. If fitness is the same or better, it replaces the current solution.

Copying over dependent variables together preserves their good synergy, effectively accounting for their dependent variables.

If mixing yields no improvement for a certain number of generations the same procedure is performed again, but with the random donor replaced by the current best solution. This procedure is referred to as Forced Improvement. The standard threshold for normal GOMEA is $1 + \lfloor \log_{10}(n) \rfloor$, although the variant multiplies this value by 10: $10 + 10 \lfloor \log_{10}(n) \rfloor$.

Finally GOMEA incorporates an automatic population sizing scheme — to make sure the population size does not need to be configured manually. This scheme starts with a population of size n_{base}. Every time a GOMEA has gone through 4 generations the next GOMEA in sequence goes through one generation. If there is no next GOMEA in sequence, a new one is created with a population size twice that of the previous GOMEA.

Due to the complexity, an overview is given in Algorithm 1.

2.4.4. HSSGA

Unlike the previous approaches HSSGA is a metaheuristic specifically for the OAS problem as described in Section 2.1 [10]. It is a mixture of a genetic algorithm and a local search algorithm; and operates directly on the sequence of orders π.

Initial solutions are generated using a mixture of random choice and using a greedy heuristic.

New solutions are generated by either crossover or mutation. In the case of crossover two solutions are selected from the population[3] and have uniform crossover applied removing later duplicates. In the case of mutation a single solution is selected from the population and has 6 orders removed. In both cases orders which are now missing are re-inserted through an insertion operator. Local Search using a Swap Neighbourhood is applied if the newly generated solution is within 0.05 of the original solutions' quality.

The worst solution is removed from the population and the new solution is added in to replace it.

This approach is specifically for the OAS problem due to the use of a greedy heuristic for initial solution generation. Other elements are applicable to other permutation problems.

2.4.5. Generic GA

A simple baseline implementation of a Genetic Algorithm. This algorithm represents a simple initial approach to trying to solve the problem at hand, without in-depth research on the problem itself incorporated into it. This implementation therefore represents early attempts to solve the problem without analysing the problem itself, for example when a problem is first proposed.

The GA consists of uniform crossover combined with elitist selection. Solutions are shuffled and every block of 2 solutions has uniform crossover applied to it, generating two new solutions. The best two out of four replace the current block in the population.

In order to make uniform crossover work the Random Key encoding is used, this is described in Section 2.3.

The population is generated randomly using a uniform distribution over $[0, 1]$ and has a size determined by a binary search like approach, making it larger as long as this improves the final solution quality.

[3] Paper uses a binary tournament with a 0.5 chance of taking the worst solution instead.

Algorithm 1 Permutation GOMEA

function CreateNewGOMEA(f, n, η)
  return a GOMEA on function f, that takes n parameters with a population size of η.
end function

function GetFoS(g) ⇨ In the case of Univariate
  return $\{\{1\},\{2\},\ldots,\{n-1\},\{n\}\}$
end function
function GetFoS(g) ⇨ In the case of Linkage Tree
  $D \leftarrow$ getDependencyMatrix(g)
  $\mathcal{F} \leftarrow$ UPGMA(D)
  return $\mathcal{F}$
end function

function Mix($\vec{s}, \mathcal{D}, \mathcal{F}$)
  for $b \in \mathcal{F}$ **do**
    $\vec{d} \overset{\text{rand}}{\leftarrow} \mathcal{D}$ ⇨ Get a random donor
    $\vec{s}' \leftarrow \vec{s}$ ⇨ Copy s.
    $\vec{s}'_x \leftarrow \vec{d}_x$ for each $x \in b$ ⇨ Move over values from donor $\vec{d}$ for variables in f.
    $\vec{s}' \leftarrow$ Rescale($\vec{s}', b$) with $P = 0.1$ ⇨ Random Rescaling
    $\vec{s} \leftarrow \vec{s}'$ if $f(s') \geq f(s)$
  end for
  return $\vec{s}$
end function

procedure Step(g) ⇨ Perform a step on a GOMEA g
  $g.population \leftarrow$ [ReEncode($\vec{s}$) for $\vec{s} \in g.population$] ⇨ Reencoding
  $\mathcal{F} \leftarrow$ GetFoS(g)
  for $\vec{s} \in g.population$ **do**
    $\vec{s} \leftarrow$ Mix($\vec{s}, g.population, \mathcal{F}$) ⇨ Replace $\vec{s}$ with the improved solution.
    if $g.generationsWithoutImprovement < Limit_{FI}$ **then**
      $b \leftarrow$ best($\mathcal{G}$)
      $\vec{s} \leftarrow$ Mix($\vec{s}, [b], \mathcal{F}$) ⇨ Forced Improvement
    end if
  end for
  $g.generations \leftarrow g.generations + 1$
  Update statistics for g. ⇨ For example generations without improvement.
end procedure

procedure multiPopulationGOMEA(f, n)
  $\mathcal{G} \leftarrow$ [CreateNewGOMEA(f, n, η_{base})]
  $steps \leftarrow 0$
  while ! Termination Condition **do** ⇨ Can be time, solution quality.
    for $g \in \mathcal{G}$ **do**
      Step(g)
      if mod ($g.generations, 4$) $\neq 0$ **then**
        break ⇨ No generational step for the remaining GOMEA's
      end if
      if g is last($\mathcal{G}$) **then**
        $\mathcal{G} \leftarrow \mathcal{G} \cup$ [CreateNewGOMEA($f, n, 2 \cdot g.\eta$)]
      end if
    end for
  end while
  return best($\mathcal{G}$) ⇨ Return the best solution from all GOMEA's.
end procedure

The LS variant reuses the Local Search as defined by Chaurasia for HSSGA [10] by including it in the black-box. This serves as an initial look into mimetic algorithms: the combination of genetic and local search.

2.5. Quality Measures

Not every instance is made equal. For example larger instances may involve larger objective values. Simply calculating the average objective value does not have a significant meaning.

An absolute metric for quality would be to compare the percentage difference between the optimal solution m^* and the solution found by the metaheuristic m for a given instance, and averaging over that.

$$\text{Gap} = 1 - \frac{m}{m^*} \tag{2.8}$$

Obtaining the optimal solution and its objective value is however unfeasible for larger problems due to the NP-hardness of the problems that will be analysed in this thesis. As a replacement for this both the gap with a known upper bound and the best found solution is used in place of the optimal solution.

The use of an upper bound shows absolute performance, and thus gives a metric that shows how close approaches get to optimal. It should be noted however that if the upper bound for an instance is not tight the gap for a found solution will never be 0.

The use of the best found solution does not have this problem: at least one approach will have a gap of 0. This makes it easy to quantify performance between the evaluated algorithms, but can underestimate how bad the solution is when all tested approaches perform badly.

2.6. Simple Extensions

Using the Encoding approaches above OAS instances can now be solved, but the amount of information available to the solver is lacking compared to specialized approaches. The state-of-the-art approach HSSGA uses information unavailable to a black-box approach for initialization as well as search. This gives HSSGA a significant edge over the black-box approaches and makes comparison more difficult: is it outperforming on basis of general search methodology or on basis of additional information alone? Does a black-box approach, when given the same information in some structured manner, still get outperformed?

The goal of the following extensions is to provide some of this information to the solver without modifying the approach itself to equalize the field in a more generally applicable manner.

In non-selection machine scheduling problems the total tardiness, makespan or some other objective would be minimised. This information is unavailable currently to the optimizer and including this information as a minor element into the objective can help. A schedule that takes less time for the same selection of orders can be seen as better, additionally this may guide the optimizer so that there is more time available to fit in new orders. This approach is used in HSSGA [10] to enforce a preference over solutions with equal profit.

Using this hint does not require modification of the optimization approach itself. Using the completion time of the last order C as well as the latest deadline $\bar{d}_{\max}$ we can state that $0 \leq \frac{C}{\bar{d}_{\max}} \leq 1$, as the completion time of any order is before its deadline and by extension the latest deadline.

The smallest difference possible between two solutions with different values δ[4] can then be used construct a new black box f'_{OAS} with P the profit obtained of a given permutation. Such that $f'_{OAS}(a) > f'_{OAS}(b)$ iff $P(a) > P(b)$ or $P(a) = P(b)$ and $C(a) < C(b)$.

$$f'_{OAS}(x) = P(x) - \delta \cdot \frac{C(x)}{\bar{d}_{\max}} \tag{2.9}$$

This hint is hereafter referred to as the Time hint.

Another potential hint to be given to a black-box optimizer is in the form of initial solutions by way in which they can be generated. An order with an early release date is more likely to be near the front of the permutation than an order with a late release date. Similarly a selected order with an early deadline is also likely to be more near to the front than a order with a late deadline. An idea of Lei He

[4] The smallest difference in value δ is bounded by the differences the profits and weights can make occur. One can thus use the GCD of all the weights together to get a value for δ

(personal communication, December 2018) is to set the range of the random keys to be proportionate to its release time and its deadline or due date for initial solution generation.

When the black-box uses random keys as input, most black-box optimization approaches account for the presence of lower and upper bounds. Here we propose to use Lei's idea for initial solution generation above in a different but very similar manner: use the release time and deadline as bounds of the variables themselves.

This is hereafter referred to as the Full Bounds hint.

By doing this the underlying permutations in the initial solutions are no longer generated in a completely random fashion; more orders will be scheduled in the initial solutions giving a "hot start" to the optimization process as it no longer needs to figure out the time windows itself. In the case of low width[5] the gains are the biggest: the ordering is already mostly determined by the windows themselves.

Instances on which most time windows are the same and therefore has a very high width the gains disappear and generation becomes (almost) random again.

Care has to be taken when using this approach however. This construction may not allow for the encoding for an optimal answer and hence cause the heuristic to be theoretically unable find it when using the original encoding as above. Proof of this (by an example) is as follows.

Lemma 2.6.1. *If the range of the random key v_i for an order i is restricted by an lower bound $lb_i = cr_i$ and upper bound $ub_i = cd_i$: $lb_i \leq v_i \leq ub_i$, and where an order is not performed if it passes its deadline. Then there are instances where the optimal answer cannot be encoded by the ordering given by the random keys v.*

Proof. Given an instance with two orders a and b, where $\bar{d}_a < r_b$ (hereafter referred to as 1), $e_b > e_a$ (2), and $s_{ab} > \bar{d}_b - r_b - p_b$ (3), lastly a and b can each be performed before its deadline as the first item (4).

We will call the random key value for order a and b, v_a and v_b respectively.

Given the lower and upper bounds for each random key:

$$v_a \leq \alpha \cdot \bar{d}_a \text{ and } \alpha \cdot r_a \leq v_b$$

Combining this with $c \cdot \bar{d}_a < c \cdot r_a$, due to property (1) of the given instance. We obtain:

$$v_a \leq c \cdot \bar{d}_a < c \cdot r_a \leq v_b$$

And therefore $v_a < v_b$, any permutation that sorts these two keys thus ends up with the same result: $[a, b]$. In this permutation a completes before its deadline (by property 4), thus b incurs a startup time of s_{ab}. b therefore would complete after its deadline due to (3):

$$C_p = r_i + s_{ab} + p_b < r_i + \bar{d}_b - r_b - p_b + p_b < \bar{d}_b$$

As b completes after its deadline, no profit is obtained for order b, the total profit for any schedule found using this encoding is thus equal to e_a.

The permutation $[b, a]$ has at least a profit of e_b however, as by (4) b will be performed. Given that $e_b > e_a$, any solution encoded using this approach will be suboptimal. □

A short summary: the restriction placed on the random keys may make an order be selected and performed before another, stopping the latter from being performed even if the latter has more value. The implicit nature of selection together with the Full Bounds hint therefore do not work well together.

Replacing the proportional upper bound with a simple upper bound of 1 and setting the lower bound to be

$$lb_i = \frac{r_i - \min_j(r_j)}{\max_j(\bar{d}_j) - \min_j(r_j)} = \frac{r_i}{\max_j(\bar{d}_j)} \text{ if } \min_j(r_j) = 0$$

will not have this issue and can encode every single permutation[6]. But has a similar property to the approach above; the likelihood of one order ending up before the other is still proportional to its starting time.

[5] Width is defined to be the maximum number of overlapping time windows over all points in time, as used and (more precisely) defined in Baart's thesis [4]

[6] Any instance of the OAS problem where $\min_j(r_j) \neq 0$ can be turned into an instance $\min_j(r_j) = 0$ without affecting solutions themselves by shifting the all time windows.

Lemma 2.6.2. *Given that $r_i < \bar{d}_i$ ∀ orders i. If the starting time of the random key v_i for an order i is restricted proportional to its release time r_i, then all permutations π can still be encoded.*

Proof. All random keys can take on values in the range $[lb, 1]$, where lb satisfies the following property.

$$lb \geq lb_i \forall \text{ orders } i \tag{2.10}$$

The lowest lb with this property is given by the maximum over all lower bounds.

$$lb = \max_i(lb_i) \tag{2.11}$$

$$= \max_i \frac{r_i - \min_j(r_j)}{\max_j(\bar{d}_j) - \min_j(r_j)} \tag{2.12}$$

$$= \frac{\max_i(r_i) - \min_j(r_j)}{\max_j(\bar{d}_j) - \min_j(r_j)} \tag{2.13}$$

$max_i(r_i) < max_i(\bar{d}_i)$ as well. As otherwise an i exists for which $r_i \geq max_i(\bar{d}_i) \geq \bar{d}_i$, which is a contradiction with that $r_i < \bar{d}_i \forall$ orders i.

$$lb < \frac{\max_i(\bar{d}_i) - \min_j(r_j)}{\max_j(\bar{d}_j) - \min_j(r_j)} = 1 \tag{2.14}$$

As $lb < 1$, we can pick keys $k_i = lb + \frac{i}{1-lb}$ and have $k_i < k_{i+1} \forall i \in 1 \dots n$ and $lb \leq k_i \leq 1 \forall i \in 1 \dots n$. Any permutation q can therefore be encoded by the assignment of these keys to v's. □

In the usual random key encoding values are picked from $[0, 1]$ for all orders. Resulting in $P(v_a < v_b \mid a \neq b) = \frac{1}{2}$. The aforementioned approach however is intuitively more likely to put an order with an early release time before one with a later release time.

Assuming both v_a and v_b are drawn from a uniform distribution with their corresponding range $\mathcal{U}_i(lb_i, 1)$ for $i \in \{a, b\}$. Assume that v_a is the one with the lowest release time of the two.

$$P(v_a < v_b \mid a \neq b, lb_a < lb_b) = P(v_a < v_b \mid a \neq b, lb_a < lb_b, v_a < lb_b) \cdot P(v_a < lb_b \mid lb_a < lb_b) \tag{2.15}$$

$$+ P(v_a < v_b \mid a \neq b, lb_a < lb_b, v_a \geq lb_b) \cdot P(v_a \geq lb_b \mid lb_a < lb_b) \tag{2.16}$$

When $v_b \geq lb_b > v_a$

$$P(v_a < v_b \mid a \neq b, lb_a < lb_b, v_a < lb_b) = 1 \tag{2.17}$$

And similar to the two uniform cases above

$$P(v_a < v_b \mid a \neq b, lb_a < lb_b, v_a \geq lb_b) = \frac{1}{2} \tag{2.18}$$

$P(v_a < lb_b)$ can be calculated according to the distribution

$$P(v_a < lb_b \mid lb_a < lb_b) = \frac{lb_b - lb_a}{1 - lb_a} \tag{2.19}$$

And as $P(v_a < lb_b) + P(v_a \geq lb_b) = 1$

$$P(v_a \geq lb_b \mid lb_a < lb_b) = 1 - \frac{lb_b - lb_a}{1 - lb_a} \tag{2.20}$$

Therefore

$$P(v_a < v_b \mid lb_a < lb_b, a \neq b) = 1 \cdot \frac{lb_b - lb_a}{1 - lb_a} + \frac{1}{2}\left(1 - \frac{lb_b - lb_a}{1 - lb_a}\right) \tag{2.21}$$

$$= \frac{1}{2}\frac{lb_b - lb_a}{(1 - lb_a)} + \frac{1}{2} \tag{2.22}$$

$$= \frac{1}{2}\frac{r_b - r_a}{\left(\max_i(\bar{d}_i) - r_a\right)} + \frac{1}{2} \tag{2.23}$$

Swapping around $lb_a < lb_b$ to $lb_b < lb_a$ yields

$$P(v_a < v_b \mid lb_b < lb_a, a \neq b) = 1 - \left(\frac{1}{2}\frac{r_a - r_b}{\left(\max_i(\bar{d}_i) - r_b\right)} + \frac{1}{2}\right) \tag{2.24}$$

$$= \frac{1}{2}\frac{r_b - r_a}{\left(\max_i(\bar{d}_i) - r_b\right)} + \frac{1}{2} \tag{2.25}$$

Altogether

$$P(v_a < v_b \mid a \neq b) = \frac{1}{2}\frac{r_b - r_a}{\left(\max_i(\bar{d}_i) - \min(r_a, r_b)\right)} + \frac{1}{2} \tag{2.26}$$

This does come at the cost of losing the influence of the deadlines themselves however. If release times for example are almost simultaneous but deadlines are not, performance may degrade compared to the approach using both bounds. Which could be made to work by addition of an alternate procedure such as a local searcher.

For example, when one order's a window surrounds another's b the probability that a appears before b is higher than preferable. In such a case there are negative consequences for being able to schedule orders such as b with smaller windows due to the larger amount of orders (such as a) scheduled beforehand, even if these orders can also be performed later.

Being unable to find the optimal solution due to the optimal solution being unencodable is arguably worse than slightly worse initial solutions. The choice was made to use the one-sided Bounds hint in all circumstances and will therefore be referred to as the Bounds hint.

2.7. Summary

- The OAS Problem is a complex problem merging both scheduling and selection.
- Solutions for the OAS problem can be encoded as permutations.
- These permutations can be encoded as continuous values using the Random Key encoding. This was already used in Permutation GOMEA, but now also for continuous black-box approaches for the OAS problem.
- The approaches considered include the state-of-the-art for the OAS problem, HSSGA, as well as the black-box approaches GOMEA, GSS and ADE. Furthermore, a simple genetic algorithm is included as point of reference.
- Without modifying a black-box approach internally hints can be provided though the range of continuous values for each variable, as well by modification of the black-box itself.
- Examples of this are the Time Hint and the Bounds Hint which use completion time and the release times & deadlines respectively to provide a hint. The bounds hint is new for the OAS problem.
- Solutions strictly adhering to the bounds hint may not be able to encode the optimal solution due to interaction with the implicit selection.
- This can be resolved by loosening the bounds, for example by only using the release time to determine bounds.

3

Experimental Setup and Instance Generation

In the previous Chapter 2 the problem itself, as well as the approaches used to solve them, were introduced. In order to improve and find the good parts of each approach and where things go wrong we need to know well the current approaches perform. In order to be able to evaluate the performance of these approaches on this problem, test cases – instances – for the OAS problem are required.

This chapter proceeds to set up the experiments in which these instances are used for both configuration of parameters and comparison. This chapter finally states the hypotheses and expectations for these experiments.

