UTRECHT UNIVERSITY

MASTER THESIS IN ARTIFICIAL INTELLIGENCE

Comparison between Permutation GOMEA and Biased Random-Key Genetic Algorithm for the Winner Determination Problem in Multi-Combinatorial Auctions

Author:
Andrea Scorza

Student Number:
6649173

Supervisor:
Dr. ir. Dirk Thierens

May 24th, 2021

Abstract

This study presents a comparison between the Gene-pool Optimal Mixing Evolutionary Algorithm for permutation problems (GOMEA) and the Biased Random-Key Genetic Algorithm (BRKGA). The performance of the two algorithms is evaluated on the Winner Determination Problem in Multi-Combinatorial Auctions. The purpose of the study is to understand how well, the linkage tree model on which GOMEA relies, is able to capture and explain the structure of the problem and how well it performs when compared to an algorithm that's already been proven excellent in solving the aforementioned problem as the BRKGA.
To test the algorithms problem instances are generated using the Combinatorial Auction Test Suite (CATS). The test suite is able to generate ad hoc problem instances varying in terms of dimension, hardness, and distribution.
Additional FOS models like the Univariate model, for the GOMEA algorithm, are tested and evaluated. GOMEA is also tested with the RKGA and Ordering Messy Genetic Algorithm (OMEGA) on deceptive permutations problems.
Results show that GOMEA performs well, especially on harder problem instances when compared to the BRKGA, even though the linkage tree model doesn't fully represent the problem structure.

Contents

1 Introduction

An auction is a process in which goods or property are sold to the highest bidder. A bidder is an individual or an organization who makes a formal offer, called a bid, in order to buy an item at an auction. The whole process is regulated, in traditional auctions, by the auctioneer, who conducts the auction by accepting bids and declaring items sold. There are many different types of auctions like the English auction, the Japanese auction, the Dutch auction, the Sealed-bid auction, etcetera.
All the different types of auctions can be grouped into three categories:

- Single unit auctions, where one good is involved
- Multi-unit auctions, where more tokens of the same goods are involved
- Combinatorial auctions, where more tokens of different goods are involved

During this research, the auction that will be examined is a combination of the last two types of auctions, the multi-combinatorial auction. [1]

1.1 Problem definition

Multi-combinatorial auctions are a specific type of auctions where a bid consists of a set of items rather than a single item. The Winner Determination Problem (WDP) is an NP-hard problem by reduction from the weighted set packing problem [2].
The objective of this combinatorial optimization problem is to maximize the auctioneer's revenue, by choosing an allocation that maximizes the sum, over all bidders, of the bidders' valuation for the subset of items that they receive [3].

The set of bidders is denoted by $N = \{1, ..., n\}$ and the set of items is denoted by $M = \{1, ..., m\}$.

A bundle S is a set of items: $S \subseteq M$. The bid that bidder i makes for bundle S is denoted by $v_{\mathrm{i}}(S)$. The allocation of the items is described by the variable $X_i(S)$:

$$X_{\mathrm{i}}(S) \in \begin{cases} 1, & \text{if bidder } i \text{ gets bundle } S \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

An allocation is said to be feasible if it allocates no item more than once:

$$\sum_{i \in N} \sum_{S \subseteq M, j \in S} x_i(S) \leq 1, \text{for all } \; j \in M, \tag{2}$$

and at most one subset to every bidder:

$$\sum_{S \subseteq M} x_i(S) \leq 1, \text{for all } \ i \in N. \tag{3}$$

Given the bids v_i, $i = 1, ..., n$, the Winner Determination Problem is defined by:

$$x = \text{argmax} \left(\sum_{i \in N} v_i(S) x_i(S) \mid x \text{ is a feasible allocation} \right). \tag{4}$$

For a large number of bids, it is unfeasible for an algorithm to check every possible combination of bids, due to the exponential nature of the problem. Since any subset of bids is allowed there are $n(2^m - 1)$ bids with n for the number of bidders and m for the number of goods. During the scope of this research, the auction in exam will be a single-round auction, with non-negativity constraints. A single-round auction differs from a multi-round auction as the bids will be placed only once throughout the auction.

1.2 Affinity with the Multi-Dimensional Knapsack problem

There are many studies that report a similarity between the Multi-Dimensional Knapsack Problem (MDKP) and the Winner Determination Problem in combinatorial auctions [4, 5, 3, 6] in particular they affirm that the WDP can be seen and modeled as the MDKP.
Following a study by Holte [5] the multi-unit combinatorial auction is precisely a MDKP, where each item in the auction is a dimension and accepting a bid corresponds to inserting an item into the knapsack. In their studies, they focused on the performance of hill-climbing algorithms on combinatorial auctions which includes both problems. The different types of hill-climbing algorithms were found to score particularly well for both the MDKP and the WDP.
Kelly proposed another important study that shows the relations between the two combinatorial problems [7]. It starts with the data center allocation problem. In this problem, there is a constant number of few recourse types and a large number of units of these resources that need to be allocated, which is comparable to the WDP. It is then intuitive to generalize the optimal allocation problem as an MDKP.
Lehmann et al. [3] modeled the WDP_{XOR}, that corresponds to the multi-dimensional multiple-choice knapsack problem, by an integer linear program that is identical to the one for the weighted set packing problem which objective is to find a non-intersecting set of maximal total weights [2].

$$\begin{array}{cll} \max & \sum_{i=1}^{n} \sum_{S \subseteq M} v_i(S) x_i(S) & \\ (WDP_{XOR}) & \sum_{i=1}^{n} \sum_{S \subseteq M, S \ni j} x_i(S) \leq 1 & \text{for all } j \in M \\ & \sum_{S \subseteq M} x_i(S) \leq 1 & \text{for all } i \in N \\ & x_i(S) \in \{0, 1\} & \end{array}$$

It was previously suggested in the literature to interpret a multi-item version of the WDP_{XOR} as a generalized knapsack problem [5]. In the general KP, there is a set of objects with respecting weights and values and the algorithm has to maximize the total value by selecting a set of objects respecting a weight threshold, whereas in the case of the WDP_{XOR} it is a multi-dimensional multiple-choice knapsack problem.

2 Previous works

Numerous meta-heuristics approaches have been undertaken for solving the winner determination problem, in her work, Boughaci, [8] compares four of these approaches both single oriented, like Stochastic local search and tabu search, and population oriented, like genetic algorithms and memetic algorithms.
The different algorithms were implemented and tested on various benchmarks. The Memetic Algorithm (MA), which is a population-based approach that combines local search methods with crossover operators, resulted more efficient than the rest of the algorithms. The reason for this improved efficiency resides in the stochastic local search component that allows the crossover operator to create better solutions.

The Biased Random-Key genetic algorithm has been proven successful for the winner determination problems in multi-combinatorial auctions [4]. The algorithm uses standardised chromosome encoding which consists of a vector with t uniformly drawn random keys (alleles) over the interval [0, 1] [9] and parametrised uniform crossover.
Different approaches are examined: Chromosomal approach, Greedy approach and, Surrogate duality approach, all of them with an LP-relaxations variant. They compared the algorithm with two exact and two heuristic algorithms, in both cases state-of-the-art algorithms. The newly proposed algorithms outperformed the competitors, and between them, the version using LP-based initialisation performed better than the approaches that used instead of a random vector as initial population.

The biased Random-key Genetic algorithm has been used to solve various combinatorial optimization problems as resource-constrained multi-project scheduling problem, the unequal area facility layout problem, 2D, and 3D bin packing problems etcetera.
In their work, Gonçalves et al. [10] showed several applications for the algorithm which in the vast majority of cases performed better than the other GAs or heuristics. They were able to easily test it for 13 different problems due to its adaptability. The algorithm is divided into problem dependent and problem independent parts. For what concerns the problem independent parts, with the BRKGA one does not have to worry about crossover operators and mutation because they are already specified contrary to what happens in a normal genetic algorithm.
The problem-dependent part of the algorithm is merely the fitness function hence it becomes straightforward to incorporate a heuristic algorithm with the BRKGA obtaining a better solution than the single heuristic. The BRKGA is an improvement of the random-key GA proposed by Bean [9] whereas one of the parents is always chosen from the elite set. This adds to the greediness of the algorithm instead of pure randomization which considerably increases the algorithm performances.

The relatively new introduced Gene-Pool Optimal Mixing Evolutionary Al-

gorithm (GOMEA), demonstrated its capabilities of scaling excellently on Cartesian-space optimisation problem. In a recent work by Bosman et al, [11] they demonstrated that using GOMEA in combination with the Biased Random-Key genetic algorithm (BRKGA) could also be used to solve permutations optimisation problems. One of the main advantages of the algorithm is the use of the family of subsets (FOS) concept that captures and encodes dependencies between problem variables.
The FOS model used by the algorithm is the Linkage Tree model, which is able to capture both low and high order dependencies, in this case, using bottom-up hierarchical clustering. The algorithm proved itself state-of-the-art with the Permutation Flow-shop Scheduling problem as a benchmark.

Not only BRKGA is positively advantaged by using a heuristic for the population initialization, but Gene-Pool Optimal Mixing Evolutionary Algorithm as well. Precisely in their work, Aalvanger et al. [12] showed how the algorithm can be improved by incorporating a constructive heuristic to seed the initial population and compare it to the state-of-the-art algorithm for the Permutation Flowshop Scheduling Problem.
As heuristic it was utilized the LR(x) [13] which simply works in three steps: It first sorts the jobs according to an index function, then creates a list of schedules, and finally, it picks the best schedule generated so far. There are two different types of index functions, the first one penalizes the idle time of the machines, the second one rewards the sum of completion times.
The algorithm is tested against the Variable Neighbor Search Algorithm (VNS4) [14], one of the most successful algorithms in literature for PFSP, on three different types of instances. Job correlated instances, where processing times are dependent on the job, and not on the machines, Machine correlated instances, where the structure is reversed and Mixed correlated instances which are equal to MCI but with the difference of the processing times of each machine to be job-dependent.
Results show that for job-correlated and mixed correlated instances, Permutation GOMEA is able to higher benefits from the problem structure and always outperform the VNS4 algorithm. The same is not encountered for machine-correlated instances with a high amount of structure where the VNS4 algorithm performs better. This is most likely due to the distance measure used to build the linkage model that fails to fully capture the structure of the problem. Overall results show that seeding the initial population with a heuristic for Permutation GOMEA is an effective technique.

In their work, Bosman et al. showed how local search can improve the results even in a black-box optimisation where partial evaluations are not possible [15]
.
Optimal mixing was also examined, and taken into consideration as an integration of local search into the variations operators of evolutionary algorithms, which results in a better combination compared to generic memetic algorithms. Local search was found to be particularly useful when dependencies between

problem variables were identified using the model building.

The Linkage Neighbour, a variant of the Linkage tree model, has been introduced by Bosman et al. [16] where even with the simplest learning approach better results are obtained on the linkage benchmarks problem compared to the LT model. The main difference between the two linkage models resides in the ability of the LN to model overlapping building blocks. The LT model is still able to model overlapping linkage relations but these are organised in a hierarchical way where lower-order constituents of a linkage set are mutually exclusive.
With the LN, which to determine a linkage neighbour uses the likelihood-ratio test, it is possible to singularly represent linkage neighbour for a determined variable singularly, instead of clustering all the variables together as it would happen for the LT. Moreover, they extended the LN to a multi-scale variant that combines the benefits of the LN in terms of overlapping blocks and the benefits of the LT in terms of representing different scales of linkage.
The Multiscale hierarchy is obtained by reducing the size of the neighbourhood with a specific ordering, that differs for different decomposition, creating inclusive subsets. This method had proven beneficial compared to representing linkage at single scale.

Another solution to overlapping problems in linkage learning has been proposed by Przewozniczek et al. [17] in their Parameter-Less Population Pyramid for permutation-based problems (P4). The main difference from other evolutionary methods is the structure of its population. Individuals from subpopulations, called levels, have their separate linkage information, and together they form the pyramid structure. Each pyramid level has its separate Dependency Structure Matrix (DSM) and a separate linkage tree.
At each iteration a new individual is crossed with each individual of the pyramid using the optimal-mixing operator and if it improves it is added to the higher level of the pyramid. The method is tested against LT-GOMEA using Permutation Flowshop Scheduling Problem as the comparison base.
It is found that for a higher number of jobs and a small number of machines, LT-GOMEA outperforms P4, however, the LT-GOMEA dominance vanishes with the increasing numbers of machines. This is due to maintaining a smaller number of linkages, hence P4 proves to be successful in preserving better diversity.

2.1 BRKGA

Genetic algorithms that utilize random keys, known as random-key genetic algorithm (RKGA) were first introduced in 1994 by Bean [9]. The algorithm was proposed to solve combinatorial optimization problems, where the solution can be represented as a vector of randomly generated real numbers in the $[0, 1]$ interval.
Each individual of the population is represented as a vector of random keys

and a deterministic algorithm called *decoder*, takes as an input the individual, which is the vector of random keys, and associates it to a feasible solution in the combinatorial optimization problem space. The decoder computes the solution's fitness and returns its value; the higher the fitness, the better the solution.

The algorithm starts with the random initialization of the population. The population is composed of p individuals which are encoded as vectors of random keys. At each generation, until the stopping criteria is met, the population is evaluated and a new population is formed for the next generation. At the beginning of each generation, the fitness of every individual is calculated using the decoder. Based on their fitness, the individuals of the population are then divided into two groups, a small group of elite individuals, which a higher fitness value, p_e, and the remaining non-elite individuals. The elite individuals from generation k are copied unchanged to generation $k+1$.
To introduce diversity in the population, some mutants are added to generation $k+1$, those elements, p_m are newly randomly generated individuals. The remaining $p - p_e - p_m$ individuals are the offspring created through the crossover process.

RGKA selects the parents for the crossover at random from the entire population and allows them to be selected more than one per generation. The difference with the Biased RKGA lies in the parent selection; one parent (ParentA) is selected from the elite set and the other parent (ParentB)from the remaining population [4], parents can still be selected more than once per generation.
Both versions of the algorithm implement parameterized uniform crossover, where p_a indicates the probability that the offspring inherits the vector component from ParentA, and $1 - p_a$ is the probability of inheriting ParentB's vector component. If $p_a = 0.5$ then it's the case of the standard uniform crossover. This increases exploitation since with $p_a > 0.5$ there's a higher probability that the offspring will inherit from the elite parent.
Once the population is complete a new generation will start and the iteration will continue until the stopping criteria is met; the algorithm returns the best solution found.

2.2 GOMEA

The Gene-pool Optimal Mixing Evolutionary Algorithm is a population-based, stochastic search algorithm. It is able to efficiently learn dependency information between variables through a Family-Of-Subset model (FOS), it efficiently decides between competing building blocks and transfers the optimal ones from the parents to the offspring solution.
It starts by creating a random population of individuals, and subsequently, the generation cycle starts and will terminate once specific conditions are satisfied. The generation is divided into two steps: generating a FOS model and Optimal Mixing.

2.2.1 Family Of Subsets model

The objectice of the FOS model is to identify groups of problem variables that together make an important contribution to the quality of solutions. These variable groups interacts in a non-linear way and should be processed as a block which will be called building block. The FOS model should capture the structure of the problem which can only be approximated from the knowledge in the current population and provide the building blocks to construct the solution through optimal mixing of the population. There are different types of FOS models of different sizes and there is a trade-off between size and information. A bigger FOS model with more building blocks means more times and steps for the algorithm to perform the Optimal Mixing, but a smaller FOS model is not always able to capture dependencies hence fewer informations.

- Univariate FOS structure: it assumes every variable is independent from the others and the model itself is a set of singleton whose union corresponds to the set of all variables: for n vairables: $F = \{\{x_1\}, \{x_2\}, ...\{x_n\}\}$. It is the simplest FOS model.
- Marginal Product FOS structure: it creates mutually independents sets of variables. These sets are formed by variables that possess a certain degree of dependency between each other. The sets are not overlapping and their union form the sets of all variables: for n variables: $F = \{\{x_1, x_4\}, ...\{x_8, x_9, ..., x_n\}\}$
- Linkage Tree FOS structure: it is a binary tree that has the set containing all variables as root and singletons of all variables as leaves. Subsets or branches are composed of sets of variables that share dependency between each other. Problem variables in a subset are considered to be dependent on each other but become independent in a child subset. The Linkage Tree has l leaf nodes and $l-1$ internal nodes; l corresponds to the number of problem variables. To create the linkage tree, the first step is to create the univariate structure and after that, it builds the linkage tree using bottom-up hierarchical clustering algorithm.

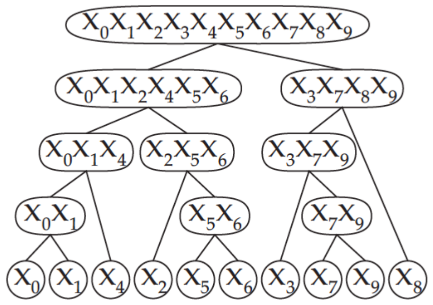


Figure 1: Exmple of linkage tree with 10 problem variables

In this work, the univariate and the linkage tree FOS will be analyzed.

2.2.2 Optimal mixing

Optimal mixing (OM) can be regarded as a greedy improvement of existing solutions. The term optimal mixing comes from the fact that better instances for substructures are immediately accepted and are not influenced by the noise coming from other parts of the solution.
After the FOS structure is completed, the second phase of the generation can start. For each individual of the population, a donor solution is randomly selected from the population. The individual has to be recombined with the donor once for every subset of the linkage tree. The subset is used as a crossover mask, and the recombination is greedy; only improvements of the individual are accepted.

Example of recombination:

$$\begin{aligned} &Donor &&= [0,0,0] \\ &Individual &&= [1,1,1] \\ &Subset &&= [x_2, x_3] \\ &NewIndividual &&= [1,0,0] \end{aligned}$$

In this study, Gene-pool optimal mixing (GOM) will be analyzed, which differs from the previously introduced Recombinative optimal mixing (ROM) [18], since for each subset of the FOS model a new donor is randomly selected while for the ROM a single donor solution was selected to perform OM, for each individual of the population.

Once the algorithm has traversed entirely the FOS model, a new generation starts, a new model is built, and the iterations continue. The algorithm stops when it meets the termination criteria; the best solution is then returned.

2.3 Deception and genetic algorithms

Deceptive problems are misleading for genetic algorithms because they are structured in such a way that low-order building blocks that contain suboptimal solutions are distant from the global solution. Genetic algorithms work by recombining low-order building blocks and, deceptive functions are able to attract the algorithm towards a local optima instead of the global.

For example, the deceptive trap function, Counting Ones. The goal is to maximize the function value of a binary vector; the global optimum is defined as the vector of only 1.