3.1. Instances

The instances used in the experiments cannot be created completely randomly. In particular instances that are trivially solvable are unlikely to show differences nor weaknesses of the approaches themselves.

Instead, the instances need to be varied as to find cases in which an approach works well compared to the others; and more importantly: the cases in which it lags behind.

As such the question “What instances have interesting properties that warrant inclusion for testing?” has significant importance for the experimental setup. A visualisation was developed in order to visually inspect properties of instances and solutions, as the large amount of parameters for an instance of the OAS problem can make it difficult to discern structure.

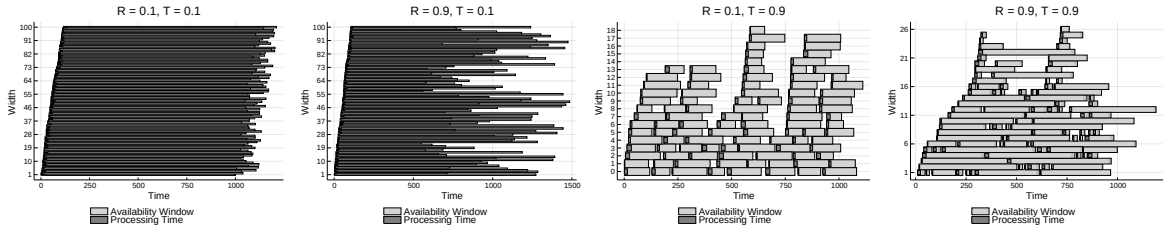
The first instances are a standard benchmark set from literature. The only benchmark set that is easily available and in common use for the OAS problem is one by Cesaret, Oğuz and Salman [9] and is based off a made-to-order case.

While this benchmark set has many realistic traits, it is missing many others as well. If an approach works well on one case that does not mean the performance transfers nicely to other real world situations.

This is illustrated by one notable missing trait in the current benchmark set: in real world situations time and value are often strongly correlated due to pricing being per hour worked and not completely independent. This correlation has been noted by Baart to negatively impact performance for the exact algorithm proposed within the same thesis [4]. Including such a case is hence important to make sure the same degradation of performance does not occur for metaheuristics as well.

Another trait of the standard benchmark set includes random setup times; in practice setup times are often more structured. It can correspond to traveling time and may have the triangle inequality hold, or be clustered for example when reconfiguration of a machine is required when switching between different kinds of tasks. One approach could exploit this structure better than another, showing a relative performance improvement whenever such traits are present. Rather than a degradation, this is a case where a performance increase could be found.

The instances themselves should be sufficiently difficult as well, an easy instance to solve likely has all approaches working very well; potentially to the point of solving the instance to optimality. Such



(a) $R = 0.1\ \tau = 0.1$ #1 (b) $R = 0.9\ \tau = 0.1$ #1 (c) $R = 0.1\ \tau = 0.9$ #1 (d) $R = 0.9\ \tau = 0.9$ #1

Figure 3.1: Plots of Cesaret Instances

instances cause no discernible performance difference between approaches. The instances hence should be of sufficient difficulty in order to locate performance differences, both positive and negative.

As such new benchmark instances are generated on basis of these real world scenarios — Correlated Time and Value and Satellite Scheduling — take up the majority of the new instances.

The remainder of the instances contain deceptive or misleading structure to find weaknesses. Many approaches have weak points related to deceptive structure, local optima that are the opposite of closely related to the actual optimal solution.

The remainder is taken up by a pair of instances related to the artificial construction of local optima as an investigation into the ability of approaches to deal with various less natural structures.

Cesaret Benchmark Instances

The most common benchmark set for Order Acceptance and Scheduling with Sequence Dependent setup times originates from the Tabu Search paper by Cesaret et al. [9]. It is an improvement over the instance generation procedure described in [20] which occasionally generated instances containing unperformable orders.

These instances are parameterized by τ and R: R relates to the variability in the size of the availability windows, whereas τ relates to the variability in release times; as can be seen in Figures 3.1a through 3.1d.

The parameters p_i and e_i are drawn from $\mathcal{U}\{0, 20\}$[1], and s_{ij} from $\mathcal{U}\{0, 10\}$.

The next few ranges make use of p_T — the sum of processing times — in their definition: $p_T = \sum_{i=1}^{n} p_i$.

The release times r_i are picked from $\mathcal{U}\{0, p_T * \tau\}$. The due dates d_i are generated by $r_i + \max_j(s_{ji} + \max(\text{sample from } \mathcal{U}\{p_T \cdot (1 - \tau - R/2), p_T \cdot (1 - \tau + R/2)\}, p_i)$. The deadline is computed by $\bar{d}_i = d_i + \lceil R \cdot p_i \rceil$, and the weight is calculated by $w_i = \frac{e_i}{\bar{d}_i - d_i}$.

The aforementioned instances are interesting, but may not cover all potential use cases. For example the triangle inequality holds in many real world examples and may allow for a significant increase in performance. For example in the case of the Travelling Salesperson problem the triangle inequality allows for a c-approximation algorithm, while such a result is not possible in general.

The instances as defined by Cesaret et al. also have some notable properties. As stated before, many of the items can be scheduled for the instances and the orders that are selected usually have a high value/time ratio. For satellite scheduling the amount of orders that come in for a restricted window of time can be great, requiring a smaller section of the orders to be selected. For businesses the adage "time is money" exists: in practice longer tasks cost more. Value and time should therefore be positively correlated.

Correlated time and Value

Since positive correlation between processing time and value is a realistic trait occurring in many real world situations, the adage "time is money" would indicate this as well.

Considering the effects of this on the performance of approaches is important. It impacts the value/time ratio: large differences do no longer occur and as such picking the orders with the highest value/time will be notably less likely to work. Other problem elements such as the availability windows and sequence-dependent processing times may have increasing importance for selection over this ratio.

[1] The notation $\mathcal{U}\{a, b\}$ indicates the Discrete Uniform Distribution ranging from a to b (both inclusive)

This effect was clear on Baart's exact approach which used knapsack instances as a basis [4]. In this thesis this trait is combined with more natural numbers for the remaining parameters by reusing the generation procedure for the Cesaret benchmark set described in [9].

Most parameters of the instance are therefore generated in the same manner as the Cesaret benchmark set. An additional parameter C is defined to be used as a weight corresponding to correlation. The revenue of each task is then generated by the following formula:

$$e_i = \text{round}\left((1-C)\cdot \mathcal{U}\{0,20\} + C\cdot p_i \cdot \mathcal{U}\{0,20\}\right) \tag{3.1}$$

Satellite-like Instances

This benchmark set models the traits found in satellite scheduling problems. The relatively small availability windows model the fact that an order needs to be visible for a satellite in order to be performed. The sequence-dependent processing times model the movement time between two orders.

Instance generation assumes a satellite moving along the x-axis with a certain speed v_s. Orders are represented by n points (x_i, y_i) for which x_i and y_i are drawn from $\mathcal{U}\{0, \text{duration}\}$ and $\mathcal{U}(-h, h)$ respectively. All orders i have processing time $p_i = 1$ and revenue $e_i = 1$. Orders within a radius of h of the satellite are deemed visible: the release time is the earliest moment this is the case, the due date the latest. This can be given by the following formulas:

$$\begin{aligned} r_i &= \left\lceil \frac{x_i - \sqrt{h^2 - {y_i}^2}}{v_s} \right\rceil \\ d_i &= \left\lfloor \frac{x_i + \sqrt{h^2 - {y_i}^2}}{v_s} \right\rfloor \end{aligned} \tag{3.2}$$

Deadline is equal to due date, and correspondingly weight is equal to revenue.

Setup times s_{ij} are computed based on a maximum 'movement speed' v_r. While the satellite is moving across the point field, it is able to rotate within its orbit to target spots or compensate for its natural orbital movement. This is then simplified as movement with a maximum speed within a certain range – the time window.

The sequence-dependent setup time is then computed based on the solution for the following model:

$$\begin{aligned} &\text{minimize } s_{ij} \\ &v_s \cdot s_{ij} + rx = x_j - x_i \\ &\qquad ry = y_j - y_i \\ &rx^2 + ry^2 \le \left(v_r \cdot s_{ij}\right)^2 \\ &t \ge 0 \quad rx, ry \text{ free} \end{aligned} \tag{3.3}$$

The final value for s_{ij} is the rounded value of this solution. If the system above is infeasible, $s_{ij} = \bar{d}_j - r_j$ is chosen to make this sequence of orders impossible.

Finally instances are generated using a constant speed and total duration, causing larger counts of orders to require a more selective approach relative to the total.

Finding the optimal solution to these instances can be done within a reasonable amount of time using Baart's approach [4]. This approach was used to compute exact optimal solutions for these instances - not just upper bounds.

Trap Instances

Not all instances are equally easy to solve, metaheuristics make use of structure in order to find a good solution. If such structure is missing or misleading — for example in the case of the "deceptive trap" instances used in [27] — finding a good solution becomes significantly more difficult. Doubly so for finding the optimal solution.

The first of the specially designed instances — hereafter named **distracting** — uses a "distraction": a pair of orders with high revenue that disallows any further orders from being performed. Meanwhile performing the orders in a specific order (but excluding one of the pair) yields a better and optimal result. The high value of this "distracting" pair is expected to dominate the search, stopping the metaheuristics from performing proper diversification and therefore ending up in the local optimum.

The instance is parameterized by the amount of orders n and a measure of difficulty k, with all orders i having $r_i = 0$, $p_i = 1$, $\bar{d}_i = 2n + 1$, $p_i = 1$ and $e_i = 2$. The default value for setup times is 5: $s_{ij} = 5 \, \forall i, j$. Furthermore deadlines and setup times are set to be suggestive: $d_i = 2n - 4 - 2i$ and $s_{i(i-1)} = 1$ for all orders i. This makes it so that the descending[2] sequence of orders is the best choice for this problem when k is high enough. Orders $n - 1$ and n are the distracting pair, and have different values for some of the parameters. First of all the deadlines and due dates differ: $\bar{d}_{n-1} = d_{n-1} = 4$ and $\bar{d}_n = d_n = 2 * n + 1$. The distracting element is the notably higher value of order n: $e_n = 2n - 4 - k$; it can only be performed later though because of its release time – $r_n = 2n - 2$ – and only after the first element of the distracting pair — the setup time $s_{in} = 10$ for all i is high enough to make it impossible to schedule order n within its time window with the only exception being $n - 1$ as it has a setup time of $s_{(n-1)n} = 1$. To make the pair more likely to be selected $s_{0(n-1)} = 1$.

Finally w_i is computed by the OAS's problems definition:

$$w_i = \begin{cases} \frac{e_i}{\bar{d}_i - d_i} & \text{if } \bar{d}_i - d_i \neq 0 \\ e_i & \text{otherwise} \end{cases}$$

The second type of trap instance — hereafter named **deceptive** — consisting of n orders (and some parameter k). Orders 1 till $n - 1$ have a due date equal to their release time, this provides significant pressure to minimize the time given that every time unit later incurs a penalty. The special order can only be performed after the first of these "normal" orders. $s_{ij} = \max(j - i + \left\lceil \frac{n}{k} \right\rceil, 2)$, but with $s_{i(i+1)} = 1$, and $s_{(i+1)i} = 2$. This provides pressure to end up with descending order. Any descending order that is not sequential will incur penalties. Meanwhile sequentially decreasing order (with the special order stuck to the end) will give the optimal solution by design whenever the penalty incurred would big enough.

Whenever this penalty is big enough a choice has to be made between performing more orders and being able to perform order 1 and therefore 100. As order 100 has the same value as many orders altogether, the latter would be preferred, and fewer orders would be scheduled.

3.2. Experiments

The experiments consist of a few major groups: first of all the effect of the extensions — limiting random key bounds and the inclusion of completion time in the objective — is evaluated for the black box approaches, and approach-specific parameters are configured. These experiments are performed using the original OAS dataset.

The best configuration(s) for each black box approach, and the reference implementations are then used to compare performance by means of the average gap versus time spent and function calls required to obtain this performance. These experiments will be done on all instances — including the newly defined ones.

As GOMEA builds a model, it will likely require less function calls to obtain a better result than the other approaches under test, this does come at the cost of building a model however. Limiting the amount of function calls can thus be potentially unfair as not all elements that constitute runtime are accounted for.

Therefore all approaches are then evaluated on performance using a fixed time budget on the remaining adjusted datasets. The base time budget is determined as a function of the amount of orders in an instance, given by $t = n^2 + 100$. This value was chosen due to the general magnitude of runtime of the specialized approach HSSGA, with 100ms for general setup and clean-up.

When evaluation involves time, the multipliers $[0.1, 0.5, 1.0, 2.0, 10.0]$ are applied on the base time budget. These multipliers are spread exponentially, giving a good overview of how approaches behave from a tight until a generous time budget. In order to have time measurement be representative all evaluations are performed on the same machine.

In particular evaluation is performed on a virtual machine with access to two cores of a "Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz" CPU and 8 GB of RAM, running Ubuntu 18.04.1 LTS on version 4.15.0 of the Linux Kernel. All evaluations are performed single threaded, eg. on a single core, and are not parallelized.

[2] Ascending is preferred by the decoding procedure when given identical keys for all orders, and as such may be found too easily.

Every black-box solving approach defined in Section 2.4 has parameters. The first set of experiments will focus on each solver and vary their parameters. In all cases the impact of the hints, both the Time hint and the Bounds hint, is investigated.

Configuration Determining the right values to every parameter is an important but already relatively well-researched topic. Prior knowledge on population size for example says that higher is better in terms of avoiding local optima but worse for speed. Given this, most of the process of describing and determining these parameters can be found in Appendix B.

GOMEA An outlier here is GOMEA. The population size is already automatically determined by the procedures within the approach itself [7].

GOMEA models the relationships between variables through the use of a Family of Subsets (FoS) model. The simplest approach to obtain such a model is the Univariate factorization, which assumes independence of all variables and does not perform linkage learning. Other options include Neighbours — assume dependencies with orders next in the encoding itself — and the Linkage Tree — which uses a statistical measure and clustering to find subsets of dependent variables.

The Linkage Tree expands upon the univariate factorization and takes into account dependencies. As variables in a permutation problem are not independent, the Linkage Tree is expected to outperform the Univariate FoS model.

This parameter is hence additional attention as unlike normal parameters the model choice turns it into an adaptive 'learning' approach, and is the core to good performance. Does the addition of linkage learning help or is it solely a complicating element?

Comparisons

Cesaret & Correlated These experiments are performed similarly to those performed for analysing the impact of the hints. The approaches defined in Section 2.4 will be compared on the standard Cesaret benchmark instances and the modified variant with correlated value and time.

No good upper bounds are available for the correlated instances due to difficulty in solving them exactly. A comparative analysis is performed instead.

Satellite-like As the triangle inequality applies to setup times in this case, these instances should be easier. Due to this property evaluating based on hitting time, the time required to find a solution with the optimal objective value, is feasible for these instances.

For these instances the hitting time will be measured with a time limit of 20 seconds. This will be repeated 10 times. During each of these runs the best solution found so far is recorded every 0.1 seconds.

Trap These instances contain features that mislead search strategies, such as local optima that are significantly different from the global optimum. This may make them difficult to solve.

Instances are ran for n^2 ms 100 times each.

3.3. Hypotheses and Expectations

This section contains the hypotheses and expectations for the experiments stated in the section above. For most configuration tasks the goal is simply to determine a good value for the parameters without prior expectations. An outlier for this is the Family of Subsets model used in GOMEA as well as the effect of the hints.

Configuration - GOMEA Model The Linkage Tree (and its variants) are the only models that make use of the linkage learning process. If linkage learning were to be beneficial — which is expected due to the scalability effects on GOMEA itself [27] — these models should outperform models such as univariate which assume independence.

No good performance is expected of the Neighbours model either: the position of the order in the genetic string itself can be random and is random for many instances sets.

Hypothesis 1. *The Linkage Tree Family of Subsets structure (and variants) have improved performance compared to the univariate model.*

This hypothesis is accepted if the median gap is lower — performance is better — and the value is statistically significantly different with $p < 0.01$ for both the normal and filtered Linkage Tree.

Configuration - Hints Giving a bounds hint is mostly beneficial for lower runtimes. By making sure the initial solutions are of better quality approaches with this hint have an head start. Wasting less time in the cases where release times and deadlines already determine a general ordering.

Hypothesis 2. *The bounds hint improves performance for lower runtimes compared to the configuration without the bounds hint.*

Whether this hypothesis holds is dependent on the instance itself, if all release times and due dates and deadlines are close to one another respectively, the bounds do not differ much either from the original 0 to 1 bounds. Therefore this hypothesis is verified for not just each approach, but also for a sample of categories $(R \in \{0.1, 0.9\}, \tau \in \{0.1, 0.9\})$ of instances.

A similar hypothesis is stated for the time hint. As the time hint is present for tie-breaking it is expected to influence search at a later point when the profit itself is relatively static.

Hypothesis 3. *The time hint improves performance compared to the configuration without the time hint.*

This hypothesis is also verified for each approach as well as a subset $(R \in \{0.1, 0.9\}, \tau \in \{0.1, 0.9\})$ of instances.

Statistical Significance tests are performed using the Sign Test;difference is accepted if $p < 0.01$. The hypotheses are accepted if the approach both provides a lower median gap (eg. produces better results) as well as passes the aforementioned statistical test.

Following this is a set of expectations listed for each benchmark set. More general expectations are included in the expectations for the Cesaret benchmark set due to its variety and use in literature.

The remainder of expectations relates to the traits of the instances themselves: each instance was designed to have traits different from that of the original – Cesaret – benchmark set and each other.

Cesaret Instances

1. Approaches including a quick improvement strategy, such as the local search component in HSSGA or the compass search will perform relatively better for lower runtimes.

2. Performance with a larger time budget relies on the ability of getting out of local optima. Population based methods have an edge here compared to compass search, as they natively have this ability. More tuned and specialized methods - such as HSSGA and GOMEA - will likely perform even better.

3. According to literature certain combinations of τ and R have different difficulty levels. Hence performance on instances is expected to vary in general.

4. Specialized – white box – approaches such as HSSGA outperform black box approaches.

Correlated time and Value Instances Selection can no longer be simply done on basis of value per time unit due to correlation: the value per time unit for different order has a differ less from one another, meaning other aspects will be more important. Conversely, more solutions with similar revenue may exist, causing a good solution to be found to be easier. Giving good expectations for these instances is therefore difficult.

Solving to optimality given correlation between time and value turns out to be a more difficult problem for Roberts' exact approach in [4] as well. This difficulty may transfer to instances with correlation as well.

Most instances for Cesaret include a significant portion of the orders however, selection may therefore be less of an issue. Performance can therefore expected to be in line with Cesaret's instances.

Satellite-like Instances More structure is present in the sequence-dependent setup times due to these representing travelling time in a Cartesian plane. Together with the lower width these instances can be regarded as easier. Performance is therefore expected to be better than the Cesaret and correlated instances.

Making use of the triangle-equality like trait is key however: approaches that cannot sufficiently utilize this will perform badly.

Selection is more of an issue however. Only a smaller subset of orders can be scheduled in a solution due to the small overarching time window. This can potentially lead to a non-zero gap.