Counting Ones function: $x_i \in \{0,1\} : CO\left(x_1 \ldots x_\ell\right) = \sum_{i=1}^{\ell} x_i$

The vector is divided into low-order building blocks of length $k = 4$ which are subfunctions containing two optimum, 1111 and 0000. Since the trap function consist of $100/4 = 25$ concatenated subfunctions, it contains $2^{25} - 1$ local optima and one global optima which is the string of all ones.

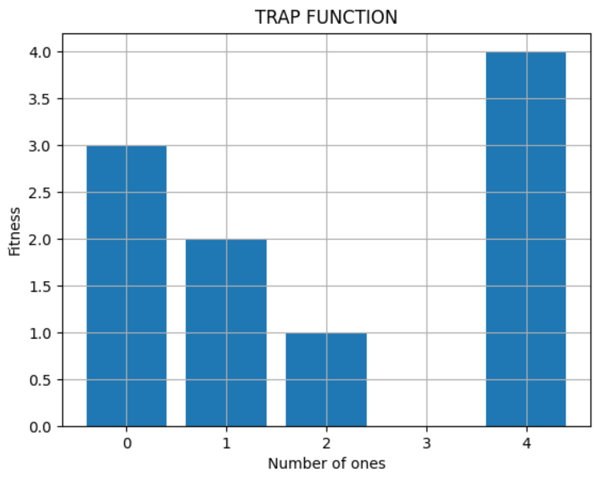


Figure 2: Distribution of fitness and the two optima of the trap function

In deceptive functions, for each subfunction, the lower-order schema fitness averages that contain the global optimum 1111, have a lower value than the lower-order schema fitness averages that contain the local optimum 0000.

$$\begin{cases} F(111*) = 2 \\ F(000*) = 2.5 \end{cases}$$

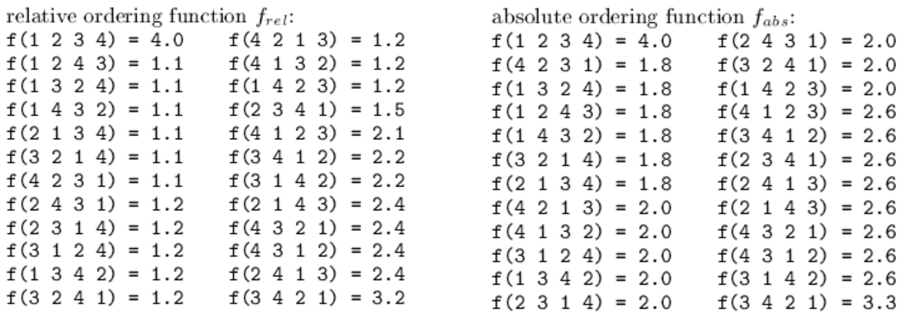
$$\begin{cases} F(11**) = 1.25 \\ F(00**) = 2 \end{cases}$$

$$\begin{cases} F(1***) = 1.125 \\ F(0***) = 1.5 \end{cases}$$

2.3.1 Ordering Messy Genetic Algorithm

In their study, Knjazew et al. compared the Ordering Messy Genetic Algorithm (OMEGA) which is a fast messy genetic algorithm that uses random keys to represent chromosomes, and the Biased Random Key Genetic Algorithm, on deceptive combinatorial optimization problems. [19]. These problems come from the work of Karugpta et al. [20] where they defined two deceptive functions of order 4: the relative ordering function f_{rel}, and the absolute ordering function

f_{abs}. The problem is a 32-allele permutation problem in which 8 size 4 permutation sub-problems are coupled together
The functions are defined as follow:



relative ordering function f_{rel}:

```
f(1 2 3 4) = 4.0    f(4 2 1 3) = 1.2
f(1 2 4 3) = 1.1    f(4 1 3 2) = 1.2
f(1 3 2 4) = 1.1    f(1 4 2 3) = 1.2
f(1 4 3 2) = 1.1    f(2 3 4 1) = 1.5
f(2 1 3 4) = 1.1    f(4 1 2 3) = 2.1
f(3 2 1 4) = 1.1    f(3 4 1 2) = 2.2
f(4 2 3 1) = 1.1    f(3 1 4 2) = 2.2
f(2 4 3 1) = 1.2    f(2 1 4 3) = 2.4
f(2 3 1 4) = 1.2    f(4 3 2 1) = 2.4
f(3 1 2 4) = 1.2    f(4 3 1 2) = 2.4
f(1 3 4 2) = 1.2    f(2 4 1 3) = 2.4
f(3 2 4 1) = 1.2    f(3 4 2 1) = 3.2
```

absolute ordering function f_{abs}:

```
f(1 2 3 4) = 4.0    f(2 4 3 1) = 2.0
f(4 2 3 1) = 1.8    f(3 2 4 1) = 2.0
f(1 3 2 4) = 1.8    f(1 4 2 3) = 2.0
f(1 2 4 3) = 1.8    f(4 1 2 3) = 2.6
f(1 4 3 2) = 1.8    f(3 4 1 2) = 2.6
f(3 2 1 4) = 1.8    f(2 3 4 1) = 2.6
f(2 1 3 4) = 1.8    f(2 4 1 3) = 2.6
f(4 2 1 3) = 2.0    f(2 1 4 3) = 2.6
f(4 1 3 2) = 2.0    f(4 3 2 1) = 2.6
f(3 1 2 4) = 2.0    f(4 3 1 2) = 2.6
f(1 3 4 2) = 2.0    f(3 1 4 2) = 2.6
f(2 3 1 4) = 2.0    f(3 4 2 1) = 3.3
```

Figure 3: Definition of the function from the work of Karugpta et al. [20]

In the relative ordering problem, the relative ordering of the allele is important, while in the absolute ordering problem, not only the order of the allele but also the position of the allele is taken into consideration.
For both problems, the global optima string is the $f(1234)$ one and has a value of 4.0, whilst the second highest fitness string is $f(3421)$ which has a value of 3.2 in the relative ordering problem and 3.3 in the absolute one.

Knjazew et al. in their work enhanced the problems by introducing tight and loose coding of the problem. For tight coding, it means a coding scheme where the blocks are tight with a defining length of 3. The defining length of a block is the distance between the first and the last gene of the block, and deflen6 defines this length to 6. Loose coding is a type of coding where the defining length is maximal.

subfunction No.	deflen6	loose
1	1, 3, 5, 7	1, 9, 17, 25
2	2, 4, 6, 8	2, 10, 18, 26
3	9, 11, 13, 15	3, 11, 19, 27
4	10, 12, 11, 16	4, 12, 20, 28
5	17, 19, 21, 23	5, 13, 21, 29
6	18, 20, 22, 24	6, 14, 22, 30
7	25, 27, 29, 31	7, 15, 23, 31
8	26, 28, 30, 32	8, 16, 24, 32

Figure 4: Deflen6 and loose coding, from the work of Knjazew et al. [19]

These problems are an interesting benchmark on which GOMEA will be tested; contrary to the winner determination problem, these deceptive problems have a clear structure and their complexity can be clearly evaluated.

3 Problem Representation

3.1 Combinatorial Auction Test Suite

For the generation of problem instances, it is used the Combinatorial Auction Test Suite (CATS), a universal test suite for combinatorial auctions, from the work of Leyton-Brown et al. [21]. In their work, they presented a suite of distribution families for generating realistic, economically motivated combinatorial bids.
The suite offers the option to generate bids according to all the previously published test distributions in order to facilitate the comparison with previous works. Some of the distributions are Uniform, Normal, Exponential, Random, Linear Random, Quadratic etcetera.

CATS generate the instances in the form of a txt file, and depending on the parameters specified in the CATS command line, the instances follow different distributions, have different numbers of bids and goods. Additional parameters like polynomial models, or feature weighting can also be specified to tweak even more the resulting instances.

Once the txt file is generated, a Python function takes care of transforming the file into an array, respectively with the following values:

$$[goodsNumber, bidsNumber, bidsValue, bids]$$

which corresponds to:

- *goodsNumber* : an integer corresponding to the number of goods of the auction
- *bidsNumber* : an integer corresponding to the number of bids of the auction
- *bidsValue* : a list of real numbers, representing the value of each bid
- *bids* : a list containing the bids. Each bid is in the form of a list of integers where each integer corresponds to the index of the good associated with the bid.

3.2 Representation in BRKGA

The population of the BRKGA is represented as a list of individuals. Each individual is stored in the form of an array where each position is a list of two elements.
The first element is an integer, and the second element is a real value between zero and one, the random key. The first element corresponds to the index of the random key and it's sorted in increasing order at the moment the population is created.

This is an example of an individual (Ind.) taken from a problem instance with 4 bids.

```
Ind. = [[0, 0.407], [1, 0.238], [2, 0.250], [3, 0.723]]
```

The number of random keys of one individual is equal to the total number of bids of the auction which is one of the values derived from the txt file and corresponds to the length of the bids list.
The bids list is a list of lists, where, at each position, a list is present, indicating the list of goods required by that specific bid.
For each bid, the list consists of single or multiple numbers indicating an index, which is a representation, of the goods intended to be purchased. The goods index range from 0, to goodsNumber - 1.
An example of a problem instance with 4 bids and 5 goods is shown below. As shown in the example, the bids index ranges from 0 to 3 and the goods index ranges from 0 to 4. For example, the fourth and last bid of the individual, at index 3 contains respectively the indexes corresponding to three different goods, 2, 4, and 0.

```
Bids     = [[4], [1], [0], [2, 4, 0]]
Indexes :    0    1    2      3
```

The last piece of the representation is the bidsValue. It is also a list and consists of real numbers, each one of them indicating the actual value of each bid.

```
BidsValue = [618.493, 817.067, 985.098, 1095.44]
Indexes   :     0        1        2        3
```

Each random key is connected to its value and the list of goods it contains through its index.