Trap Instances For this particular set of instances performance is expected to be notably worse than in other cases, due to these instances being designed to be misleading or otherwise difficult to solve.

The **distracting** instances indicate how well the search is being diversified in the presence of an 'easy' to find sub-solution that disallows addition of more orders. HSSGA is expected to perform badly here, whereas the black-box approaches do not have an (as easy) way to find this sub-solution.

Conversely the **deceptive** instances show how well an approach deals with local optima related to order scheduling. Given that actually scheduling a new order may be quite difficult to do with crossover, approaches with a local search element such as HSSGA, but also potentially the Basic GA with local search, are expected to do better here.

3.4. Summary

- Various benchmark sets have been selected and created.
  - The most important set of instances is the one originating from literature [9], containing instances with various kinds of time windows. This benchmark set is hereafter referred to as the Cesaret benchmark set.
  - A derived benchmark set adds a missing trait: time and value are often correlated in reality. This set is referred to as the Correlated benchmark set.
  - Another benchmark set based on a simplified version of an oversubscribed satellite scheduling problem was proposed. This covers a previously unexplored case where a large portion of the orders has to be rejected.
  - Finally some misleading instances were proposed that contain structure that cause various approaches to get stuck.
- Hypotheses regarding the effects of the time and bounds hint have been set.
- General experimental setup for determining parameters and comparative experiments have been given.

4

Results and Analysis

In the previous chapter the general setup of experiments was discussed. In this chapter the results of these experiments, as well as some discussion of them is listed.

First off, the choice of model for GOMEA is evaluated and the conclusions are drawn about the benefits of linkage learning in its current configuration.

Furthermore the results for the approaches given hints on the Cesaret benchmark set is analysed. Conclusions are drawn about the usefulness of both the Time hint and the Bounds hint for various categories of benchmark instances, and potential reasons for their relative impact are given.

Following these are comparative experiments between the approaches on the introduced benchmark instances. Each of the approaches is configured keeping in mind the results as stated in Appendix B. Here the relative strengths and weaknesses of each approach are assessed, with special attention to differences with the current state-of-the-art.

4.1. GOMEA: Model Selection

Performance on all Cesaret instances with $n = 100$ is shown in Figure 4.1. Given the current absence of relations between neighbouring indices, the significantly worse performance for neighbour-using FoS structures was as expected. This could potentially be improved by changing the ordering in the instance itself; for example by making it in order of release time.

Unexpectedly, the univariate model performs remarkably well, even though it is the simplest approach implemented by GOMEA, and unable to make use of dependencies.

The two best performing techniques for building these linked subsets are the linkage tree and its filtered variant. While they have slightly more outliers than the univariate model, their median is lower over the entire set of instances.

The difference between the linkage tree and univariate is statistically significant with $p < 0.01$ and therefore Hypothesis 1 — the Linkage Tree performs better than Univariate — is accepted. The same does not hold for the Filtered Linkage Tree: the difference with the univariate model is not statistically significant with $p \gg 0.01$.

In addition to this, the differences between the top 3 seem negligible: the general value of gaps ranges across $0 - 0.02$, while the difference between medians is at most 0.001. Given that the total value of orders of this benchmark set is at most 2000, with a maximum of 20 profit per order, this difference has a value of at most 2: near the bottom of the distribution. Any of the top three will perform equivalently in practice.

This would indicate that the current setup does not identify or utilize linkage correctly and is a potential point of improvement. Given that this is contrary to expectations, a deeper look into linkage learning will occur in Chapter 5.

For ease future experiments will make use of the one with the lowest median gap: the Linkage Tree model.

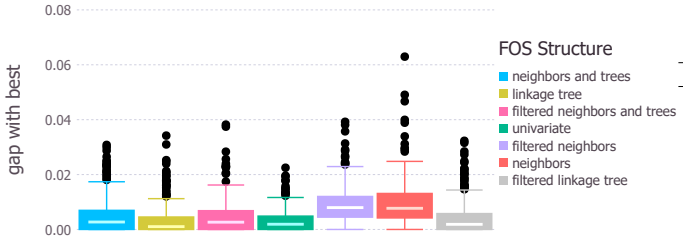


Figure 4.1: Performance of GOMEA using various FOS Structures

FOS structure	median gap (vs best)
linkage tree	**0.00103**
filtered linkage tree	$0.00187^{\dagger}$
univariate	$0.00190^{\dagger}$
filtered neighbors and trees	$0.00265^{\dagger}$
neighbors and trees	$0.00271^{\ddagger}$
neighbors	$0.00771^{\ddagger}$
filtered neighbors	$0.00796^{\ddagger}$

$^{\dagger}p < 0.01$, $^{\ddagger}p < 0.001$
according to sign test with linkage tree

Table 4.1: Median gap with best found

4.2. Impact of hints on performance

GOMEA, ADE and GSS with various have been tested the Cesaret instances, results can be found in Figure 4.2, Figure 4.2 and Figure 4.4 for GOMEA, ADE and GSS respectively. In this case various time multipliers were used on the base time limit to see how performance varies given a particular time budget. This is to investigate the importance of the quality of initial solutions over various runtimes.

The impact of the hints on ADE is similar to that of GOMEA. Note that the range of the gap is significantly higher, with notably worse performance for shorter runtimes without any hints for certain instances.

Last is Generating Set Search. The performance difference for the hints is similar to, but more expressed than for, GOMEA and ADE. Showing that these hints matter more for this local search-like approach as opposed to the population based approaches.

4.2.1. Bounds Hint

The bounds hint changes the generation of initial solutions from random to one utilizing the release times and deadlines. Difference between a variant with and without the bounds hint changes from negligible for $\tau = 0.1$, to notable in the graph for $\tau = 0.9$ for lower runtimes.

For $\tau = 0.1$ and $R = 0.9$ all instances were quickly solved to optimality by GOMEA. No conclusions can be drawn whether the bounds hint had any effect in the case of these instances.

For ADE and GSS this only happens for measurements with later timestamps. The differences with earlier timestamps are however negligible for ADE and GSS; a negligible difference of 0.001 which is insignificant with $p = 0.012 > 0.01$ for ADE and no difference $p \gg 0.01$ for GSS. For these approaches the corresponding Hypothesis 2 — the bounds hint improves performance — is rejected for these instances.

This absence of benefit continues for ADE and GSS, and can now also be shown for GOMEA for $\tau = 0.1$ and $R = 0.1$. For all approaches with $p \gg 0.01$ there is no discernible difference between the version with the bounds hint and the one without for all approaches. This can be attributed to the lack of difference between the release times and deadlines in these instances as can be seen in Figures 3.1b and 3.1d. The range corresponding to all orders using Equation 2.26 will therefore not differ much from the original random range $[0, 1]$, and correspondingly not provide a measurable improvement from completely random keys.

Hypothesis 2 — bounds hint improves performance — should thus be rejected for all approaches with the category with $\tau = 0.1$ and $R = 0.1$.

This changes for the cases in which $\tau = 0.9$, where performance improves significantly when provided with the Bounds hint. In this case Hypothesis 2 is accepted with $p \ll 0.01$ for $R = 0.1$ and $R = 0.9$ for GOMEA up till 10 seconds, ADE up until the final measurement at 100 seconds and GSS for all measured durations. The difference in performance is clearly visible in the graphs as well.

Unlike the instances where $\tau = 0.1$, release times and deadlines differ greatly — as can be seen in Figures 3.1a and 3.1c. Correspondingly the ranges and their corresponding generated permutations differ greatly from random. Orders with earlier release times are statistically more likely to be placed earlier in the sequence, time is utilized better so that more orders are scheduled in the initial solutions.

For longer runtimes ($\geq$ 10s for GOMEA, $\geq$ 100s for ADE) the gap closes. Eventually becoming statistically insignificant with $p \gg 0.01$.

When given enough time a population based approach can find and use the structure that is given by the hint — as such the hint becomes less useful for longer runtimes. Conversely, GSS always restarts from a random point and is unable to utilize the information obtained in previous evaluation steps. A good starting point — such as the one provided by the Bounds hint — is hence is significantly more important for GSS.

4.2.2. Time Hint

The Time hint performs variably on various instances as well. While on many instances the results are in favour of the approaches with the Time hint — in agreement with the expectation that the hint would improve performance — on instances with $R = 0.9$ and $\tau = 0.9$ not providing this hint seemingly performs better.

On the remainder of instances the difference is most noticeable for GSS. GSS obtains significant performance increases in all cases except for $R = 0.9$ and $\tau = 0.9$ with $p \ll 0.01$. Hypothesis 3 — the Time hint improves performance — therefore is accepted for GSS.

For the remaining approaches performance varies. Like in the situation with the Bounds hint, the instances with $R = 0.1$ and $\tau = 0.1$ seem to be easy and have converged for GOMEA, and no conclusions can be drawn here either. For ADE there is no significant improvement either: the difference is not statistically significant with $p \gg 0.01$ early on, with the approaches converging later on. Hypothesis 3 is therefore rejected for both GOMEA and ADE in this case.

Hypothesis 3 has to be rejected in the case of $R = 0.9$ and $\tau = 0.9$ with $p \gg 0.01$ for shorter runtimes and all approaches — including GSS. Furthermore Hypothesis 3 is also rejected for longer runtimes: rather than an improvement, performance regressed by providing the hint for every approach.

For GOMEA no statistically significant improvement can be found for $R = 0.1$ and $\tau = 0.9$ for both shorter and longer runtime with $p \gg 0.01$ and $p = 0.08$ respectively. The results for ADE are similar: no statistically significant difference occurs for most samples and is negligible otherwise. Consequently Hypothesis 3 must be rejected for these cases as well.

Conversely performance has notably improved for GOMEA on instances with $R = 0.1$ and $\tau = 0.1$, there is a notable difference that is statistically significant with $p \ll 0.01$ for short and long runtimes. For ADE, this takes longer, there is no perceptible difference early on, but after 10s however the solution improves notably when given the time hint, providing a notable improvement with $p \gg 0.01$. It should be noted that the difference reduces again to 0.005 with $p = 0.02 > 0.01$ after 100s.

Hypothesis 3 is therefore accepted for GOMEA and ADE on these instances, though care should be taken with ADE as the benefits are restricted to a small window of time.

The results here are a mixed bag. Unlike for the Bounds hint in which the cases that provided benefit for approaches were clear, the Time hint interacts significantly with the approach itself.

GSS seems to require the Time hint more than the population based approaches. Given that the amount of plateaus in the random key encoding is large, behaviour in how to act in the case of a tie is important. Generating Set Search has trouble here and is helped out by the addition of the time hint, which breaks up the plateaus in some cases.

More generally, it seems to hold that when large improvements are still to be made the impact of the hint itself is small. Only once good improvements are harder to come by the hint starts affecting the result itself. Given that the hint was constructed so that ties would be broken in favour of the schedule that spends less time, this makes sense.

A potential reason for the performance decrease is due to the local optima this adjustment introduces. The adjustment is meant to point the optimizer in the right direction: shorter schedules allow you to insert new orders and scheduling new orders allows you to obtain more profit. It can however steer the process into a local optimum where the compacted ordering does not have positions which allow more orders to be performed.

In the case of instances with $\tau = 0.9$ and $R = 0.9$ this is combined with the additional complexity of orders that have short availability windows. Such orders are more likely to incur a waiting period or be past their deadline entirely. In the first case the ties will not be broken in favour of this schedule, while in the latter the order is not scheduled entirely. This makes it difficult for such orders to be selected.

Knowing how the instances are constructed shows the issue in this case: the availability window is stretched so that the order can be performed and time and value are not correlated. This causes these small orders to be quite valuable in terms of profit per time unit. Not scheduling them hence incurs a notable loss and the hint pushes the approaches in that direction.

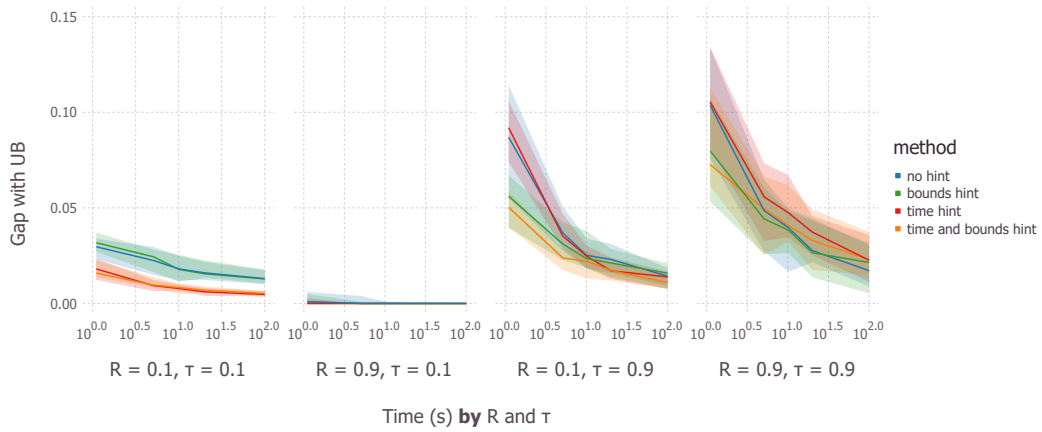


Figure 4.2: Performance vs time for instances with $\tau \in \{0.1, 0.9\}$ and $R \in \{0.1, 0.9\}$ with various hints provided to GOMEA

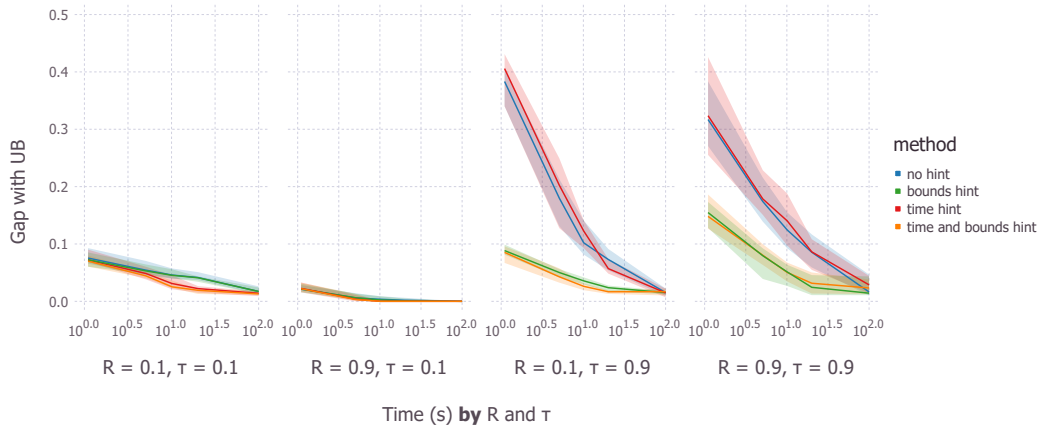


Figure 4.3: Performance vs time for instances with $\tau \in \{0.1, 0.9\}$ and $R \in \{0.1, 0.9\}$ with various hints provided to ADE

Including time in the fitness value is therefore a trade-off and hence both will have to be evaluated on the other instances.

The next few sections will describe comparative experiments between approaches on various instances. The first section describes the Cesaret Instances as this is the standard benchmark set. Followed by Correlation as they are similar in construction to the Cesaret instances. The Satellite instances are discussed next as they do represent realistic instances, but are different in construction compared to the previous two. Finally we have the more misleading instances, which are less realistic than the previously stated instances, but contain elements which may be part of different realistic instances and are difficult and misleading.

4.3. Cesaret Instances

The results for the Cesaret instances can be found in Figure 4.5.

Unlike the expectations, performing a quick improvement does not necessarily guarantee better performance on lower runtimes: ADE does outperform GSS on lower runtimes on instances with $\tau = 0.9$. Though performance on approaches using a quick improvement strategy is generally significantly better.

Getting out of local optima is important as expected. Unlike the other approaches tested, GSS does not significantly improve given a significantly larger time budget.

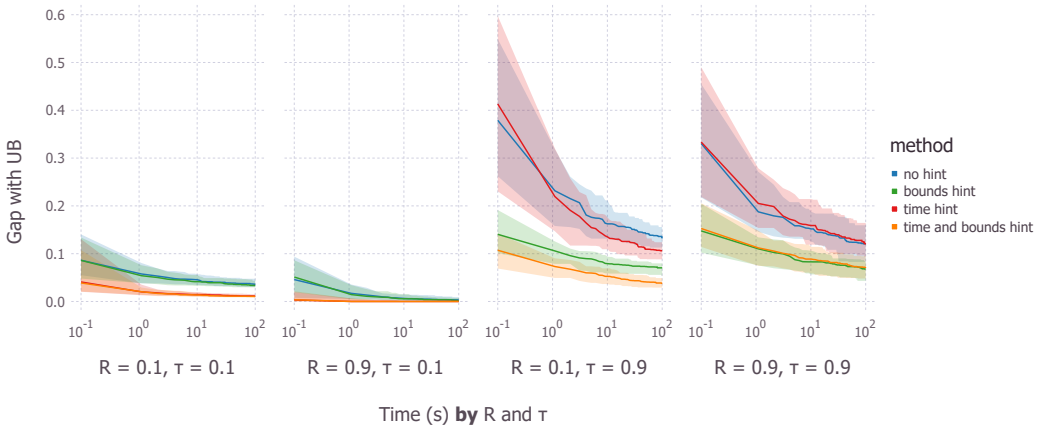


Figure 4.4: Performance vs time for instances with $\tau \in \{0.1, 0.9\}$ and $R \in \{0.1, 0.9\}$ with various hints provided to GSS

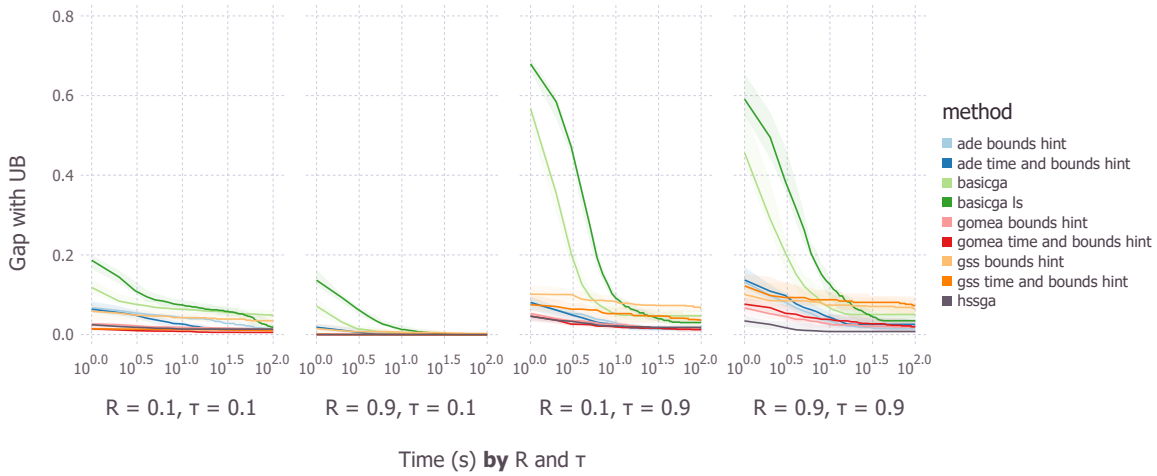


Figure 4.5: Solution quality against time for Cesaret instances

As seen in the configuration runs as well, performance is strongly related to instances themselves, with $R = 0.9$ and $\tau = 0.1$ being noticeably easier than the others, allowing for a zero gap being found for every approach. The gap is relative to an upper bound however, and not the optimal solution itself. Having a gap of zero may therefore not necessarily be possible for all instances in general.

Surprisingly, on many instances GOMEA performs similar to, or even outperforms HSSGA, the current state-of-the-art. The only outlier being instances where $R = 0.9$ and $\tau = 0.9$, where HSSGA performs significantly better than any of the other approaches - including GOMEA. This case is of particular interest as its structure is quite special as well: there are a few orders with low processing times, relatively high value and small availability windows.

Due to the small availability window scheduling these tasks is relatively difficult for most approaches: they need to be in a specific timeslot. Specifically, inserting an order into the right location turns out to be problematic for GOMEA, and a potential cause will be further discussed in Section 4.7.

4.4. Correlation Instances

The results for the Correlation instances can be found in Figure 4.7. Due to lack of good upper bounds, the gap is computed with the best found solution instead. Therefore no absolute performance statement can be made based on these results.

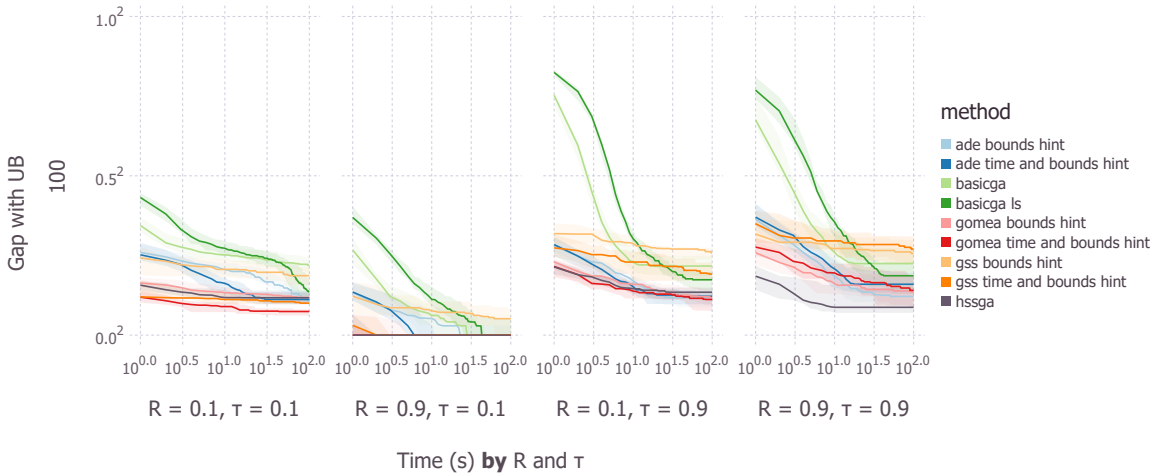


Figure 4.6: Square root of Solution quality against time for Cesaret instances

approach		ade		basicga		gomea		gss		hssga
		+b	+tb		+ls	+b	+tb	+b	+tb	
ade	**+b**	*59*	48	23	22	48	52	26	41	51
	+tb	48	*61*	23	22	49	54	26	43	54
basicga		23	23	*23*	15	23	23	15	23	23
	+ls	22	22	15	*23*	21	23	18	23	23
gomea	**+b**	48	49	23	21	*80*	53	26	40	52
	+tb	52	54	23	23	53	*188*	26	44	57
gss	**+b**	26	26	15	18	26	26	*27*	25	27
	+tb	41	43	23	23	40	44	25	*45*	45
hssga		51	54	23	23	52	57	27	45	*72*

Table 4.2: Counts of Gap with best = 0, given that another approach has a gap equal to zero

Performance seems to be similar to across the board with correlation having a negligible impact on the relative solution quality. No approach seems to degrade compared to the other approaches and all approaches therefore behave similarly relative to one another, even in the case of correlation being present.

As no statements can be made on performance relative to the optimum, and an upper bound may have degraded itself. No conclusion can be drawn about whether all approaches are affected by correlation in general.

In Table 4.2 a table is given with the count of cases in which a given approach provided the best solution, given that another (or the same) approach gave the best solution as well. This shows that GOMEA with the time and bounds hint finds the best solution 188 times out of the total of 240, whereas HSSGA finds the best solution 72 times out of 240. There exist 57 instances in which both find the best solution. From this it can be inferred that there are 18 instances where HSSGA finds the best solution where GOMEA+tb does not and 131 where GOMEA+tb does and HSSGA does not, and 203 instances where either finds the best solution. The remainder is split over GOMEA+b and ADE (+b, +tb).

Finally, whenever HSSGA finds the best solution, it is usually also quick to do so. Unlike GOMEA where finding this solution takes longer.

GOMEA+tb performs the best overall, but the aforementioned numbers indicate that there are still improvements to be had in terms of speed and performance in certain cases.

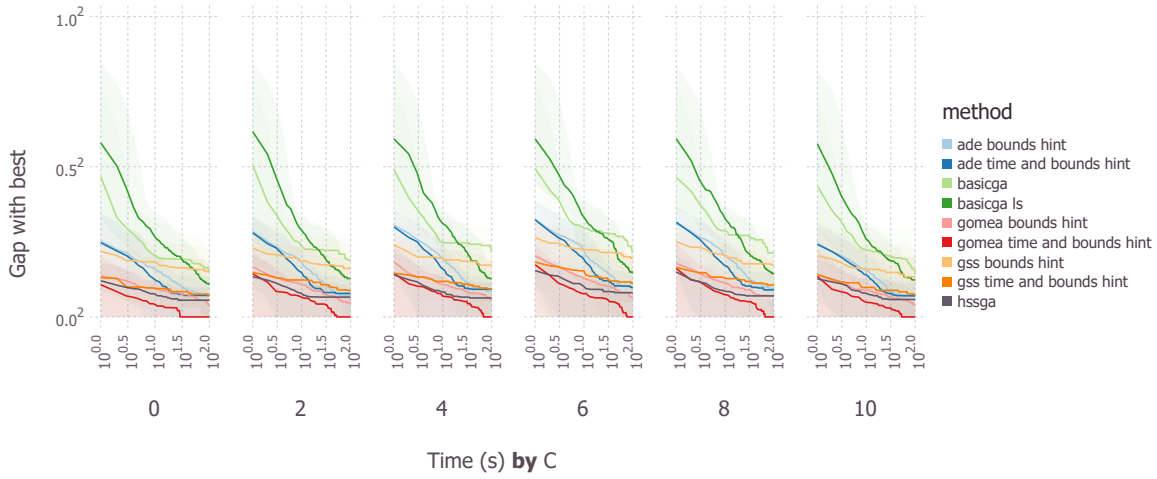


Figure 4.7: Square root of Solution quality against time for correlated instances

	Approach								
	ade		basic ga		GOMEA		GSS		hssga
n	+ b	+ tb	No LS	LS	+ b	+ tb	+ b	+tb	
100	92.0	91.0	67.0	42.0	**100.0**	**100.0**	8.0	32.0	90.0
150	67.0	69.0	61.0	0.0	51.0	77.0	0.0	1.0	**86.0**
200	66.0	**73.0**	0.0	0.0	6.0	28.0	0.0	0.0	70.0

b: bounds hint **t**: time hint

Table 4.3: Success percentages - % of cases in which optimum was found within 20 seconds for Satellite instances

4.5. Satellite Instances

The satellite instances tell another, but similar, story for n = 100. GOMEA is slower, but more likely to find the optimum for size 100 than HSSGA. See Table 4.3, Figure 4.8 and Figure 4.9.

An increase to n=150 paints a different picture however: GOMEA requires notably more time and the amount of cases in which the optimum is found drops significantly as well. HSSGA is the best performer in this case. With ADE following closely, albeit with a lower success percentage.

For n=200 ADE outperforms HSSGA, being notably quicker than any other approach - with most not finding the optimum within the time limit and HSSGA taking noticeably longer.

ADE hence seems to scale better, the final step towards the optimal solution takes notably less time. This is potentially caused by ADE looking at the random keys as continuous variables, rather than discrete possible values, thus potentially being able to make use of the underlying structure of the problem itself: all parameters are dependent on the positioning of a certain point in 2D space, giving some continuous structure to the instance itself.

Finally it should be noted that these instances can be solved within 2 seconds each using an implementation of Baart's approach described in [4]. The performance of metaheuristic approaches lags noticeably behind here. The cause of this may be related to the smaller subset of orders that are selected in a solution, causing a larger part of the random keys to be superfluous for a given solution. A crossover with such a superfluous key is similar to a random mutation and therefore likely less useful.

4.6. Trap Instances

For this particular set of instances performance is notably different from previous instances, as can be seen in Figure 4.11 and Figure 4.10.

For example HSSGA is one of the worst performers on the distracting instances for larger n, whereas it is among the best contenders for the previously stated instances. HSSGA is again ahead of the other approaches in the case of the deceptive instances.

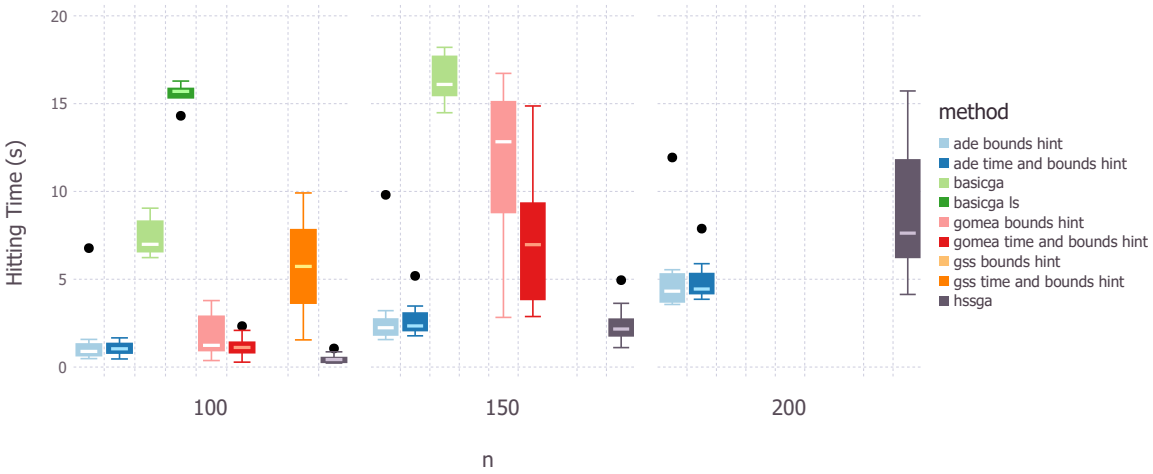


Figure 4.8: Hitting time for satellite instances

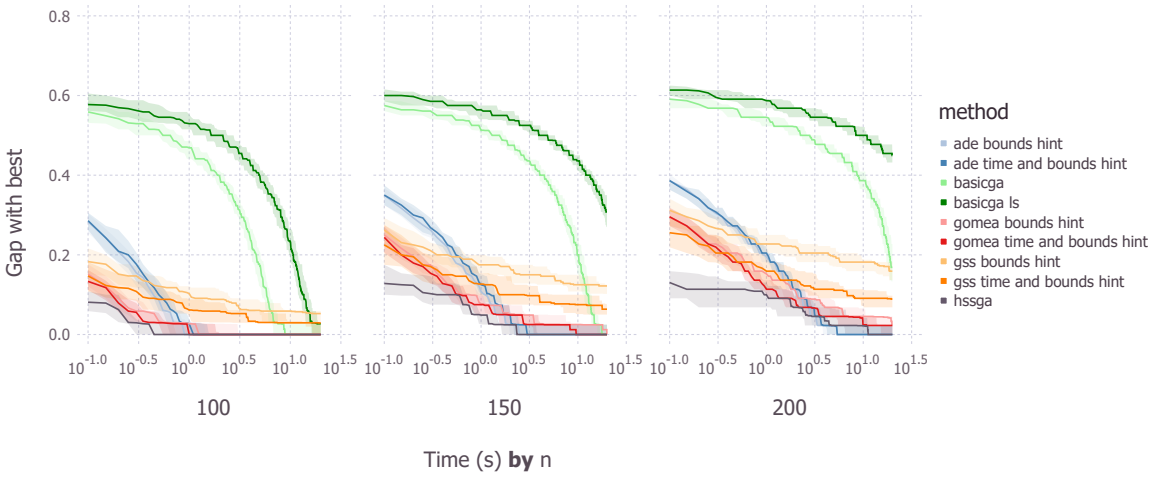


Figure 4.9: Solution quality against time for satellite instances

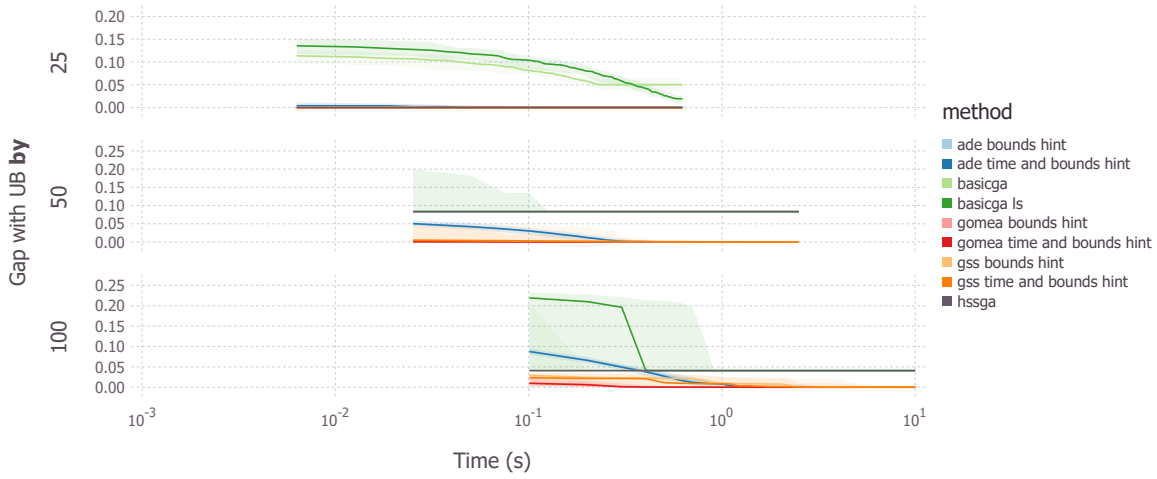


Figure 4.10: Solution quality against time for distracting instances

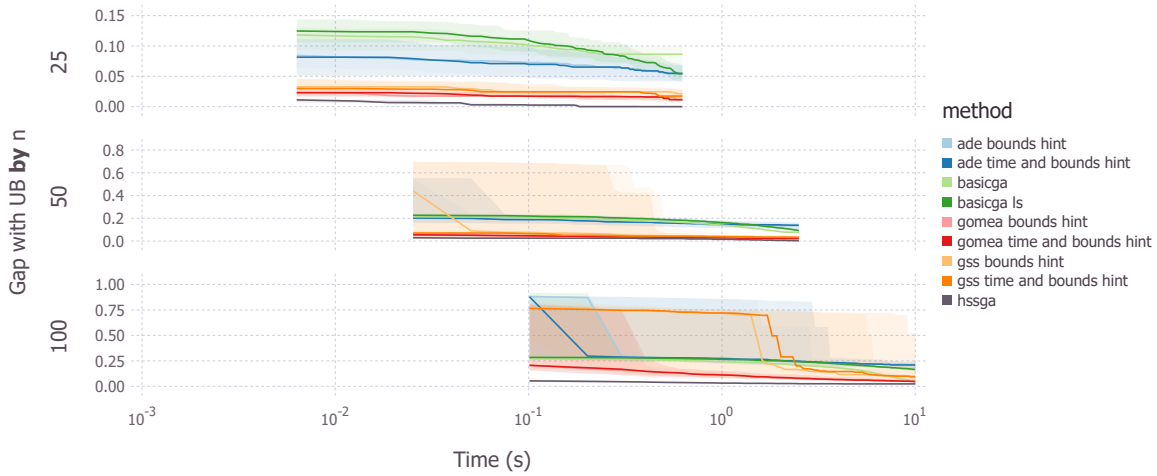


Figure 4.11: Solution quality against time for deceptive instances

Both of these seem to be caused by the insertion operator re-inserting orders that were no longer scheduled whenever this was profitable. In the first case this yields a significant number of cases where the distracting pair takes over the listed solutions, whereas in the latter case inserting orders that previously did not fit is a requirement to get out of local optima.

Generating Set Search performs relatively similar to other approaches on these instances. This is surprising as it performed comparatively worse for the 'normal' instances on longer runtimes. A step alongside an axis behaves like inserting an order, while not explicitly searching expensively for it, which could explain its relatively better performance.

Adaptive Differential Evolution degrades severely in performance on the deceptive instances. This is very much opposite compared to the satellite instances. ADE may therefore require a particular kind of structure that is absent here to work well.

GOMEA performs well in general and is the best performer on distracting instances, but does not hit the performance level of HSSGA on deceptive instances.

Globally speaking, for $n = 100$ none of the listed approaches find the optimal solution for the deceptive instances within the allotted time. This shows that these instances are generally difficult, while we are aware of the optimum.

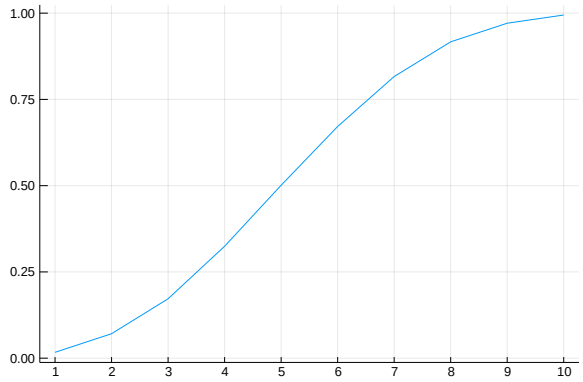


Figure 4.12: Probability of $c_k < c'_j$ for $k = 5$, $n = 10$ and $1 \leq n \leq 10$

4.7. Accidental Reordering due to Random Key Encoding

Inspection of solutions of GOMEA have shown that they often differ by small displacements. A special case was found in Section 4.3, where being a bit off in the ordering could cause an order to be unscheduled.

Given the randomness of random keys some of these this may be caused by the random key encoding. But what is the exact probability of an order being misplaced due to the encoding itself when used in a crossover like fashion? How does it impact the ability of GOMEA to find solutions?

Assuming n keys distributed Uniformly between 0 and 1. Order statistics tells us the k-th smallest key is $\beta(k, n+1-k)$ distributed [12, pg. 63]. Assuming that the donor is a different independent solution from the current solution as well, there is a probability that the key c'_j from the donor appears after the original key c_k at position k: $c_k < c'_j$. This probability is approximated and plotted for $n = 10$ and $k = 5$ with $1 \neq 10$ in Figure 4.12.