Having the index stored with the random key (see the Individual example above), allows the algorithm to not having to store additional information in case of shuffling of the population.
The random key in the first position, which has index 0, corresponds to the bid at index 0 inside the bids list. Moreover, the value of the bid stored at index 0 inside the bidsValue list also corresponds to the random key at index 0.
For example, the random key in the fourth position, with index 3 corresponds, to the bid containing 3 goods, [2, 4, 0] and has a value of 1095.44.

3.3 Representation in GOMEA

The representation in GOMEA follows the representation for the BRKGA. The only difference is between the two representation is merely a programming choice and does not affect the mechanism nor the way the problem instance and the relations between goods, and bids are represented.

The difference resides in the population representation, where for GOMEA, an individual of the population is simply a list of random keys, whilst for the BRKGA an individual is represented as a list of couples ([index, random key]). The index information, which connects the random key to the bid, is represented by the actual index of the random key inside the list (an individual is represented as a list of random keys).

This is an example from a problem instance with 5 goods and 4 bids.

```
individual = [0.40738, 0.23858, 0.25089, 0.82278]
indexes :        0          1          2          3
```

To clarify, the random key in the first position will have index 0, in the second position index 1, and so on. The indexes shown in the example above are displayed for understanding and are not stored in the data structure.

Since an individual is not carrying any additional information regarding the index of the random key, except the index of the key itself, the positions of the random keys inside the individuals remain the same throughout the computation.
During the decoding phase, where a permutation of the keys is required, an additional piece of information, representing the index of the random key is momentarily appended to the individuals.

4 Algorithms implementation

This section will explain the functioning and the implementation of the two algorithms under examination, the Biased Random-Key Genetic Algorithm and Gene-Pool Optimal Mixing Evolutionary Algorithm.

4.1 Biased Random-Key Genetic Algorithm

4.1.1 Creating the population

The first step taken by the algorithm is to create the population. The population varies in dimension, which is the number of individual elements of which it is composed and, being it a parameter of the algorithm, has to be specified a priori. To create the population, information about the auctions are necessary hence it is during this phase that the method to extract information about the auction is invoked. The total number of bids will indicate how many random keys an individual will contain and based on this information, a number of individuals corresponding to the size of the population will be generated. Each individual is composed of a list of lists, each one of them containing respectively in first position an integer, representing the index of the random key which, as stated before, is important to map each specific random key to the bid which represents, and in the second position a real number between 0 and 1 which is the random key.

4.1.2 Generation

After having created the population, the generation cycle can successfully start, and the population will evolve and mutate at each new generation. The stopping criterion used for this algorithm is based on the fitness evaluations of the population singular individuals. If the fittest element remains unchanged for a predetermined number of generations, the algorithm reached its maximum and the computation is concluded.

4.1.3 Fitness calculation

At this phase of the algorithm it is important to calculate the fitness of the population, not only as information about this current generation but also to identify which individuals can assume the elite role and be passed on to the next generation.
In order to calculate the fitness of the population, it is used as a decoder, the decoder calculates the fitness for each individual singularly and returns a list of pairs composed by each individual and its relative fitness values (See Algorithm 1). The decoder follows the chromosomal approach where the keys of an individual are ordered in decreasing order and ties are broken by keys indices.
Once the keys of the individual are ordered, starting from the one in the first position, for each one of them, following the index relative to the key, a list of

goods is retrieved. These goods are associated with this specific bid, and in compliance with the auction requirements, namely that one good can only appear in one bid, the bid is accepted. The list of already accepted goods is adjourned with the goods relative to the bid in exams, and the fitness corresponding to the bid is added to the total fitness of the individual.
If the bid does not satisfy the requirements, which means that one or more of the goods relative to the bid was already present inside the accepted goods lists, then the bid is not taken into account, the accepted goods list is not adjourned and the fitness of the individual is not increased. If the key associated with the discarded bid has a lower value than 0.5 the bid is encouraged for the next generation, which means that the random key values will be equal to one minus the bid value.
This will increase the chances for that specific bid to be taken into account in the next generations. The algorithm finishes once it checked all the random keys and the final fitness value of the individual is calculated.

Algorithm 1: Decoder

```
1  Let $S$ be an empty list to hold the solution;
2  Let $k_j$ be the key associated with bid $B_j$;
3  Let $L$ be a list of bid indexes ordered in non increasing order of keys $k$;
4  foreach $j \in L$ in the given order do
5      if $B_j$ has no marked goods then
6          $S \leftarrow S \cup \{j\}$;
7          Mark all goods of $B_j$;
8      else if $k_j > .5$ then
9          $k_j \leftarrow 1 - k_j$; // discourage bid $B_j$
10     $L \leftarrow L \setminus \{j\}$ ;
11 end
12 return the fitness $\sum_{j \in S} b_j$
```

4.1.4 Parameterized Uniform Crossover

After having decoded the fitness of the population the individuals are ordered based on their fitness.
Three constant specified a priori will decide the ratio of the population for the new generation. The first constant is e and corresponds to the ratio of elite members that will be passed on to the next generation. Based on this constant the first x number of elements of the population are copied into the next generation population.
The second constant μ specifies the ratio of new individuals randomly created that will be present in the population. This allows the population to maintain diversity and maximize the exploration. The remaining part of the population will consist of offspring generated with parameterized uniform crossover, where one parent is chosen from the elite set and the other from the entire population.

The third constant γ is used as a parameter to decide whether the key is inherited from the elite parent or the other parent. An increase in the value of the constant will increase also the probability that the inherited key belongs to the elite parent.
The parameters used for the algorithm are the ones suggested in their work by de Andrade et al. [4] and are $e = 0.4$ which is is the ratio of elite individuals, $\mu = 0.2$ the ratio or random individuals and $\gamma = 0.6$ the probability of selecting a random key from the elite parent for the crossover.

4.1.5 Termination

The algorithm terminates when no better fitness is found for a predetermined number of generations. The output results in the best fitness, the time spent by the algorithm to find the optimal value, and the total number of generations.

Algorithm 2: BRKGA scheme

```
1  Generate the initial population $P$ ;
2  while a stopping criteria is not reached do
3      Decode each chromosome of $P$ and extract its solution and fitness;
4      Sort the population $P$ in nonincreasing order of fitness. Consider
         the top $p_e$ individuals as the elite group $E$;
5      Copy $E$ to the next generation $Q$, unaltered;
6      Add $P_\mu$ randomly generated new chromosomes (mutant) to $Q$;
7      Generate $p - p_e - p_\mu$ chromosomes (offspring) by parametrized
         crossover selecting a random parent from $E$ and another from
         $P \setminus E$. Add them to $Q$;
8      $P \leftarrow Q$;
9  end
10 return best individual found
```

4.2 Gene-Pool Optimal Mixing Evolutionary Algorithm

4.2.1 Creating the population

As for the BRKGA, the first step is to create the population. An input parameter to the GOMEA specifies the number of individuals of which the population will be composed of. As previously explained in the problem representation section, contrary to what happens in the BRKGA, an individual of the population is merely represented by a list of random keys, and the index is the reference that connects the random key to the bid, resulting in a simpler architecture.

4.2.2 Family of subset: Linkage Tree

At every generation, a new Linkage Tree is created, in order to capture dependencies between variables.
The process of building the linkage tree can be separated into two main processes:

- Dependency matrix

- Hierarchical clustering

4.2.3 Dependency matrix

The dependency matrix, whose calculation is based on multiple parameters, describes the dependency of each variable compared to the other ones.
Following the work of Bosman et al. [11], a symmetric notion of dependency, composed of two different parameters, is used to calculate the values of the matrix; the bigger the value, the stronger the dependency between the variables:

$$\delta(j,i) = \delta(i,j) = \delta_1(i,j)\delta_2(i,j) \tag{5}$$

The first parameter, δ_1, focuses on relative-ordering information, which gives us information about the ordering of two different variables. It computes the entropy of the probability that the random key r_i appears before the random key r_j, in other words, the value of the random key r_i is bigger than the random key r_j.
The maximum value the entropy could reach is 1, which signifies a very weak dependency so much that the reverse order could appear with the same probability. On the other hand, the minimum value that the entropy could reach is 0, which means a strong dependency hence it means that one ordering is more plausible to manifest. The value obtained by the entropy is then negated subtracting it to 1 and obtaining the desired notion of dependency:

$$\delta_1(i,j) = 1 - \{-\left[p_{ij}\log_2(p_{ij}) + (1.0 - p_{ij})\log_2(1.0 - p_{ij})\right]\} \tag{6}$$

where

$$p_{ij} = \frac{1}{n}\sum_{k=0}^{n-1}\begin{cases} 1 \text{ if } r_i^k < r_j^k \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

and r_i^k is the random key i of the individual k of the population.

The relative-ordering information is not the only important information that can be extrapolated from the variables ordering. The proximity between two variables also plays an important role in discerning the dependency between different variables. This type of information is called adjacency information, δ_2, and it is calculated by computing the average squared difference between two random keys. Like as in the entropy case, this value is within the $[0, 1]$ limits, and likewise to a greater value corresponds a weaker dependency, hence to obtain the desired notion of dependency this value will also be negated by subtracting it to 1 in order to revert its meaning.

$$\delta_2(i,j) = 1 - \frac{1}{n}\sum_{k=0}^{n-1}\left(r_i^k - r_j^k\right)^2 \tag{8}$$

In the discussion section, it will be explained the choice behind the combination of the two deltas.

4.2.4 Hierarchical clustering

After having computed the dependency matrix, enough information is collected to build the linkage tree.
The algorithm performs bottom-up hierarchical clustering starting from the leaves of the tree and ending with the root node. It starts with the univariate structure where each variable is assigned to its own cluster.
After that, using the dependency matrix, for each cluster it is calculated the closest cluster, generating a list of dependencies. Each position of the list (index) corresponds to a cluster and the value at the specific position corresponds to the most dependent one according to the dependency matrix. The list is progressively examined and at each position, if none of the two clusters, the one corresponding to the index, and the one corresponding to the closest one, are already inside a node at this level, then the two clusters are grouped together and will be a node for the next branch. When the list is terminated, the generated nodes are grouped together inside the branch structure which is added to the linkage tree in the form of a list of lists.
The branch will be passed on to the algorithm and will be used to calculate the next dependency list, using the branch structure instead of the univariate. The branch structure will be gradually adjourned and the process will continue until there is only one cluster left, containing all the variables, the root node.
The data structure of the linkage tree consists of a list, in which every element corresponds to a level of the linkage tree. A level is itself a list, containing a list of nodes, or clusters. Each one of them contains a list of indexes, corresponding to the variables.

Example of a linkage tree representation for a 5 bids problem instance:

$Root$	$[[0,1,2,3,4]]$
$Branch$	$[[0,1,3]]$
$Branch$	$[[0,1],[4,2]]$
$UnivariateStructure$	$[[0],[1],[2],[3],[4]]$

Algorithm 3: Linkage Tree

```
1  Let $T$ be an empty linkage tree;
2  root ← GetRoot(population);
3  univariate ← GetUnivariateStructure(population);
4  tree.append(univariate);
5  while branch != root do
6      dependencies ← getDependenciesForBranch(depMatrix, branch);
7      nextBranch ← createNextBranch(branch, dependencies);
8      tree.append(nextBranch);
9      branch ← adjournBranch(branch, nextBranch);
10 end
11 return tree
```

4.2.5 Greedy recombination

The greedy recombination takes place inside the generation cycle. Once the linkage tree is calculated, each individual of the population has to be recombined, once for every branch on the linkage tree. For each branch, one individual of the population and one donor are selected, the donor is randomly chosen from the entire population, excluding the individual itself, see Algorithm 5.

4.2.6 Mechanism

A branch has several leaf nodes which will be called clusters, or as in the previously shown example, subsets. The individual of the population is initially evaluated in order to have an initial fitness value. To calculate the fitness of an individual, it is utilized the same decoder as for the BRKGA, see Algorithm 1. After the initial fitness evaluation phase, the branch is entirely evaluated and for each leaf node, the individual will be recombined with the donor. See an example of recombination in section 2.2.2.
As shown in the example, the individual inherits the random key at the positions specified inside the cluster from the donor, and then its fitness is evaluated. If the new individual has a lower fitness score than the previous version, the individual will remain unchanged, and the computation will continue with the next cluster. On the contrary, if the newly created individual has a higher fitness score, it will undergo two additional controls.

- Individual already present
- Solution already present

The first control simply checks if the individual is already present in the population as if it exists already another individual that has the same random keys positioned in the same order, if that is true, the new individual is discarded and the computation continues with the next cluster. The second control aims to

identify if there is already an individual of the population that yields the same solution. This decision will be thoroughly explained in the discussion section. As for the previous case, if there is already an element in the population that yields the same solution, the new individual will be discarded.
If both checks are positively passed, the new individual takes the place of the old one and the fitness is adjourned.
Once all the clusters have been evaluated, the individual, which is the fittest, is returned and takes the position of the initial one that was fed to the greedy recombination method.

Once every branch has been explored and the population adjourned, the generation is concluded.

4.2.7 Termination

Two different parameters are used as stopping criteria for GOMEA, the total number of generations and the changes in the population. The first one simply alt the computations after a certain number of generations is reached. If this first criterion is not in place, then the algorithm stops when no changes in the population happened for a specific number of generations.
A counter keeps track for each generation if any individual of the population changed. Every time there is a change in the population the counter is reset. After a certain number of generations, it is possible that no better solution can be found and the population will remain unchanged.
The algorithm returns the best solution found and the generation at which was the maximum found, the total number of generations, and the time used for the computation.

Algorithm 4: GOMEA scheme

```
1  Generate the initial population $P$ ;
2  while the termination criteria are not met do
3      Generate Linkage Tree ($LT$) from $P$;
4      foreach individual ($I$) in $P$ do
5          foreach Branch in $LT$ do
6              Select a random $Donor(D)$ from $P$;
7              $P(I) = GreedyRecombination(I, D, Branch)$;
8          end
9      end
10 end
11 return Best I
```

Algorithm 5: Greedy Recombination

```
1 NewIndividual = ReplaceBranchValues(Individual, Donor, Branch);
2 if ImprovementOrEqual(NewIndividual, Individual) then
3     Individual = NewIndividual ;
4 return Individual
```

4.3 Deceptive permutation problem

In order to adjust the previously implemented GOMEA algorithm to solve deceptive problems, two methods of the algorithm need to be modified:

- Creating the population
- Decoder

The method to create the population is the same as for the winner determination problem, except that the population size for the latter problem is a constant, 32. As in the previous version of the algorithm the individuals are encoded using random keys.

For what concerns the decoder, depending on the encoding input parameter, deflen6 or loose, the individual is divided respectively into 8 substrings which are mapped through an external method to their correct value in case of relative or absolute ordering. The total fitness of the individual is returned.

5 Results

This section will presents the results of the experimentations which will be discussed in the next section (Section 6).

Both algorithms were tested on different problem instances. The instances vary in terms of dimension, distribution, and complexity. Based on Leyton-Brown works et al. [22] the hardness of a problem instance depends majorly on the number of dominated bids in the auction. The higher the number, the easier it will be for the algorithm to find an optimal solution since dominated bids can be eliminated a priori from the instance resulting in a smaller problem size.
Let $v_i(S)$ be the bid that bidder i makes for bundle S, $v_i(S)$ is a dominated bid if there exist another bid from another bidder j for the same bundle S, where $v_i(S) < v_j(S)$.
Based on this information the distributions picked for testing are L3, L6, L7, respectively the uniform, exponential and binomial distributions. Two different problem instances are generated for each distribution, one medium size and one of bigger size tweaked to be as hard as the test suite can produce. The medium size instance has 100 goods and 300 bids, and the bigger instance has 256 goods and 1000 bids. This last problem instance has been generated using the **default-hard** flag, which allows the test suite to use its hardness model previously created to produce the instance; the number of goods and bids in this last case is also standard since the models provided are trained on that specific problem dimension.
The hardware used for the experiments is the free service provided by Google, Google Colab, which offers:

–GPU: 1 Tesla K80 , compute 3.7, having 2496 CUDA cores, 12GB GDDR5 VRAM

–CPU: 1 single core hyper threaded Xeon Processors 2.3Ghz i.e. (1 core, 2 threads)

–RAM: 12.6 GB Available

–Disk: 33 GB Available

The choice of Google Colab lies in an higher computational power compared to owned resources.
The two algorithms are mainly compared in terms of performance and efficiency which translates into:

- Fitness of the best individual
- Number of fitness evaluations

Other relevant aspects as the number of generations, and time required for the computation are also taken into consideration.

5.1 Medium size problem instances, 100 - 300

Both algorithms are run for the same amount of fitness evaluation, precisely 1 million, but the number of generations differs for each of them. The BRKGA algorithm performs a number of fitness evaluations per generation corresponding to the size of the population, in this case, 10000, so the algorithm continues for 100 generations. The GOMEA algorithm instead performs a number of fitness evaluations per generation depending on the linkage tree, the number of bids, and the population size combined:

*(n.bids - 1) * 2 * pop. size*

Which explains in the linkage tree having *l* leaf nodes, and *l-1* internal nodes; the root of the linkage tree is not evaluated, hence in total *l + l - 2*, which is equal to *(l - 1) * 2*.
Each element of the population is examined throughout the whole linkage tree hence the number has to be multiplied by the size of the population. For this specific problem instance GOMEA performs $(300 - 1) * 2 * 50 = 29900$ fitness evaluations per generation, with a total of 34 generations per run.
For both algorithms, to calculate the total number of fitness evaluations for the whole run it is necessary to add the initial individuals' evaluation that happens before the first generation starts, and this number is equal to the population size.
The following graphs show the behavior of the two algorithms on medium-size problem instances based on three different distributions. The title of the graph (ex: L3-100-300) is an abbreviation that indicates *distributions* - *n.of goods* - *n. of bids* of the problem instance. Results are averaged over 25 runs.