From this we can infer that (when the assumptions hold) there is a non-negligible chance that the new ordering between items can be different — even if random keys encode the same permutation: The order at position 6 of the donor may appear before the order at position 5 with $P \approx 0.30$.

The assumptions above may not hold for all approaches, the keys are unlikely to remain uniformly distributed during the optimization process and the hint makes the distributions uniform over different ranges. The reencoding operation in GOMEA trivially causes the population to satisfy these assumptions: each solution is always independently encoded with random keys originating from an uniform distribution over $[0, 1]$.

Swapping two neighbouring elements around may not affect the solution quality too much and may even provide benefits in the form of exploration. The uncontrolled and somewhat unchecked nature is however worrisome, many function evaluations could theoretically be wasted by this effect in the case where a swap of neighbouring elements has significant impact on a solution's quality.

Inspecting Figure 4.13 a plot of a Cesaret benchmark instance with $\tau = 0.9$ and $R = 0.9$. It can be seen that misplacing an order could be very destructive for this instance: small neighbouring windows are likely to be scheduled next to one another and a swap will unselect the earlier of the two, incurring a loss of profit.

4.8. Overall Conclusions and Further Questions

- Different approaches have different strengths. These strengths express themselves in better performance for different instances. Most importantly: some of these strengths are transferable. HSSGA for example includes local searchers that could be used within other approaches to improve on their weaknesses. **(How) can these strengths be combined into a better performing approach?**

- Providing the Bounds hint resulted in notable performance improvements on benchmark instances where the release times and deadlines are dissimilar between orders. Conversely the Time hint varies in its usefulness, and can even decrease performance in the case of $R = 0.9, \tau = 0.9$.

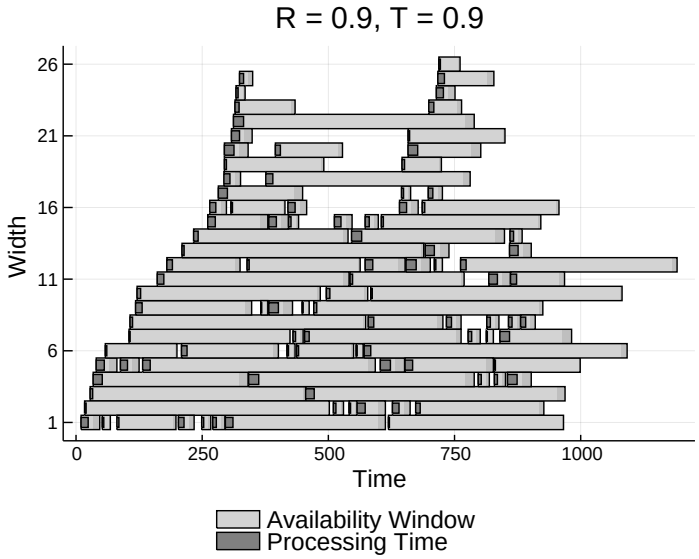


Figure 4.13: $R = 0.9$ $\tau = 0.9$ #1, repeated Figure 3.1d

- For many instances Permutation GOMEA outperforms the others when given hints. Yet many cases such as the satellite dataset and the instances with $R = 0.9, \tau = 0.9$ from the Cesaret dataset indicate otherwise. There are multiple potential causes for this.
  - The performance of GOMEA very much relies on the usage of a good model. In the configuration experiment the differences between the univariate model and the linkage tree model were small, indicating a potential issue. **What model should be used instead to improve performance?**
  - It was shown that the randomness of random keys can cause swaps to occur. Given the potential issues that would be caused by random swaps on instances like $R = 0.9, \tau = 0.9$ it may be better to drop random keys entirely. **How should GOMEA be modified to drop random keys?**

5

Improvements

In the previous chapter it was shown that the state-of-the-art of the Order Acceptance and Scheduling with Sequence Dependent Setup Times problem – HSSGA – can be outperformed by providing a black-box approach with hints, such as providing release times and deadlines as bounds, turning it into a gray-box approach.

The remainder of this thesis will focus on creating improved approaches: while GOMEA does outperform HSSGA in many cases, a number of shortcomings were identified as well. Performance on particular instances is subpar, and one of the key elements – the linkage tree – does not provide benefits beyond that of a univariate factorization; whereas in the case of the discrete Cartesian variant significant improvements are obtained.

The first section will add a local searcher to GOMEA. It is a single search trough a neighbourhood common in scheduling literature - the swap neighbourhood. The section following that will perform a large neighbourhood search using a modified version of Baart's exact approach and see whether incorporating a restricted exact approach will yield benefits.

In Chapter 4 initial results showed that the linkage learning process did not provide the expected benefit, as such the following section will look into variation with a different dependency matrix and what functions can be expected to work well using a specialized kind of analysis.

Finally this thesis proposes a more significant change to GOMEA: rather than work with the random key encoding — which carries some potentially negative effects and limitations — the usage of an encoding closer to permutations itself (saving time decoding) and modification of the mixing procedure to operate on this encoding specifically is recommended.

All experiments from this section onward will be performed on a server with access to 8 cores of an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, with each experiment restricted to the use of a single core.

5.1. Local Search: Swap Neighbourhood

Rather than reinventing the wheel, this first investigation into potential improvements utilizes the local searcher used in HSSGA. This local search procedure performs a single pass through the $O(n^2)$ neighbourhood of all possible swaps, where a new solution is selected if its revenue is higher, or equal equal revenue but with a shorter duration.

As was shown by the inclusion of a local searcher in Basic GA, always performing the local searcher is not a good idea. An $O(n^2)$ neighbourhood can be large, and takes $O(n^3)$ time to evaluate in a black box fashion in the case of OAS. A lack of diversification similar to that in the case of the distracting trap instances may occur as well.

The local search is therefore performed with a probability ρ. Where the performance for $\rho = 0.0$ should be equivalent to the implementation without local search. ρ will be determined empirically though grid search. ρ is likely dependent on n and the time budget due to the runtime cost, as well as the instance itself. In order to make this approach generalize well, the impact of these elements need to be taken into account.

The results for an initial experiment for $0.0 \leq \rho \leq 1.0$ in step sizes of 0.01 can be found in Figure 5.1. Each first instance from the Cesaret dataset for $n = 100$, $\tau \in (0.1, 0.9)$ and $R \in (0.1, 0.9)$ was used, with each point represented by the median of 10 runs. An additional green dashed line represents $\rho = 0.0$. Runs for both with and without the time hint were performed, no time hint is indicated by a dashed line. All runs were given the bounds hint.

A high value of ρ degrades performance as expected. Lower values of ρ increasingly perform better, especially in the case of $R = 0.9, \tau = 0.9$ where the gap is lower than without local search. A better value for ρ also provides better performance independent of time.

The optimal value for ρ seems close to - but not exactly - 0.0. In Figure 5.2 ρ is plotted against the final gap for $0.00 \leq \rho \leq 0.02$ in increments of 0.001. The horizontal orange line is the gap obtained by the $\rho = 0.0$ variant.

Having a value of $\rho = 0.0025$ does not negatively impact performance for instances with $\tau = 0.1$, whereas it provides an improvement in the case of $\tau = 0.9$. This value does not change significantly between these instances, any differences with surrounding potential values are minimal. Hence in the case of the cesaret instances ρ does not need to be determined for each kind of instance specifically.

This leaves the impact of n to be determined. For $n \leq 25$ GOMEA does well enough to make the choice of ρ not have a large impact on the actual performance. For $n = 50$, double the value for $n = 100$, $\rho = 0.005$ is around the optimum for $R = 0.9, \tau = 0.9$, yet low enough to work well for other instances as well. This provides an initial guess for a good value for ρ:

$$\rho = \frac{1}{4n} \tag{5.1}$$

The application of local search has negative side effects as well. Performance on the distracting trap instance is now worse than before: it now gets stuck in the trap.

This can be explained by that many good solutions including the optimal solution start with the same order as the distracting pair, a simple swap (the neighbourhood in use by the local searcher) of the second item will lead to the distracting pair being found, and hence getting stuck in a local optimum.

5.2. Local Search: Specialized Approach

The local search approach described in the previous section, a single pass through the swap neighbourhood, is relatively problem independent. This makes it generally applicable to many permutation based problems. Conversely it is relatively inefficient: the actual structure of the underlying problem is underutilized. One may skip evaluation of solutions where improvement is very unlikely, in [24] for example such a procedure is described for the OAS problem by utilizing the setup time difference. For some problems one may even be able to perform partial evaluation given a specific local searcher.

This section will introduce a large neighbourhood search based on Baart's decision diagram approach [4], as an alternative local searcher. Performing the original exact algorithm is not an option: a runtime of $O(nw^2 2^w |T|)$ is too long in general for a local search approach. Furthermore, the exact algorithm is independent of the solution, a single run on the instance will yield the optimal solution without any need for further searching.

This algorithm is therefore modified into a large neighbourhood search. By fixing the ordering of the jobs — requiring the introduction of ordering constraints — and by using the approximation strategy proposed in Baart's thesis [4], the width w and possible starting times $|T|$ are artificially limited. A selection of jobs is left unfixed, and is chosen such that width is still sufficiently limited (this value is taken to be 5).

This search approach is still significantly more expensive than the non-specialized approach above, but when the restrictions are loosened enough, and given enough time and memory, it will be able to search a significantly larger neighbourhood.

As the time required has increased significantly for this local search operator, the probability for this operator is chosen to be 100 times — approximately the difference in time — smaller than that for the swap neighbourhood local searcher.

Results In Figure 5.4 it can be seen that the increased cost in terms of time does not weigh favourably against the gained performance. Only in the case of low width — the strong point of the exact approach — the performance gained outperforms a simple swap neighbourhood local searcher. As such this

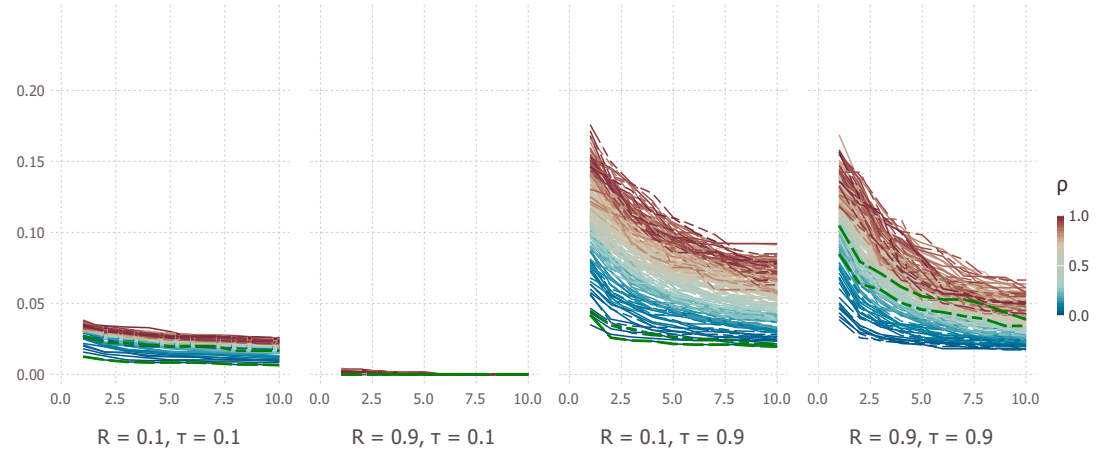


Figure 5.1: Gap vs time for inclusion of local search in GOMEA with $0.0 \leq \rho \leq 1.0$ and $n = 100$. Green line represents GOMEA without local search.

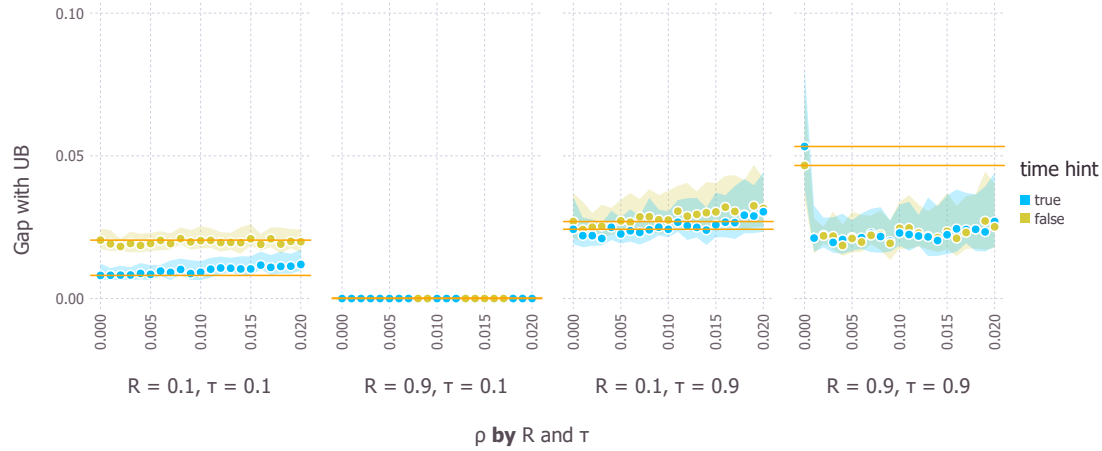


Figure 5.2: Gap at $t = 10$ vs ρ for inclusion of local search in GOMEA with $0.0 \leq \rho \leq 0.02$ and $n = 100$

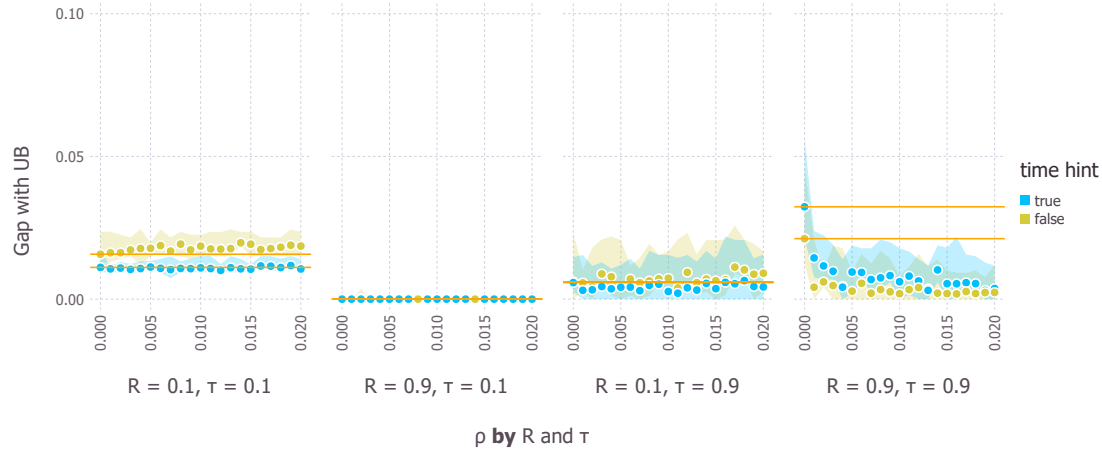


Figure 5.3: Gap at $t = 10$ vs ρ for inclusion of local search in GOMEA with $0.0 \leq \rho \leq 0.02$ and $n = 50$

approach to turning the exact approach into a local searcher is only useful in the limited number of cases where the original approach performed well.

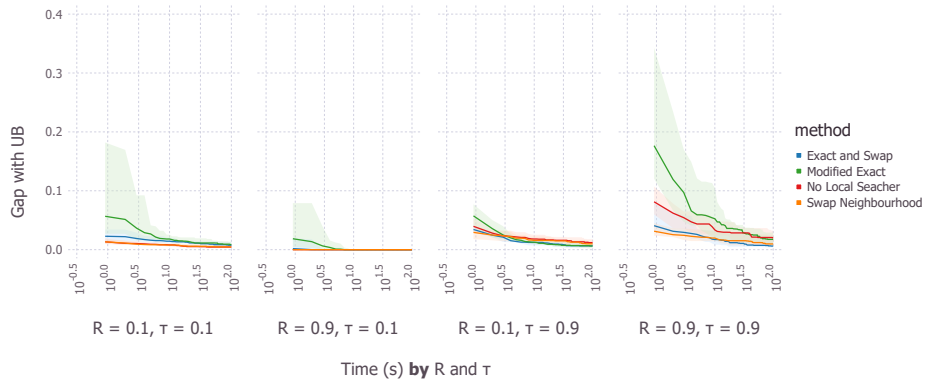


Figure 5.4: Various Combinations of Local Searchers applied to GOMEA and their performance for $n = 100$.

5.3. Linkage Learning: A different dependency matrix

As Linkage Learning is one of the major elements in GOMEA that determine its performance [27], the Linkage Tree would be expected to outperform the univariate model. The contrary was shown in Chapter 4 instead: GOMEA's performance did not increase with the use of a Linkage Tree compared to the Univariate factorization.

The performance improvement is dependent on the model. Which, in the case of GOMEA, is dependent on the dependency matrix and the clustering. As such a dependency matrix filled with random numbers is unlikely to provide an improvement over the univariate factorization, whereas a 'fitting' dependency matrix provides the factorization of the underlying problem with the corresponding performance increase.

But what is a 'fitting' dependency matrix? Our goal is to find a factorization that provides the best performance, logically a fitting matrix should result in this factorization. But what is the factorization of the underlying problem in the case of a permutation-based problem, such as OAS? Which formula would allow you to find this structure in order to fill in a matrix? And how is this factorization affected by the algorithm — the "mixing" in GOMEA — itself?

The optimal factorization is a difficult question in the case of a permutation based problem. By the problem its very nature all variables are dependent on one another — a position in an ordering can only be taken once. Another example can be found in the OAS problem: time is an important aspect that is influences all orders before and after it.

Yet some dependencies are stronger than others. While all orders contribute to time, sequential orders depend more on one another due to setup times. The sequence-dependent setup times are similar to the properties found in the Travelling Salesperson problem as well.

Furthermore the encoding itself is also an important aspect. If there is strong linkage between two elements a and b, indicated by δ_1 — the inverse entropy of $P(a < b)$ — is either high or low. If the two random keys are distributed in such a way that this already holds over all pairs in solutions as well, mixing the keys without keeping the linkage in mind won't change this property in any way either.

Additionally, the linkage according to δ_1 is very likely to be 1 – the maximum possible – in such a case. While that of pairs closer together likely have a lower value, caused in part by the randomness of random keys. This would cause clustering to be very likely to merge far away elements — which do not benefit of the act of mixing together — first.

This is not the only issue regarding this form of linkage, in some problems such as symmetric TSP the reverse of a solution is equivalent. A population could therefore consist of both a forward and backwards copy of a solution. Even if two orders a and b are linked this could result in $P(a < b) = 0.5$. As such the current approach does not suffice for properly using this ordering information.

The realization that there is less linkage over longer distances was already stated in [7]. In this

case the issue was resolved by multiplying δ_1 – the inverse entropy of $P(a < b)$ – with δ_2 – its inverse average squared distance.

Here I propose to simply use average distance instead, and contrary to the original description in Section 2.4.3 do not account for $P(a < b)$ at all. This results in a new formula that is similar to dropping the δ_1 factor while leaving δ_2. Having a low average distance means two elements are always close to one another, not accounting for their ordering. Since setup times are incurred between sequential elements this is a good fit and matches other kinds of linkage as well which were was less or not accounted for previously.

The function to compute the dependency matrix is then stated as follows.

$$D(i,j) = 1 - \frac{1}{n}\sum_{k=0}^{n-1}\left|r_i^k - r_j^k\right| \tag{5.2}$$

Using average distance helps accidental reordering due to the random key encoding. If the two orders often appear together, their inverted average distance will be high and they will be quickly merged together in the linkage tree. As the donor provides both keys, their relative ordering in the new solution does not change and can therefore not have any unexpected reordering within the section itself.

Results The effect of this modification alone can be seen in Figure 5.5. It noticeably improves performance on instances with $\tau = 0.9$, especially $R = 0.1$. For $R = 0.9$ finding the right spot for an order with a small window seems to be costly: only when sufficient time is available performance will improve. For $\tau = 0.1$ it can be inferred that there is not much to improve on average. While not performing better on average, the 95th percentile is lower for the variant using the inverse average distance formula.

Combined with the local search routine and using the Full Bounds hint[1] this seems to be less of an issue, as can be seen in Figure 5.6; HSSGA is outperformed in most (if not all) the instances.

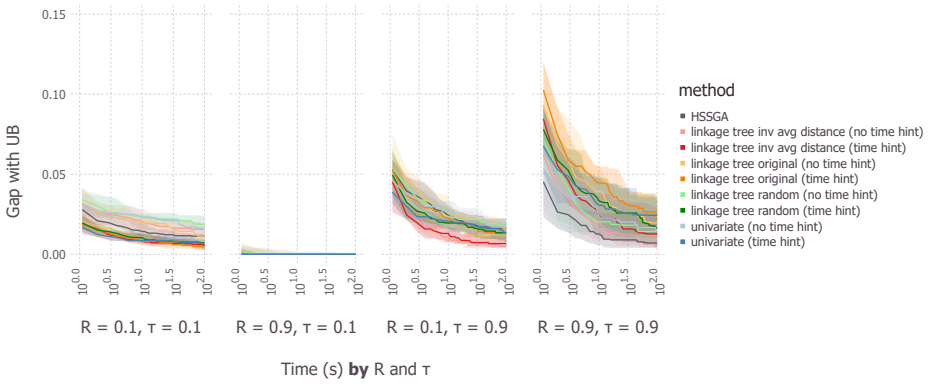


Figure 5.5: Effect of various Dependency Matrix formulas on Cesaret instances. HSSGA is included for reference.

5.4. A GOMEA derived approach

In the previous section the performance of linkage learning was described. It patched up the issue caused by random keys as described in Section 4.7 of the previous Chapter. This issue could — and likely did — have a significant negative impact on performance. But patching up an issue does not mean it is resolved.

In this section a new approach that avoids using random keys entirely is proposed, sidestepping the issue entirely. First the benefits and issues with dropping the random key encoding are stated, along with more performance reasons for a different approach. After that the definitions of new mixing operators — that do not invalidate the solutions — are given. Finally the new mixing operators are put in practice and evaluated.

[1] The local search routine ignores key bounds and swaps them whenever necessary to obtain an improvement, allowing formerly unencodable solutions to be encoded.

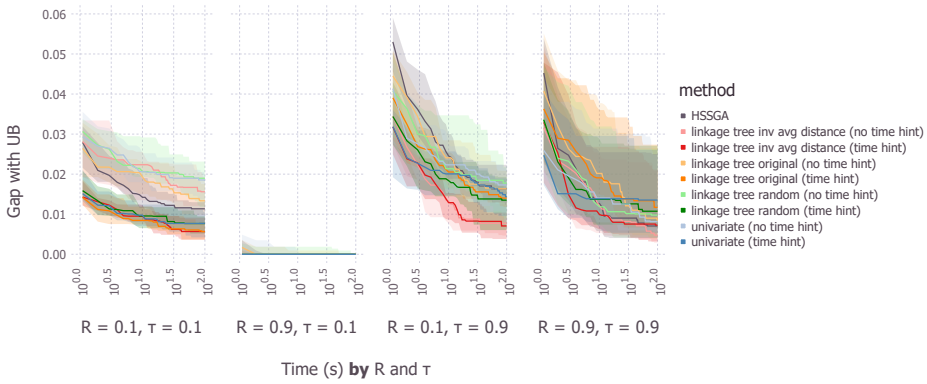


Figure 5.6: Effect of various Dependency Matrix formulas with local search and the Full Bounds hint on Cesaret instances. HSSGA is included for reference.

The key issue was seemingly caused by the usage of the random key encoding. As such dropping the usage of random keys the aforementioned issue may disappear. Yet using the original mixing approach for GOMEA without encoding the solutions comes with issues itself: an invalid solution may be constructed — this was discussed in Section 2.3 where encoding was defined.

Another element where the encoding formed a pain-point was the decoding procedure. Profiling the evaluation function pointed out that the a significantly larger portion of time was spent decoding the solution, rather than actually evaluating it.

Instead, we propose the use of the inverse permutation as encoding. Where a permutation lists the order at each position, its inverse lists the position for each order. This was chosen so that when computing the dependency matrix D this approach will be the same as the approach with random keys above. Like Random Keys the value in each position indicates an order's position.

Unlike Random Keys however, the positions are exact and no sorting needs to be performed to decode it. Decoding is the same operation as the encoding operation: taking the inverse permutation of the encoded solution. Due to the positions being exact, the issue with random swapping occur either.

These benefits do not come without their costs: the original mixing approach used in Discrete as well as Permutation GOMEA does no longer work. This issue is resolved by the use of a new mixing operator specifically designed for permutations defined as follows.

5.4.1. Mixing Operators

Information regarding permutations comes from various sources, in certain cases ordering may be important between particular elements — for example due to ordering constraints — while in others the exact position is important — for example due to time.

Rather than copying both over at the same time, the following operators focus on one of these aspects specifically. Allowing the approach to recombine pieces of information that would otherwise be lost.

These mixing operators operate on two solutions s and d. The first s is the solution which is the base, the solution onto which changes will be applied. These changes are based on the second solution d, the donor, and the subset of orders. In GOMEA this subset originates from the Family of Subsets model.

Positional Mixing

The first proposed crossover operator relates to the actual moving of orders similar to how this happens with random keys, except more deterministic. All orders from the subset are fixed in the position they have in the donor, the remaining orders are then filled into the holes using the order of the original solution.

As this operator originally both relocates and reorders — potentially destroying useful information in the ordering — a variant using a specially constructed donor is used instead.

This variant only relocates orders, preserving the original ordering and distances within the subset.

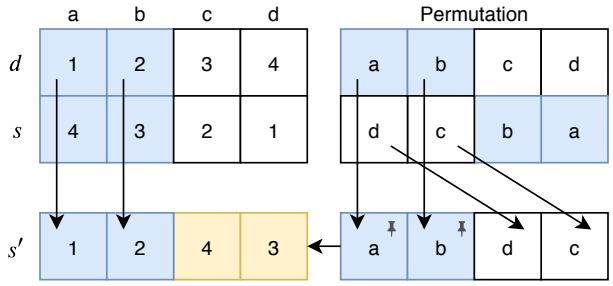


Figure 5.7: Positional Mixing applied on a subset $\{a, b\}$

Algorithm 2 Positional Mixing

```
1:  procedure MixPositional(s, d, b)                ⇨ Solution, Donor and Subset
2:      ŝ, ŝ′ ← [−1, ..., −1]                      ⇨ Initialize helpers with markers
3:      ŝ′_{d_i} ← i for every i ∈ b                ⇨ Inverse permutation for subset
4:      ŝ_{d_i} ← i for every i ∈ 1..n              ⇨ Inverse Permutation for inverse of subset
5:      i ← 1; j ← 1
6:      while i < n and j < n do                    ⇨ Replace markers in ŝ′ with non-markers in ŝ in order.
7:          i ← i + 1 repeat while ŝ_i = −1         ⇨ Seek past markers
8:          j ← j + 1 repeat while ŝ′_j ≠ −1        ⇨ Seek to marker
9:          ŝ′_j ← ŝ_i
10:         i ← i + 1; j ← j + 1
11:     end while
12:     return s′
13: end procedure
```

It does this by constructing a new donor d' on the basis of the donor d and the original solution s; and then performing the normal positional mixing operator on s with the newly constructed donor d'.

A reference point r is selected from the subset. This point directly inherits its position from the donor. For each remaining point in the subset its difference from the reference point is calculated, the new position for each order in the donor d' is then the location of the reference point in the donor plus the points difference:

$$d'_o = d_r - s_r + s_o \tag{5.3}$$

It may occur that an index goes out of bounds, if this is the case all keys are shifted so that none are out of bounds anymore[2]. The resulting donor will be referred to as the differential donor, due to the use of differences inspired by Differential Evolution.

This operator relocates a block of orders, but keeps it in the same order. The next operator performs the opposite operation: each order is reordered according to the donor, but the positions taken up are the same as those already used within the subset.

Order Mixing

The general idea of this operator is to transfer the ordering of elements within a subset without affecting the positioning of elements outside the subset, hence the name **Order Mixing**.

The goal is to be able to use variances in ordering independent of position in order to optimize, recombining the right order and position to obtain solutions — rather than requiring a solution to have both.

As any index outside the subset remains as-is, the act of mixing reorders the positional values of the solutions. This combined with the ordering requirement makes it so that the index with the smallest element within the subset of the donor will obtain the smallest index within the subset of the original

[2]As the difference is bounded by the length of the solution itself, it is impossible to both have an item that has a negative and an item with too large an index, nor will shifting them introduce a new item that goes out of bounds.

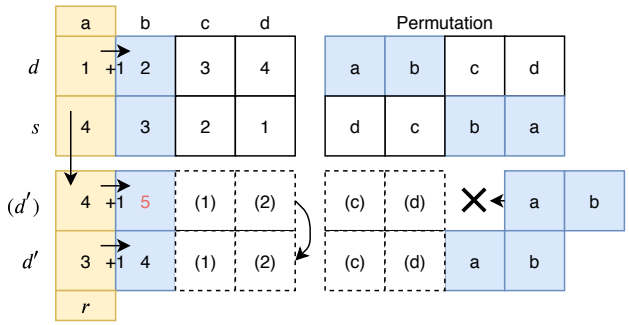


Figure 5.8: Construction of a differential donor D' on a subset $\{a, b\}$

Algorithm 3 Constructing the Differential Donor

procedure DIFFERENTIALDONOR(s, d, b) ⇨ Solution, Donor and Subset
  Initialize s' to be of size $|s|$
  $r \xleftarrow{\text{rand}} b$ ⇨ Select reference point
  $s'_i \leftarrow d_r - s_r + s_i$ for each $i \in b$
  $s'_{\min}, s'_{\max} \leftarrow \text{extrema}(s'_i \text{ for each } i \in b)$
  if $s'_{\min} < 0$ **then** ⇨ Invalid Indices before start (0)
    $s'_i \leftarrow s'[i] - s'_{\min}$ for each $i \in b$ ⇨ Shift up to get the smallest item to 0
  else if $s'_{\max} \geq |s|$ **then** ⇨ Invalid Indices after end
    $s'_i \leftarrow s'_i + n - 1 - s'_{\max}$ for each $i \in b$ ⇨ Shift down to get the largest item to $n - 1$
  end if
  return s' ⇨ s' is not guaranteed to be a valid solution outside of its subset.
end procedure

solution. This holds for the entire list, creating a mapping $\mathcal{M}$ which maps k-th smallest from the donor onto the k-th smallest of the original solution. What remains is performing the original mixing operator with $\mathcal{M}$ applied in order to obtain the replacement value for the target solution.

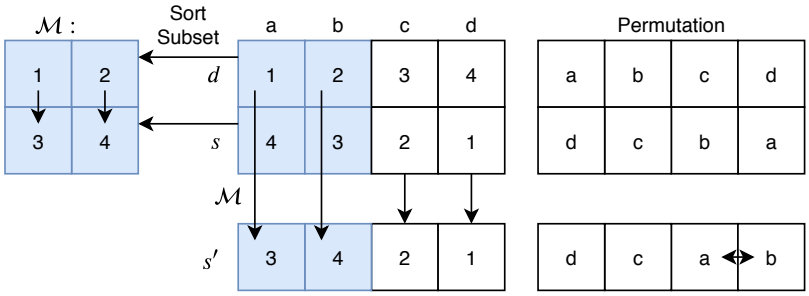


Figure 5.9: Order Mixing applied on a subset $\{a, b\}$

As this operator reorders the already present keys, it does not cause duplicates and can thus be used on the inverse permutation encoding.

Algorithm 4 Order Mixing

procedure MIXORDER(s, d, b) ⇨ Solution, Donor and Subset
  $\mathcal{M} \leftarrow \{x \to y \text{ for } x \in \text{SORT}(d_i \text{ for } i \in b), y \in \text{SORT}(s_i \text{ for } i \in b)\}$ ⇨ Construct order preserving mapping
  $s' \leftarrow s$ ⇨ Copy s
  $s'_i \leftarrow \mathcal{M}(s_i)$ for $i \in b$ ⇨ Use mapping to transfer order from donor
  return s'
end procedure

Algorithm 5 qGOMEA Replacements

function GETFOS(g) ⇨ In the case of Linkage Tree
  $D \leftarrow$ GETDEPENDENCYMATRIX(g) ⇨ Now using inverse average distance only
  $\mathcal{F}' \leftarrow \text{UPGMA}(D)$
  $\mathcal{F} \leftarrow \left\{f \mid f \in \mathcal{F}' \text{ if } |f| < n \cdot \frac{5}{6}\right\}$
  return $\mathcal{F}$
end function

function MIX($s, \mathcal{D}, \mathcal{F}$)
  for $b \in \mathcal{F}$ **do**
    $d \xleftarrow{\text{rand}} \mathcal{D}$ ⇨ Get a random donor
    for $o \in :d, :o$ **do** ⇨ Differential Positional- and Order Mixing
      if o is $:d$ **then**
        $d' \leftarrow$ DIFFERENTIALDONOR(s, d, b)
        $s' \leftarrow$ MIXPOSITIONAL(s, d', b)
      else if o is $:o$ **then**
        $s' \leftarrow$ MIXORDER(s, d, b)
      end if
      $s \leftarrow s'$ if $f(s') \geq f(s)$
    end for
  end for
end function

5.4.2. Overview of the Approach

Most of the qGOMEA is the same as the original GOMEA specification. Apart from the change in encoding — solutions are no longer random keys, and now are their inverse permutation — the mixing operators are now used to in place of the simple copy and random rescaling of solutions.

The dependency matrix is now calculated using the inverse of the average distance — as in Section 5.3 — and re-encoding is no longer performed.

The implementation only performs Order Mixing and Positional Mixing with the differential donor. Furthermore the subsets are filtered such that all sets smaller than $n \cdot \frac{5}{6}$ remain. Both Positional Mixing without differential donor and mixing with larger subsets reduce diversity in the population too much.

5.4.3. Experimental Setup

What impact does the change of operator have on the performance of the modified approaches themselves? As with the previous modifications they will be evaluated on the Cesaret benchmark instances.

In addition to this evaluation will be performed on various simpler permutation benchmark functions which are now described. Note that each of these have their input remapped in their experiments, for example reversed, to make the solution less obvious.

Initial runs indicate very fast premature convergence, hence similar to the procedure in the original Permutation GOMEA paper [7] an additional configuration with the Forced Improvement (FI) condition increased by a factor of 10 is added. Unlike this paper a configuration with Forced Improvement disabled entirely was added as well.

Functions

Number of Inversions Counts the number of inversions in a permutation π, where an inversion is a case where $\pi_i > \pi_j$ while $i < j$. The goal of this function is to minimize it and therefore has the ascending list $[1, 2, \ldots, n-1, n]$ as its corresponding optimum as it has no inversions. As the current setup maximizes the formula that is used for evaluation is as follows.

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \begin{cases} 1 & \text{if } \pi_i < \pi_j \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Number of Sequential Inversions An alternative version only takes into account the next item for computation of the objective and is otherwise identical to the original function.

$$\sum_{i=1}^{n-1} \begin{cases} 1 & \text{if } \pi_i < \pi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Number of Sequential Pairs Restricting the amount of information further is the following, which counts not decreases but increases in increments of 1. Unlike the previous functions this function should be maximized, and as such the formula below is the same as its definition.

$$\sum_{i=1}^{n-1} \begin{cases} 1 & \text{if } \pi_i + 1 = \pi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

5.4.4. Experimental Results

Performance on benchmark functions can be found in Figure 5.10 and is promising. While performance on the standard Inversions function is similar in the end; on the more complex functions qGOMEA outperforms GOMEA notably, especially the variants with the tenfold FI threshold and no FI.

A general speed improvement can be seen in the scalability graph in Figure 5.11. Larger problems are solvable in the same amount of time, but the degree of scalability seemingly remains the same to that of the original GOMEA as long as it does not prematurely converge. Unlike GOMEA, in qGOMEA no mutations occur and resolve this issue. Not performing Forced Improvement seems to avoid this issue, it would hence be recommended to not perform FI in these circumstances.