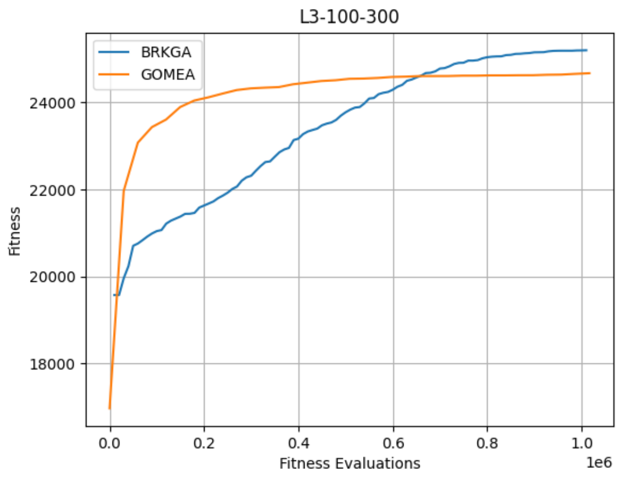


Figure 5: Problem instance based on the uniform distribution

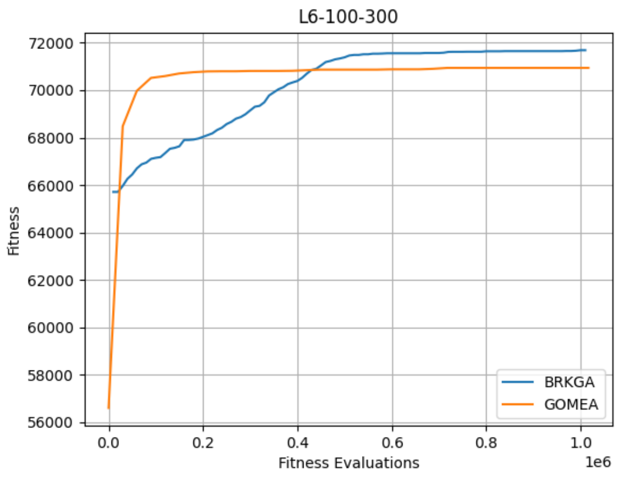


Figure 6: Problem instance based on the exponential distribution

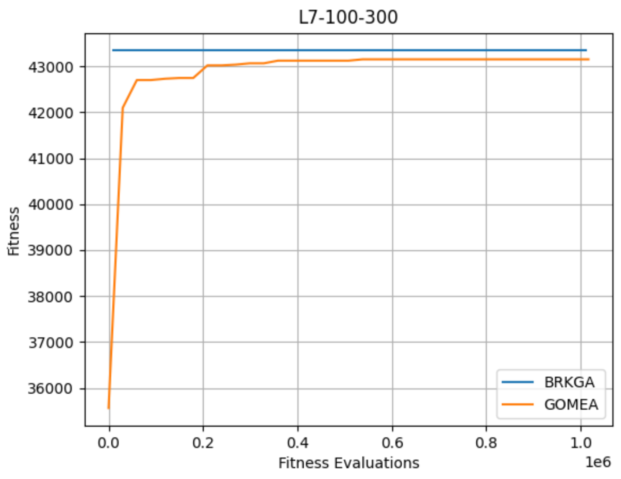


Figure 7: Problem instance based on the binomial distribution

5.2 Big size problem instances, 256 - 1000

The same experiment has been done for the largest set of problem instances. The algorithms are analyzed for the same number of fitness evaluations, 1 million, and results are averaged over 25 runs.
The BRKGA still has a population of 10000 elements and computes for 100

generations, whilst the GOMEA algorithm has a decreased size population of 30 individuals and computes for 17 generations.

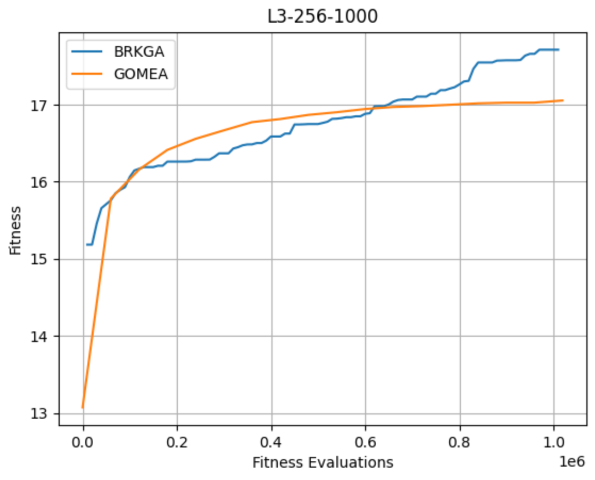


Figure 8: Problem instance based on the uniform distribution

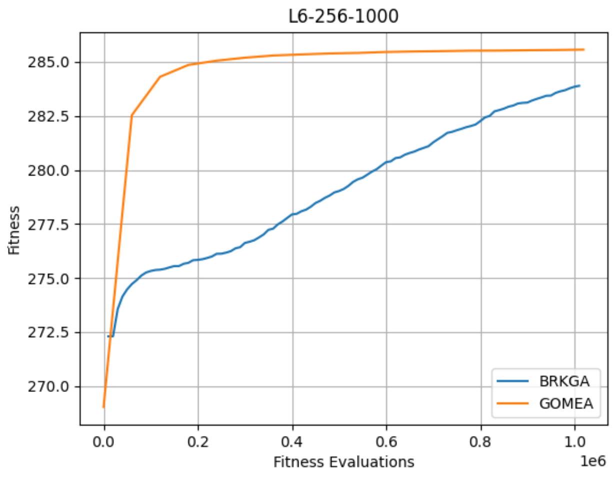


Figure 9: Problem instance based on the exponential distribution

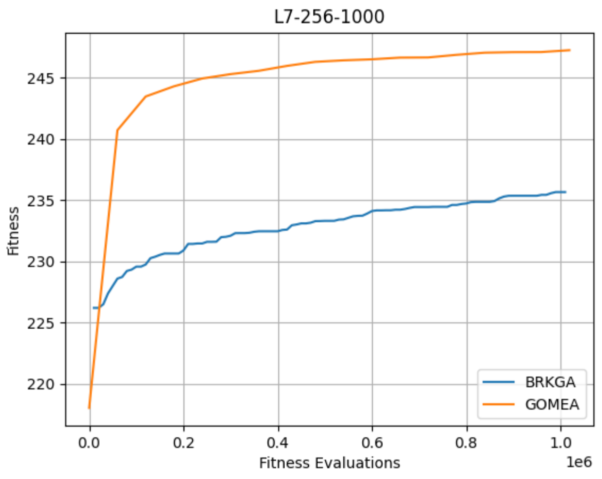


Figure 10: Problem instance based on the binomial distribution

5.3 Univariate model

In order to assess the performance of the linkage tree model, a comparison between the GOMEA algorithm that uses the linkage tree as FOS model and GOMEA that uses the univariate as FOS model has been computed.

The problem instance selected for the comparison is part of the hard problem instance and was generated using the binomial distribution, L7-265-1000. The number of fitness evaluations per generation for the univariate FOS model is almost half the number required for the LT model. This number is calculated using the formula $n.bids * pop.size$, where $n.bids$ corresponds to the number of variables of the problem instance or the number of leaf nodes of the linkage tree.

In order to have comparable results, the two algorithms are set to perform the same number of fitness evaluations per generation, therefore the univariate model has double the population size of the LT version of the algorithm, respectively 60 individuals and 30 individuals.
Both versions of the algorithm use the same stopping criteria, 1 million fitness evaluations threshold and results are averaged over 25 runs.

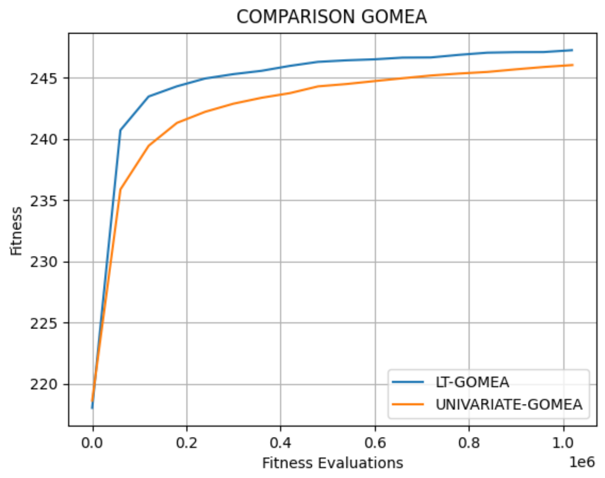


Figure 11: Problem instance based on the binomial distribution, L7-256-1000

5.4 Deceptive permutation problem

In order to have comparable results, GOMEA was tested for the same amount of fitness evaluation as in the previous study, a total of 2 million function evaluations [19].
Instead of comparing the fitness of the best individual of the population, during this comparison, it is tested the total amount of correct subfunctions that the best individual of the population obtained.
This is due to the nature of the deceptive function that makes the algorithm converge to a local optima which still yields considerable fitness value but it is far from the global optima. A good fitness score is not an indicator of the correct functioning of the algorithm.

GOMEA has been tested with different population sizes, 100, 500, and 1000. The best results, averaged over the 4 different problem instances, are obtained with a population of size 500, which translates into 31000 fitness evaluations per generation; a total of 65 generations are needed to reach the 2 million fitness evaluations threshold. Results are averaged over 25 runs.

Below, on the left are the results from the study of Knjazew et al. [19], on the right the results of the GOMEA algorithm on the same deceptive problem.

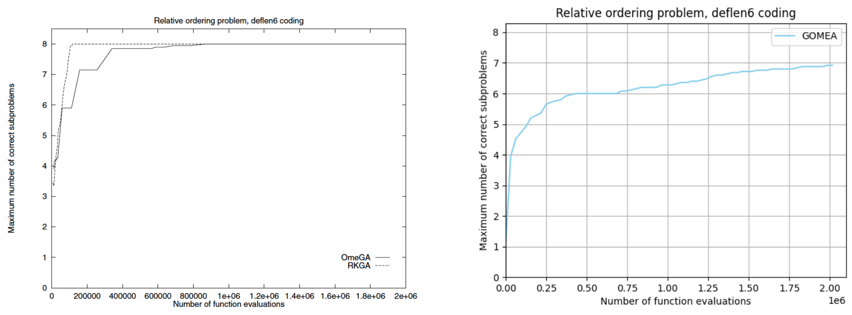


Figure 12: Relative ordering problem, deflen6 coding

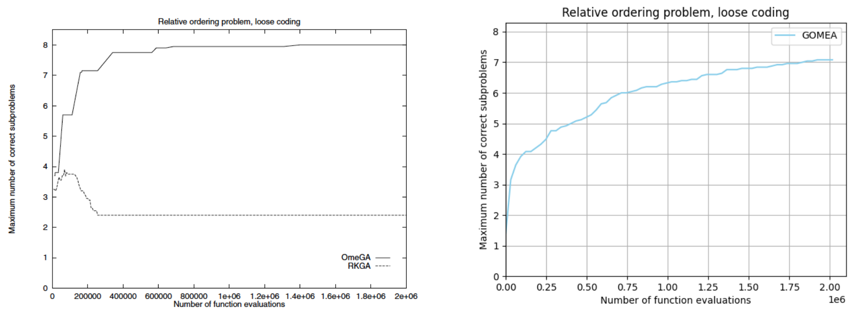


Figure 13: Relative ordering problem, loose coding

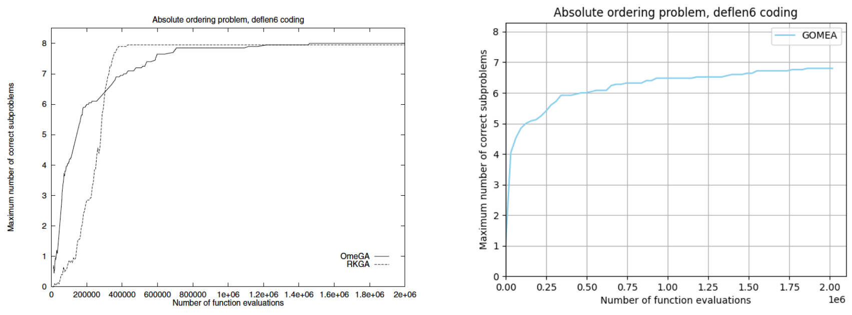


Figure 14: Absolute ordering problem, deflen6 coding

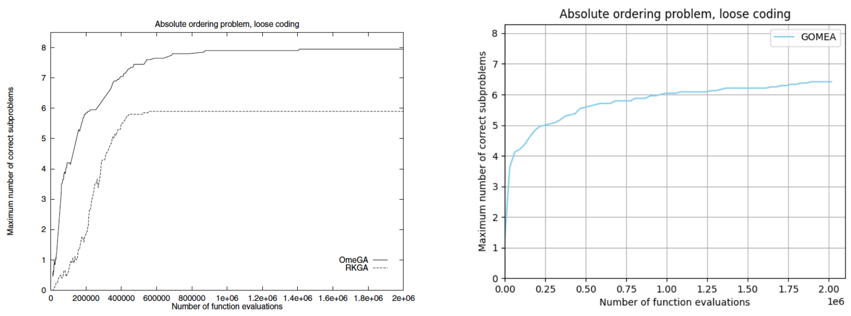


Figure 15: Absolute ordering problem, loose coding

6 Discussion

6.1 Comparison between the two algorithms

As it is shown in the graphs, for the small problem instances BRKGA performs better than GOMEA, in all three different distributions. This cannot be said for the harder problem instances, where just on the uniform distribution instance BRKGA performs better while on the two other distributions GOMEA's score is clearly superior.
When faced with a simpler problem instance the BRKGA scores better results, particularly in Figure 7, where the initial random population of 10000 individuals is enough to already obtain the global optimum value of the problem; the BRKGA line is parallel to the x-axis. The reader will recall that the population size of the BRKGA was chosen in order to compare both algorithms over 1 million fitness evaluations. A smaller population size for the BRKGA would have resulted in hundreds if not thousands of generations reaching the 1 million fitness evaluations threshold.
With harder problem instances instead, the BRKGA fails in most cases to reach the optimum value. Figure 9 and Figure 10 shows that GOMEA outperforms the BRKGA. It is not possible to guarantee the difficulty of the problem if not by testing it with the algorithms that are being tested on these problem instances themselves, but the binomial distributions problem instances are a clear explanation of what was said before. The small version of the binomial distribution, see Figure 7, is extremely easy to solve, in all the 25 tries BRKGA was able to find the global optimum just by generating random solutions. The same distribution was applied with the **default-hard** flag, generating, according to the CATS documentation [3], a much harder distribution in terms of size and non dominated bids. On this harder problem instance, BRKGA fails to achieve satisfactory results compared to the GOMEA algorithm.

To further understand these results we need to take into consideration two important factors, the number of fitness evaluations and the population size.
Due to time constraints, the number of fitness evaluations is set to 1 million, since the results need to be averaged over 25 runs, resulting in hours of computation. At first glance, it seems that the GOMEA algorithm reaches the peak considerably faster compared to the BRKGA, but this is just the effect of having a smaller population.
The graph below shows the behavior of the BRKGA on the medium size problem instance L3-100-300 with three different population sizes, 50, 1000, 10000. The stopping criterion is different for each population size algorithm, since for a population size of 50 individuals, in order to reach 1 million fitness evaluations, it would require 1000000/5 = 200000 fitness evaluations. The stopping criterion for the population size 50 algorithm is 200 generations without an increase of fitness of the best individual in the population. For the 1000 population size algorithm the stopping criteria is 500 generations, and the usual 100 generations (1 million fitness evaluations) for the 10000 population size one.

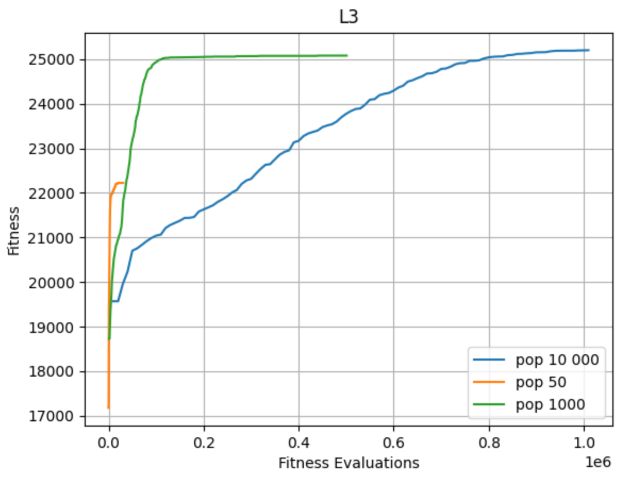


Figure 16: Comparison of different population size for the BRKGA

As it is shown, the smaller the population size, the faster the algorithm reaches its peak in terms of fitness evaluations, hence having the GOMEA a population many times smaller than the BRKGA it is explained why it reaches the peak earlier than the BRKGA.
It is interesting to notice that if on the x-axis is used the number of generations instead of the Fitness evaluations, like in the graph below, the fitness lines follow the same trend for different population sizes while reaching different fitness level; they all seem to reach the peak around the 100 generations mark.

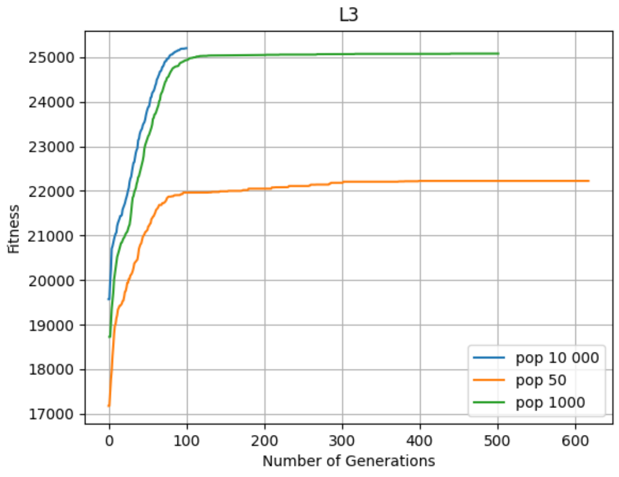


Figure 17: Comparison of different population size for the BRKGA

An increase of 20 times the population size, from 50 to 1000 introduces an important increase in the fitness of the population, but with an additional increase of 10 times the population size, there is little to no increase in the fitness value. The same cannot be said looking at Figure 16, where the same increase in population size from 1000 to 10000 individuals reduces drastically the convergence speed of the algorithm.