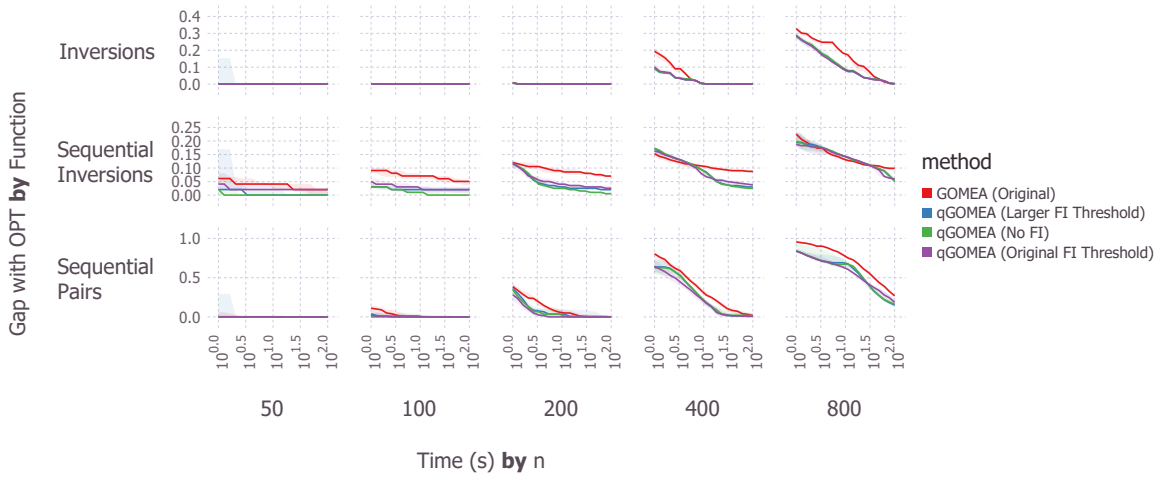


Figure 5.10: Performance of GOMEA and qGOMEA on various benchmark functions of various sizes over time

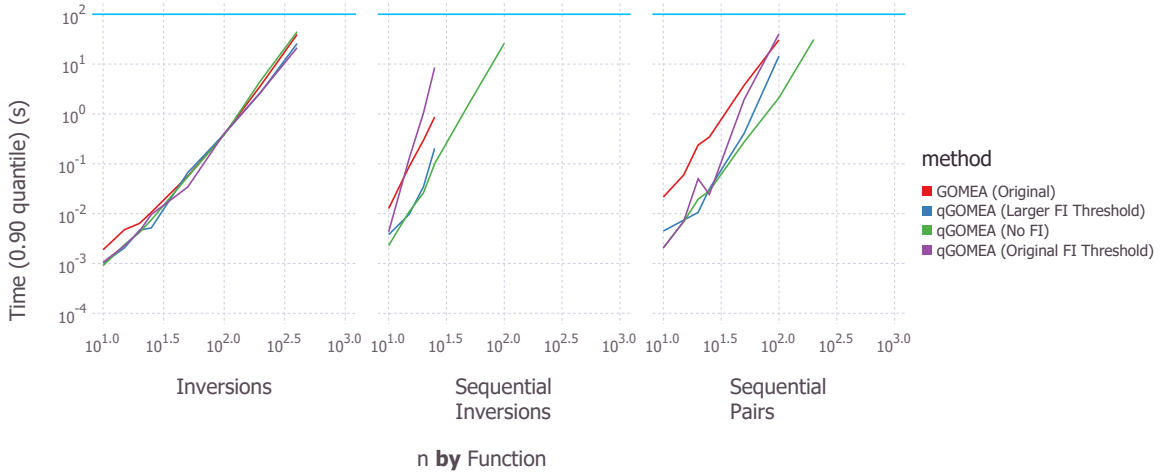


Figure 5.11: Scalability (time to optimum) of GOMEA and qGOMEA on various benchmark functions.

Experimental results on the Cesaret benchmark set for the OAS problem show a similar trend, as can be seen in Figure 5.12 and Figure 5.13 which provide no hints, and both the Time and Bounds hint respectively to both GOMEA and qGOMEA. HSSGA always uses its constructive heuristic and built-in time hint.

The most notable performance gain is for $R = 0.9$ and $\tau = 0.9$ where originally GOMEA required both hints, a modified dependency matrix and a local searcher to outperform HSSGA; the new approach only requires hints on shorter runtimes and outperforms otherwise.

Rather than patching up GOMEA's weaknesses with a local searcher, the introduction of new mixing operators and the following removal of randomness allowed for better performance across the board.

5.4.5. Conclusions

The application of a specialized mixing operator allows GOMEA to operate on permutations directly and achieve performance beyond that of its original configuration, and that of the state-of-the-art for the OAS problem, in many cases even when fully operated as a black-box approach. The modified approach no longer contains any mutations and is better able to solve permutation functions to optimality, as shown by previously problematic OAS instances and difficult benchmark functions.

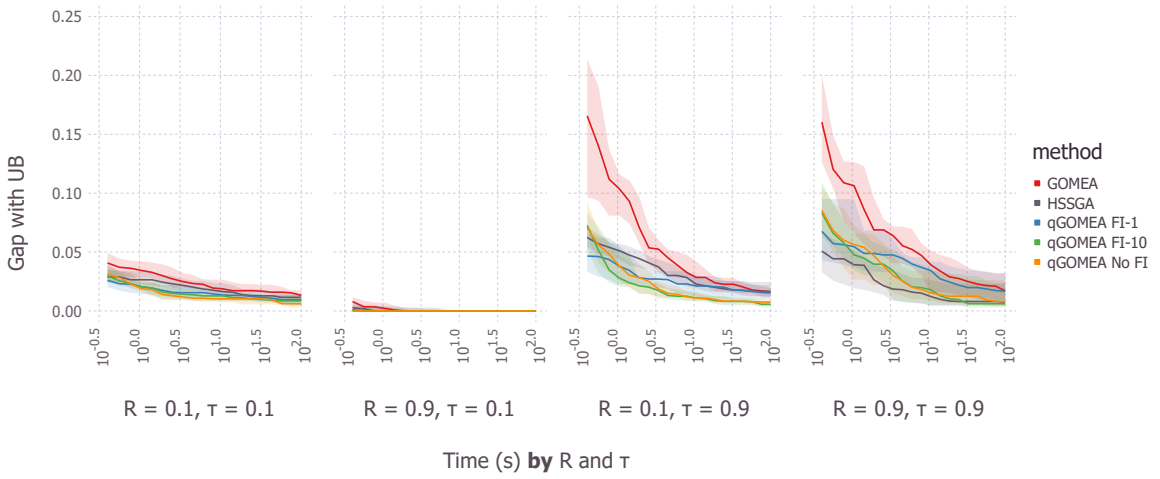


Figure 5.12: Performance vs Time on categories of OAS instances for HSSGA and GOMEA & qGOMEA without hints

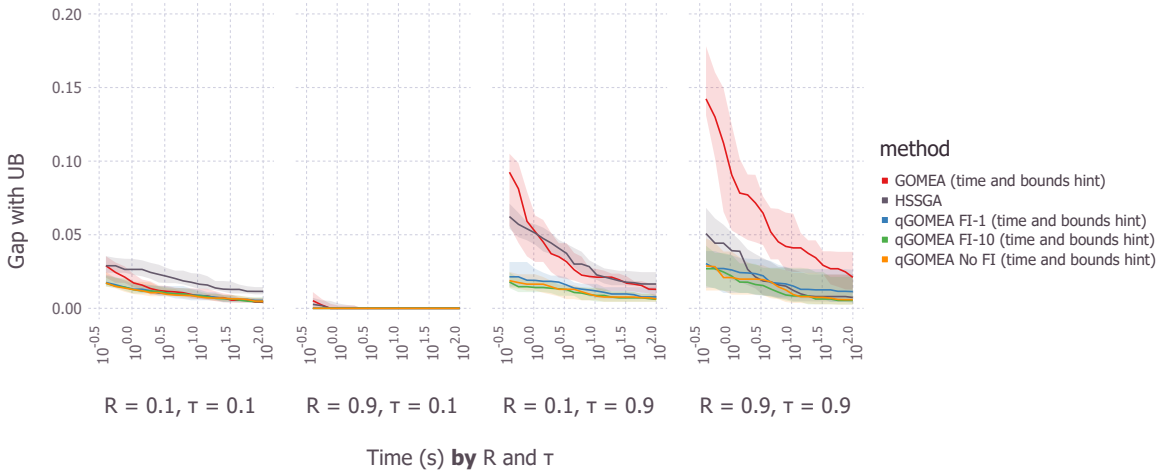


Figure 5.13: Performance vs Time on categories of OAS instances for HSSGA and GOMEA & qGOMEA with both Time and Bounds hint provided

5.4.6. Discussion

The current configuration of mixing operators is fixed; yet the actual usefulness of an operator may vary depending on the problem and the current state of the search. Inclusion of operator may negatively impact the performance of the approach. This was the case with the normal Positional Mixing operator on the instances it has been tested on, it caused the population to converge too quickly. Adapting their use could provide further benefits by reducing unnecessary mixing and therefore saving time.

The removal of the random key encoding and usage of special mixing operators makes finding the elements that have changed in some manner easier, opening up the possibility of partial re-evaluation. Given the small changes this approach occasionally will make this could provide a speedup, depending on the function itself. For the Inversions benchmark function more so than the OAS problem itself: the inductively defined time and the lower computational complexity of the evaluation function itself make it less likely to be faster.

Disabling Forced Improvement yielded improved results on many benchmark functions, allowing it to converge to the optimum in notably less time than alternatives. It does stop the approach from being able to exchange information between various populations, as the only mixing between solutions from different populations occurs within Forced Improvement. The separation could be something that negatively impacts performance: a larger population has to re-invent the wheel rather than using already known good solutions when starting with a new population. Alternatively the separation promotes exploration by not going down the same path of the already known to be quite good solutions. Whether this separation is a cause of the improved behaviour with respect to premature convergence instead of only the reduced variation caused by Forced Improvement remains an open question.

5.5. Summary

- The randomness of random keys has a negative impact on the performance of GOMEA.
- Use of a local searcher can improve performance and counteract the random effect of random keys. But not every evaluation of the objective function should invoke the local search procedure, it should rather be performed with a small probability in order to increase performance.
- A specialized local searcher does not necessarily add much value, while being a significant time investment to get working.
- The current formula used in order to compute the dependency matrix is not a good fit given the OAS problem. Modifying the formula to use only average distance is a better choice (which helps with the random keys as well!)
- The addition of hints remains useful to improve performance, even for different approaches, and when various modifications are applied.
- A better way to resolve the issues surrounding random keys is by not using them; and using special mixing operators to work with permutations themselves instead.
- The two proposed mixing operators focus on positioning and ordering respectively, while preserving the other as much as possible.
- The resulting approach is faster and works with a smaller population size.
- Additionally it can solve instances that previously required significant modifications and additions such as a local searcher in order to solve.
- Contrary to the previous approach, no mutations are present. Maintaining proper diversity is therefore more important.
- Forced Improvement causes diversity to drop too quickly, and hence should be disabled as it is in its current configuration.

6

Conclusions and Future Work

In this thesis the potential of black-box approaches was investigated in order to obtain better performing metaheuristics for the Order Acceptance and Scheduling with Sequence Dependent Setup Times problem. To accomplish this three approaches — Adaptive Differential Evolution, Generating Set Search and Permutation GOMEA — were analysed and evaluated on various benchmark sets. In order to allow these approaches to work on permutations the Random Key encoding was used. The primary benchmark used was proposed by Cesaret et al. while new additional benchmark sets were designed, covering missing cases. Missing cases include correlated time and value, and significantly more demand than capacity — requiring a smaller percentage of orders to be selected.

Approaches were tested and analysed on a variety of instances, providing useful insights into the strengths and weaknesses of each approach. In the comparative experiment there was one approach that stood above the rest. Permutation GOMEA outperformed all other approaches — including the state-of-the-art HSSGA — in many cases when supported by hints. But this was not the case for all instances: in the case of a small availability window, performance degraded significantly.

Scheduling an order at exactly the right moment proved difficult for GOMEA. This was caused in part by the Random Key encoding, which turned out to be a double-edged sword: it works well and encodes permutations in a natural manner, but it is also random in a way that makes precise positioning difficult.

Other shortcomings with GOMEA were identified as well, most notably an issue affecting Linkage Learning: the performance difference between GOMEA when using the Linkage Tree and the Univariate model was negligible, indicating that the expected gain in performance did not materialize.

Linkage Learning is performed by analysing the population using a statistical measure, the resulting value then corresponds to the amount of dependency between two variables. Changing the measure to only account for distance, as opposed to both ordering and distance, improved performance notably in many cases. Additionally, the randomness could be counteracted by using a strong point of HSSGA: its local search routines. Combining the improved Linkage Learning with occasionally using HSSGA's Swap Local Searcher allowed GOMEA to position orders more precisely, and outperform HSSGA – the state-of-the-art – on all standard Cesaret benchmark instances.

qGOMEA Yet the most notable improvement does not use HSSGA's local searcher. Rather than using Random Keys and patching up its side effects, this approach operates directly on permutations or permutation-like encodings and sidesteps the issues with the random keys entirely.

Modifying Permutation GOMEA to allow for this to happen required the use of an alternative mixing operator. Rather than choosing to only modify the original operator to avoid duplicates; two new mixing operators were introduced to ensure less information was lost in the process of mixing. The general idea behind the operators is that a solution is not just good because an order is at a specific position, but rather that a combination of relative positioning and ordering is responsible for the quality of a solution.

The first operator changes the position of orders, while keeping the relative positioning to a certain reference point. As the relative positioning stays the same, the ordering within the subset stays the same as well. This allows the position information to be defined as an offset to a reference point rather than an absolute value. This deals better with aspects such as sequential setup times.

The second operator focuses on transferring the ordering of the donor instead, preserving the positions used. This transfers over the information the previous operator kept as-is; covering the other side of the coin. This change provided significantly better performance over Permutation GOMEA on all instances.

The removal of diversifying operators such as re-encoding and random rescaling caused premature convergence to limit potential gains. Disabling Forced Improvement further improved performance, beyond that of the state-of-the-art in the remainder of cases.

This new approach improves upon the state-of-the-art in all cases for Order Acceptance and Scheduling with Sequence Dependent Setup Times, and improves upon GOMEA for more difficult variants of a general permutation benchmark function. When provided with better initialisation this extends towards all runtimes.

The improvements in qGOMEA over the original Permutation GOMEA are most notable in the case where ordering is stricter, for example in the case of dominating setup times and smaller time windows. In such a case qGOMEA can in half a second provide solutions with similar quality to that of GOMEA and the current state-of-the-art HSSGA, given 100 seconds of runtime.

Given qGOMEA's performance compared to Permutation GOMEA, as well as its performance on benchmarking functions, qGOMEA is applicable to many more problems.

6.1. Future Work

Applicability to Other Permutation Problems Setup times and their similarity to distances in Travelling Salesperson show that there are many more properties which occur in real world situations. While quite a few benchmark sets have been proposed, many of properties — such as setup times adhering to the triangle inequality — are of potential interest. For example using the transformation described in Appendix A TSP instances can be converted, furthermore improvements gained on such instances likely provide performance gains in more cases than just TSP, such as Vehicle Routing Problems. Such instances provide insight on the applicability of approaches such as qGOMEA on problems less aimed at the progression of time and more at ordering itself.

More Encoding Approaches Not all encodings are equivalent — the random key encoding, after encoding as a permutation with implicit selection, is only one option. How do different encodings that take the place of the random key encoding act? How do approaches perform when the act of selection is not implicit? Can an approach such as GOMEA make use of this selection information, and does it need to be explicitly part of the genetic information of a solution in order to do so? Many questions around encoding remain to be answered. A different approach to encoding could provide better performance, depending on the approach used.

More Solving Approaches The improved version of GOMEA, qGOMEA, draws inspiration from how the other approaches work. Testing a larger variety of solving approaches, especially those that focus specifically on permutations themselves, could potentially find traits and approaches for the the next set of improvements. Furthermore, more insight is obtained into what works well, and what does not, for a certain problem.

Improving qGOMEA The newly proposed qGOMEA can still be improved. Possible performance improvements be obtained through different usage of the mixing operators, different choice of formula for the dependency matrix, potentially even having a separate model for each operator is an option. Removal of the Forced Improvement operator — while necessary for good performance — was incorporated into the original GOMEA for a reason: speed. Can there be an alternative to Forced Improvement that provides similar benefits with less negative impact on diversification? Can information from previous populations be (better) utilized to improve upon newly generated populations?

Partial Evaluation The choice of encoding makes it easier to perform partial re-evaluation. For the given mixing operators the range of orders which have changed can be inferred quickly. This allows for re-evaluation of a small subset to happen, potentially quicker than evaluating the entire solution.

For the OAS problem this is not trivial: the change in time can affect the penalty incurred and the orders scheduled, even those outside of the sequence. Using the fact that the penalty incurred per

time unit and the set of scheduled orders do not change given the difference is small enough to not push an order past its due date or deadline, computational time can be saved.

Both diversification and partial evaluation impact the performance of qGOMEA, getting them right could provide further improvements to the approach.

Better upper bounds The benchmark instances of Cesaret et al. do not have tight upper bounds in all cases, causing any valid solution to have a gap larger than zero. This makes closing the gap further an impossible task. Furthermore there is an instance where the upper bound is wrong for $n = 50$, giving a negative gap in certain circumstances. When analysing the performance using an absolute point of reference these upper bounds are important and having all bounds tight and correct would help both close the gap and avoid confusion when comparing metaheuristics: it is difficult to compare performance when the upper bound used is different.

Expanding upon OAS While the OAS problem is quite general, many real world problems cannot be modelled sufficiently due to the absence of important elements, such as ordering constraints and having multiple machines to schedule. The proposed approach may therefore require additional modification depending on how such elements are incorporated into the problem itself.

In the presence of order selection, ordering constraints can be more complex due to a single question: “Can an order still be performed if the preceding order is not performed?”. It may be possible that an operation blocks another operation from happening afterwards, while not performing it does not stop the operation from happening beforehand.

For multiple machine scheduling there is the choice of how to encode this, for example using decision rule for assignment, splitting up the assignment in pieces or having a permutation and a discrete part of the encoded string.

Such aspects likely cause different behaviour in certain aspects of approaches, requiring modifications in order to work well. One particular complication for qGOMEA is “How to make linkage learning work well with multiple types of variables?”. This is especially relevant for cases where multiple machines are present and machine assignment needs to take place. For example in a Vehicle Routing Problem where vehicle assignment is done through discrete variables.

A Different Approach While an alteration to an existing approach was suggested in this thesis, the properties underlying permutations allow for more. In Baart’s thesis [4], decision diagrams are used in order to find the exact optimal solution. These decision diagrams bear a close resemblance to state machine, and state machines can be learnt from examples. A potential future approach to optimize over permutations could therefore use state machine learning techniques to model the underlying problem with an approximate state machine. Such a model could be used in an Evolutionary Distributional Algorithm to obtain another approach to optimizing permutations.

The proposed approach in this thesis is therefore certainly not the final stop for black-box optimization applied to permutation problems. There is much more potential to use this approach to problems, especially in the case of more complicated problems beyond Order Acceptance and Scheduling with Sequence Dependent Setup Times.

A

Mapping from TSP to OAS

This appendix describes a mapping from the Traveling Salesperson problem (possibly symmetric, but can also be asymmetric).

The general idea behind this mapping is that the time window needs to be large enough to accommodate any permutation of orders as time windows are not an element in the Traveling Salesperson problem, such a value is upper bounded by the sum of the worst processing times (which is named v). They all share the same time window as well, thus the release time for all orders is zero, and the due dates and deadlines are all equal to v to obtain such a large window.

The distances are mapped onto the setup times with processing times set to zero. Distance is hence mapped to time, but minimizing time is not the objective: it needs to be converted back into profit.

This is done by a special order q is specially set up to be always the last. By disallowing any orders after it being performed by setting the setup time to v and setting the value for each order high enough (to v) so that all permutations that schedule q last will be worth more than ones that do not. This is to make sure all orders are accounted for in the time before turning it into profit: this order q has its due date set to 0 instead so that a penalty is incurred for every time unit of its completion time.
More formally: Given a traveling salesperson problem described by its (positive) distance matrix D and set of cities C.

A specially related instance of Order Acceptance and Scheduling with Sequence Dependent Setup Times can be constructed as follows. (With 0 being the special 'initial state' for setup time)

Definitions:

$$v = \sum_{i \in C} \max_{j \in C} D_{ij}$$

$q = \text{first}(C)$ May also be arbitrarily selected from C

Instance:

$$r_i = 0 \qquad \forall i \in C$$
$$p_i = 0 \qquad \forall i \in C$$
$$d_i = v \qquad \forall i \in C - \{q\}$$
$$d_q = 0$$
$$\bar{d}_i = v \qquad \forall i \in C$$
$$e_i = v \qquad \forall i \in C$$
$$w_i = v \qquad \forall i \in C - \{q\}$$
$$w_q = 1$$
$$s_{ij} = D_{ij} \qquad \forall i \in C - \{q\}, j \in C$$
$$s_{qj} = v \qquad \forall i \in C$$
$$s_{0j} = D_{qj} \qquad \forall i \in C - \{q\}, j \in C$$

The instance is then described by release times r, processing times p, due dates d, deadlines $\bar{d}$, revenue e, weights w and sequence-dependent setup times s.

Lemma A.0.1. *A solution S_{TSP} (a sequence) with fitness f_{TSP} for TSP problem defined by explicit distance matrix D and set of cities C, has a corresponding solution S_{OAS} (a sequence) with fitness $f_{OAS} = -f_{TSP} + v|C|$ in the corresponding OAS instance as defined above.*

Proof. Circularly shift S_{TSP} so that the last city being visited is q, name this new solution to be S'_{TSP} with f'_{TSP} as its fitness. $f'_{TSP} = f_{TSP}$ as circularly shifting a TSP solution does not change the underlying encoding and fitness value.

Take S_{OAS} to be S'_{TSP}. The completion time of order i is defined to be $min(t, r_i) + s_{ji} + p_i$ with j the previous order. Initially $t = -\infty$ and j is the special order 0.

Whereas the total profit is defined to be $x_j + e_i - \max(0, (t_i - d_i)w_i)$, or x_j if $t_i > \bar{d}_i$. Where x_j is the previous profit and initial profit equal to 0.

As $r_i = 0$ and $p_i = 0$ in this case the function can be simplified to s_{0i} for the base case, and $t + s_{ji}$ for each increment. Expressing this in indices of D: $t_i = D_{qi}$ and $t_i = t_j + D_{ji}$. Time up to point i can therefore be expressed as $t_i = D_{qS^1_{OAS}} + \sum_{j=1}^{i-1} D_{S^j_{OAS}S^{j+1}_{OAS}}$. Note: $S^j_{OAS} \neq q$, as q is the last item.

All orders in S_{OAS} are performed before their deadline v. Since $D_{ij} \leq \max_r D_{ir} \forall j \in C$ (by definition of maximum) [1] and $v = \sum_{i \in C} \max_r D_{ir}$ [2].

To prove by induction: $t_i \leq \sum_{j=1}^{i} D_{jr}$ *Base case:*

$$t_1 = D_{qS^1_{OAS}} \leq \max_r D_{qr}[1]$$

Inductive case: Given $t_{i-1} \leq \sum_{j=1}^{i-1} \max_r D_{jr}[3]$ show that $t_i \leq \sum_{j=1}^{i} \max_r D_{jr}$.

$$t_i = t_{i-1} + D_{S^{i-1}_{OAS}S^i_{OAS}} \leq \sum_{j=1}^{i-1} \max_r D_{jr}[3] + \max_r D_{ir}[1] = \sum_{j=1}^{i} \max_r D_{jr}$$

As per the induction lemma, as well as the fact that S^i_{OAS} is a valid permutation (a given i only occurs once):

$$t_i \leq \sum_{j=1}^{i} D_{jr} \leq \sum_{j \in C} \max_r D_{jr} \leq v[1] \forall i \in C$$

As all orders except for q are therefore performed before their due date (v) as well, the formula for profit can be simplified to:

$$x_j + e_i = x_j + v$$

and

$$x_j + e_q - \max(0, (t_i - d_q)w_q) = x_j + v - t_i = x_j + v - (D_{qS^1_{OAS}} + \sum_{j=1}^{|C|-1} D_{S^j_{OAS}S^{j+1}_{OAS}})$$

As

$$f_{TSP} = f'_{TSP} = (D_{qS'^1_{TSP}} + \sum_{j=1}^{|C|-1} D_{S'^j_{TSP}S'^{j+1}_{TSP}}) = (D_{qS'^1_{OAS}} + \sum_{j=1}^{|C|-1} D_{S^j_{OAS}S^{j+1}_{OAS}})$$

The final profit is therefore

$$f_{OAS} = |C| \cdot v - f_{TSP}$$

□

Lemma A.0.2. *Given $\mathbb{S}_{TSP}$ and $\mathbb{S}_{OAS}$ are the sets of all feasible solutions for an TSP instance and its OAS mapping respectively, and $\mathbb{S}_{TSP \to OAS}$ to be the set of solutions obtained by mapping an $S_{TSP} \in \mathbb{S}_{TSP}$ to OAS using the procedure above.*

For a solution $S_{OAS} \in \mathbb{S}_{OAS}$ with fitness $f_{S_{OAS}}$ either $S \in \mathbb{S}_{TSP \to OAS}$ or $f_{S_{OAS}} \leq f_{S'_{OAS \to TSP}} \forall S' \in \mathbb{S}_{TSP \to OAS}$.

That is, a solution for the OAS problem either has respective feasible solution for TSP problem, or has strictly worse fitness than any solution that does.

Proof. For all solutions $S_{OAS} \in \mathbb{S}_{OAS}$ ending with order q, $S \in \mathbb{S}_{TSP \to OAS}$. As $\mathbb{S}_{TSP \to OAS}$ is the subset of all permutations ending in q by definition of the mapping itself.

Hence any solution $S \notin \mathbb{S}_{TSP \to OAS}$ does not end in q either. And at least one other order has to appear after it.

As $s_{qi} = v$, $t_i = t_q + v \geq v = \bar{d}_i$ for any order i after q. As i is therefore not performed/selected, $e_i = v$ is no longer part of the profit and thus at most $f_S = (|C| - 1) * v \leq |C| * v - f_{TSP}$ as $f_{TSP} \leq v$ by definition. □

Theorem A.0.3. *Maximizing f_{OAS} is the same as minimizing f_{TSP}.*

Proof. By Lemma A.0.2, all cases for which the reasoning in Lemma A.0.1 does not hold have worse fitness than in any case in which it does hold.

By Lemma A.0.1 $f_{OAS} = -f_{TSP} + v|C|$. Maximizing $-f_{TSP} + v|C|$ is the same as maximizing $-f_{TSP}$ as $v|C|$ is a constant. Maximizing $-f_{TSP}$ is the same as minimizing f_{TSP}. □

Uses

This mapping — apart from being an NP-hardness proof — also provides the possibility of generating another benchmark set. In these instances all time is spent is due to setup times thanks to the construction above.

Any TSP instance can be used to construct new instances. The instances from TSPLIB [23] are useful as they have the solution supplied. Alternatively generating some random Euclidean instances with some containing clustering traits by generating more points in a small normal distribution cantered on points.

Any fitness value (including upper and lower bounds) can be converted by using the formula $f_{OAS} = -f_{TSP} + v|C|$ from the proof of Theorem A.0.3. This allows for the computation of the gap with the optimum by using (fast) TSP specific solvers such as Concorde [3] or by using the already known optimum itself.

B

Parameter Configuration

Every approach has its parameters. When they are set well an approach can perform magnitudes better than when they are set poorly. A good comparison between approaches therefore should make sure that all approaches are configured properly: outperforming a badly configured approach is not special.

For each approach the parameters that require configuration are listed below.

B.1. Parameters

GOMEA Being a populated-based approach, GOMEA also has population size as a parameter. No experiments have to be ran for configuring this parameter however, the value itself is determined through the specialized procedure.

ADE Adaptive Differential Evolution conversely requires the population size to be set. Generally speaking this parameter is required to be large enough to avoid getting stuck in local optima. Conversely a large population makes an generation take longer. A trade-off needs to be found here. A base population size of around n is taken to strike a balance between these two, accounting for the time budget. Evaluating a solution is $O(n)$, together with the $O(n)$ population size this yields a total cost of $O(n^2)$ for evaluating the solutions in a generation. Which is of the same order as the time budget ($O(n^2)$).

ADE automatically determines the P and CR values used within the scheme. And henceforth do not require configuration.

GSS Generating Set Search has a starting step size and a pair of multipliers used on this step size, apart from that it has a minimum step size which indicates the point at which it has converged - at which point it restarts from a random point within the given bounds. The default configuration starts with a step size of 0.5 times the minimum search space diameter eg. 0.5 in the case of all bounds being between $[0.0, 1.0]$ and multiplies by $\Gamma = 2.0$ in case of success and $\Phi = 0.5$ in case of failure to find an improvement. It estimates convergence to be when the step size falls below $10^{-}10$.

These values do not necessarily fit this problem well. Convergence likely happens earlier than a step size of $\Delta\text{max} = 10^{-10}$. Whenever the step size is smaller than the smallest distance between two orders the resulting permutation - and therefore the objective value - will no longer change. As such we will have converged. Assuming n orders with their random keys (x, y) distributed according to an uniform distribution over $[a, b]$. A derivation shows that the expected minimum distance between items is according to Equation B.1.

$$E\left[\min_{x\in[1,n],y\in[1,n],x\neq y} |x-y|\right] = \frac{c-b}{n^2-n+1} \tag{B.1}$$

The process of searching however can cause a deviation with respect to this uniform distribution. Yet this formula does give the general order of magnitude around which convergence is very likely. For

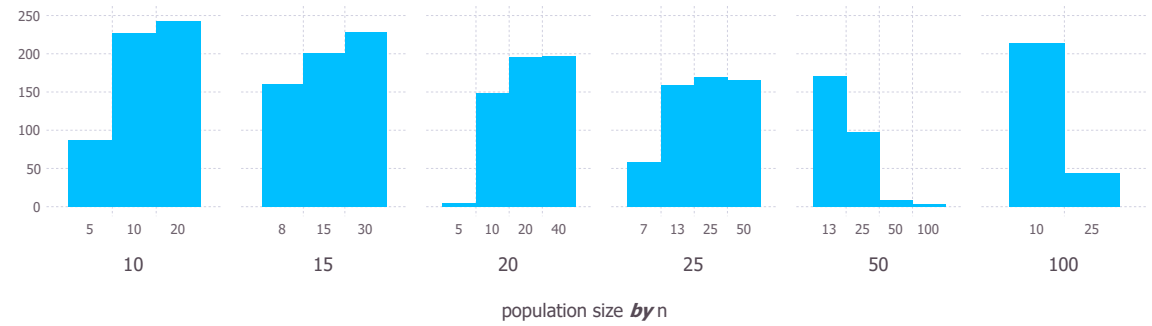


Figure B.1: Histogram of population sizes with highest found profit for ADE

100 orders between $[0, 1]$ this gives a value around $\Delta\text{max} = 10^{-4}$, a few orders of magnitude higher than the current default.

For the parameters Γ and Φ there is no clear good value. Making Γ too large and Φ too small will cause the search to switch between very large and small steps, while having them too small will cause a search with a medium step size.

For a randomly selected instance performance was generally better for $1 < \Gamma \leq 2$. $\Phi = \frac{1}{\Gamma}$ is a choice used by default and seems to work well in general, but it is interesting to take a value closer to 1 to search a bit more before restarting.

The experiment will therefore be ran with $\Delta\text{max} \in \left[\frac{1}{n^2}, 0.1 \cdot \frac{1}{n^2}, 10^{-10}\right]$, $\Gamma \in [1.01, 1.1, 1.5, 2.0]$, with $\Phi \in \left[\frac{1}{\Gamma}, \sqrt{\frac{1}{\Gamma}}\right]$. Due to the larger variability in objective values 10 runs are performed. The approaches are evaluated with a time budget of 10 seconds on instances of size 100. Due to time constraints 16 representative instances are selected rather than running all of them: from the Cesaret benchmark set the first instance for $\tau \in [0.1, 0.3, 0.7, 0.9]$ and $R \in [0.1, 0.3, 0.7, 0.9]$ is used.

B.2. Results

B.2.1. Adaptive Differential Evolution

Unlike GOMEA, Adaptive Differential Evolution is a population based approach that requires the population size to be configured manually. The parameters F and CR of Differential Evolution [21] are configured automatically using an adaptive scheme instead.

The results for population size are visualised in Figure B.1 as a histogram of population size(s) corresponding to the best found solution.

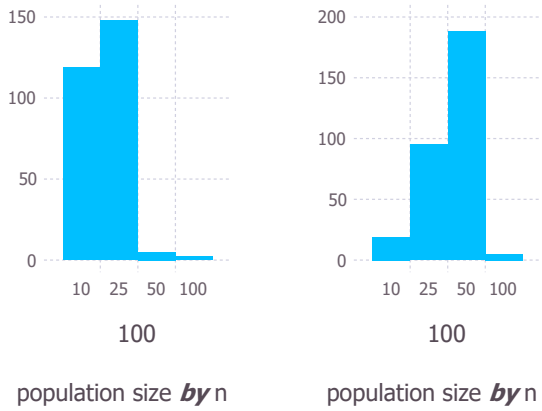
When given sufficient time, higher is better. For higher n the time budget is too restrictive and a lower population size performs better. A lower population size allows for quicker improvements by having more generations in exchange for getting stuck in local optima. Running with double the amount of time tells a similar story Figure B.2, the increase in time budget allows a larger population size to become better than a smaller population size.

B.2.2. Generating Set Search

Generating Set Search is not a population based approach and as such does not have a population size that needs to be configured. The parameters Γ, Φ and Δmax have been configured. As well as the tests regarding the hints again.

Tuning the parameters of GSS does not yield large differences in terms of performance. Bad values can lead to degradation of performance, for example for $\Gamma = 1.01$ in Table B.2. The variability between runs found is large, especially for the more difficult instances. Easy instances are solved to optimality within the time limit of 10 seconds with all parameter choices.

The effect of the parameter values is therefore limited. The effect is small and changing them does not necessarily lead to large improvements, especially in the case of seemingly easier instances.



(a) No Hints (b) Time and Bounds Hint

Figure B.2: Histogram of population sizes with highest found profit for ADE, with a doubled time budget. $n = 100$ only

Δmax	gap mean	gap median	gap max	gap 95	σ
0.0001	**0.0705817**	**0.025288**	**0.308417**	**0.204055**	**0.0737215**
$1.0 \cdot 10^{-5}$	0.0712718	0.0256408	0.335992	0.208044	0.075288
$1.0 \cdot 10^{-10}$‡	0.0762208	0.0286943	0.421011	0.221606	0.0813508

†$p < 0.01$, ‡$p < 0.001$
according to sign test compared against Δmax $= 0.0001$

Table B.1: Gap summary for various GSS configurations, grouped by Δmax

Γ	gap mean	gap median	gap max	gap 95	σ
2.0	**0.0677292**	0.0240341	**0.245546**	**0.195685**	**0.0705082**
1.5	0.0682037	**0.0222677**	0.257206	0.198749	0.0717987
1.1‡	0.0715398	**0.0222519**	0.280842	0.210041	0.0752084
1.01‡	0.0832932	0.0296816	0.421011	0.255537	0.0878529

†$p < 0.01$, ‡$p < 0.001$
according to sign test compared against $\Gamma = 2.0$

Table B.2: Gap summary for various GSS configurations, grouped by Γ

Φ	gap mean	gap median	gap max	gap 95	σ
$\sqrt{\Gamma^{-1}}$	**0.0710677**	**0.02594**	0.421011	**0.204735**	**0.0750152**
Γ^{-1}†	0.0743153	0.0270167	**0.326065**	0.215483	0.0786832

†$p < 0.01$, ‡$p < 0.001$
according to sign test compared against $\Phi = \sqrt{\Gamma^{-1}}$

Table B.3: Gap summary for various GSS configurations, grouped by Φ

B.2.3. Summary

- A good value for the population size is dependent on the instance kind and size, but also the method itself and the time budget. Using a procedure to automatically determine this parameter - like in GOMEA - is quite a bit more convenient, and potentially determines a population size that wastes less resources and performs better than a more generally chosen number.
- As long as the parameters for GSS are not chosen too badly performance on OAS has a low variance between configurations.

Bibliography

[1] GH Aalvanger, NH Luong, PAN Bosman, and D Thierens. Heuristics in permutation gomea for solving the permutation flowshop scheduling problem. In *International Conference on Parallel Problem Solving from Nature*, pages 146–157. Springer, 2018.

[2] Foto Afrati, Stavros Cosmadakis, Christos H Papadimitriou, George Papageorgiou, and Nadia Papakostantinou. The complexity of the travelling repairman problem. *RAIRO-Theoretical Informatics and Applications*, 20(1):79–87, 1986.

[3] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver. `http://www.math.uwaterloo.ca/tsp/concorde/index.html`, 2006.

[4] Robert Baart. Algorithms for a bounded-width order acceptance and scheduling problem with sequence-dependent setup time using decision diagrams. Master's thesis, Delft University of Technology, 11 2018.

[5] James C Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994.

[6] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[7] Peter AN Bosman, Ngoc Hoang Luong, and Dirk Thierens. Expanding from discrete cartesian to permutation gene-pool optimal mixing evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 637–644. ACM, 2016.

[8] Vicente Campos, Manuel Laguna, and Rafael Martí. Context-independent scatter and tabu search for permutation problems. *INFORMS Journal on Computing*, 17(1):111–122, 2005.

[9] Bahriye Cesaret, Ceyda Oğuz, and F Sibel Salman. A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, 39(6):1197–1205, 2012.

[10] Sachchida Nand Chaurasia and Alok Singh. Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem. *Applied Soft Computing*, 52:725–747, 2017.

[11] Robert Feldt. Blackboxoptim.jl. `https://github.com/robertfeldt/BlackBoxOptim.jl`, 2018.

[12] James E. Gentle. *Computational Statistics (Statistics and Computing)*. Springer, 2009. ISBN 780387981444.

[13] Ilan Gronau and Shlomo Moran. Optimal implementations of upgma and other common clustering algorithms. *Information Processing Letters*, 104(6):205 – 210, 2007. ISSN 0020-0190. doi: https://doi.org/10.1016/j.ipl.2007.07.002. URL `http://www.sciencedirect.com/science/article/pii/S0020019007001767`.

[14] Nicholas G Hall and Chelliah Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525, 1996.

[15] N. Hansen, S.D. Muller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1): 1–18, 2003.

[16] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696. ACM, 2010.

[17] Tamara G Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[18] Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Recent developments in deterministic sequencing and scheduling: A survey. In *Deterministic and stochastic scheduling*, pages 35–73. Springer, 1982.

[19] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[20] Ceyda Og, F Sibel Salman, Zehra Bilgintürk Yalçın, et al. Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, 125(1):200–211, 2010.

[21] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[22] A Kai Qin and Ponnuthurai N Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1785–1791. IEEE, 2005.

[23] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384, 1991.

[24] Yuri Laio TV Silva, Anand Subramanian, and Artur Alves Pessoa. Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & Operations Research*, 90:142–160, 2018.

[25] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[26] Anand Subramanian and Katyanne Farias. Efficient local search limitation strategy for single machine total weighted tardiness scheduling with sequence-dependent setup times. *Computers & Operations Research*, 79:190–206, 2017.

[27] Dirk Thierens and Peter AN Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 617–624. ACM, 2011.

[28] Roel W van den Broek. Train shunting and service scheduling: an integrated local search approach. Master's thesis, 2016.

[29] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE, 2008.

[30] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.