6.2 Effectiveness of the linkage tree

Although GOMEA is able to achieve interesting results from a fitness point of view, some experiments have been conducted to understand the capability of the linkage tree to obtain information about the problem and build an actual model that would increment the population's fitness.
Two different variants of the algorithm, that will be called Random, and Random 2, have been created and applied to a problem instance and compared with the normal version of the GOMEA algorithm.
The difference between the standard GOMEA and the Random version is that the linkage tree is built upon a newly created random generation, instead of using the population resulting from the previous generation to calculate the dependency matrix. That allows the algorithm to create a random linkage tree that isn't able to capture dependencies between the problem variable of the individuals of the population since the real population is in fact a different permutation of random keys and values.
To ensure complete randomness, the Random2 variant has been introduced; this version of the algorithm substitutes the dependency matrix, upon which the linkage tree is calculated with a randomly filled matrix with real numbers

between the $[0 - 1]$ interval.

Below is shown the comparison between the 3 versions of the GOMEA algorithm, on the L7-256-1000 hard problem instance. Results are averaged over 25 runs and the stopping criterion is 1 million fitness evaluations.

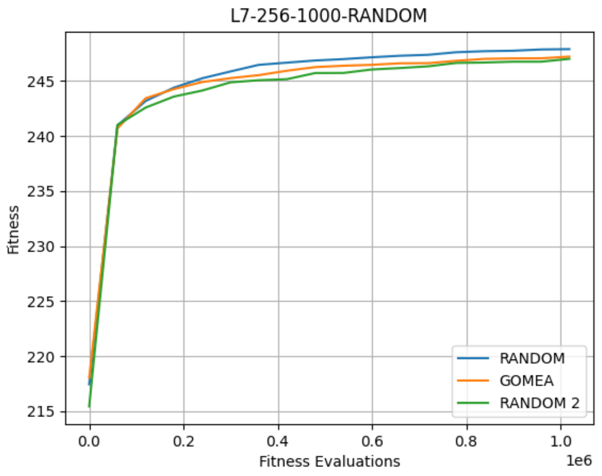


Figure 18: Problem instance based on the uniform distribution

As it is shown in the graph, there is little to no difference between the results of the three versions of the algorithm.

6.3 Delta1 & Delta2

The implementation of the GOMEA algorithms follows the implementation of Bosman et al. [11], for the permutation flow shop scheduling problem. As previously explained the algorithm combined with the BRK encoding demonstrated high capabilities in permutation optimization problems, in both terms of model construction and results obtained.
Since the winner determination problems in multi-combinatorial auction is also a permutation optimization problem, and the GOMEA algorithm with the BRK encoding is being utilized to solve it, it comes naturally to implement it as the previous authors did, utilizing the two deltas, one measuring the relative-ordering information and one the adjacency information.
Different approaches although have been researched, without any positive feedback. Once the bids which constitute the individuals of the populations are encoded using the BRK encoding, they are stripped away of every numeric value, except the random keys, which are sorted in non-increasing order at the moment of generating a feasible solution from the individual.
There is no information that they carry apart from the key itself which does not

contain any value regarding the bid, except its position which is what is important in a permutation problem. Delta 1 which calculates the relative-ordering information is extremely important in building the structure of the linkage tree since it measures the probability of one bid appearing before another, using the value of the random key. This is intuitively an important correlation since depending on the random key value, during the decoding phase, the bids will be sorted, hence having a higher value compares to the other random keys one bid will have a favorable position in the list. One bid can exclude another hence being in the top position of the list increase favorably the possibility of the bid to be accepted.
Delta 2 measures the adjacency information or the distance between two random keys which carries information about the relations between two bids; a bigger distance indicates weaker dependency and it increases the probability of the latter bid being discarded from the solution hence a useful piece of information to build the model. The two deltas are then combined to form a single piece of information.

6.4 Linkage Tree model vs Univariate model

The objective of this comparison is to understand if a simpler FOS model like the univariate structure is able to perform equally in term of fitness, compared to the more structured linkage tree FOS model.
As previously explained, the univariate model is the base structure upon which the linkage tree is built, therefore its implementation was already present in the linkage tree version of the algorithm.
The reason behind this comparison is due to the poor performance of the GOMEA algorithm in learning the problem structure. This is clearly visible in Figure 12, where the standard version of the algorithm shows comparable results with the other two random versions. As shown in the graph, the linkage tree model outperforms the univariate model, although maintaining the same fitness curve.
Performing a T-test shows that there is a statistically significant difference between the results of the two different FOS models, T value 2.89000 and P value 0.00532.

Since the size of the population utilized for the Univariate version of the algorithm is twice as big as the population of the LT version, both algorithms visited the same number of solutions per generation. Therefore these results demonstrate the importance of having building blocks of more than one bit, which allow the LT model to perform more efficient recombination resulting in the linkage tree model outperforming the univariate one.

6.5 Deceptive permutation problem

Results show that the GOMEA algorithm is outperformed by the OMEGA algorithm in all 4 problem instances. For what concerns the RKGA instead,

GOMEA is able to score better just when the problem instance uses loose encoding. In the absolute loose encoding problem instance, the RKGA reaches a plateau at almost 6 correct subproblems whilst GOMEA scores an average of 6.41 correct subproblems. In the relative loose encoding problem instance instead, the RKGA stabilizes approximately around 2.5 corrects subproblems, whilst GOMEA reaches an average of 7.08 correct subproblems.
In all the other problem instances both the RKGA and the OMEGA algorithm reach 8 correct subproblems. It is noticeable the difference in performance for the RKGA between the loose and deflen6 encoding, whilst GOMEA doesn't show any statistically significant difference between the problem instances with a different encoding method. Performing a T-test between the two loose encoded problem instances and the two encoded using Deflen6, yields T value = 0.8221 and P value = 0.41302.
Performing the same statistical analysis but using the ordering method, relative and absolute as a distinction between the two groups yields different results. The T value is 3.18261 and the P value is 0.00196, which means that there is a statistically significant difference between the two groups. The average of the results of the relative ordering problems is 7.0 while the average of the results for the absolute ordering problems is 6.86; the GOMEA algorithm is influenced by the ordering function of the problem instance.
The data from the OMGEA and RKGA test are not available hence it is not possible to perform a statistical analysis to compare them with the GOMEA algorithm.

The two graphs below show a comparison between three different population sizes for the GOMEA algorithm, 100, 500, and 1000 individuals. Two different metrics are used for the analysis on the x-axis, fitness evaluations and the number of generations. Results are averaged over 25 runs.

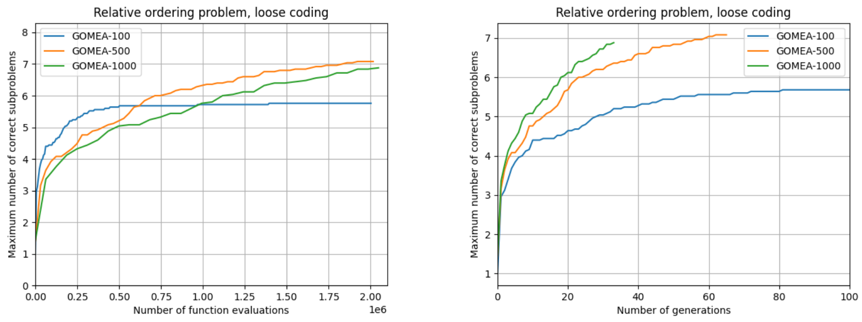


Figure 19: Absolute ordering problem, loose coding

It is interesting to notice that although the 500 individual population size version, performs better in terms of fitness evaluations, according to the graph on the left, the algorithm reaches its peak around 2 million fitness evaluations

whilst the 1000 individual version is yet to achieve its peak and it is still steeply increasing in terms of fitness.
Allowing the algorithms to run for a higher number of generations, removing the 2 million fitness evaluations threshold, would result beneficial for GOMEA algorithm with 1000 individual population size.

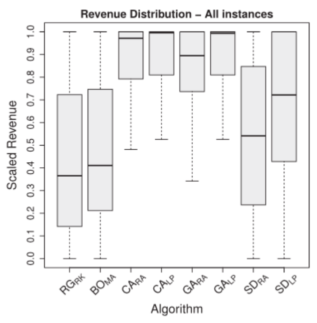
Although there is a difference in the average correct number of subproblems between the 500 and 1000 population size version of the GOMEA algorithm, there is no statistical difference between the groups. Specifically, the two groups score an average of 7.08 and 6.88 correct subproblems respectively. Performing a Ttest between the two different population size versions of the algorithm shows a T value of 0.78974 and a P value of 0.43355, the null hypothesis is rejected, therefore there is no statistically significant difference.

6.6 Implementation choices

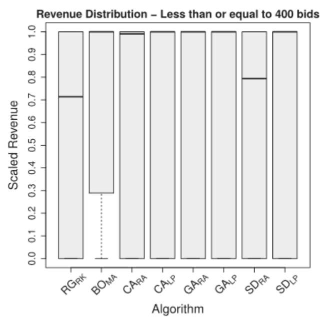
6.6.1 BRKGA Chromosomal Approach

The implementation of the BRKGA comes from the work of De Andrade et al. [4] where the algorithm was tested using three different approaches; the chromosomal approach (CA_{RA}), the greedy approach (GA_{RA}), and the surrogate duality approach (SD_{RA}). Each one of these approaches was then implemented and tested with (CA_{LA}, GA_{LA}, SD_{LA}) and without linear programming relaxation to initialize the population to speed up the convergence of the algorithm. The interest of this study lies in the approaches without LP-relaxations and after careful analysis of the results, as shown below the chromosomal approach, which is the most basic approach, performs better than the other variants. This is due to the fact that the greedy and surrogate duality approaches make the algorithm converge prematurely to poor local optima, resulting in lower fitness.

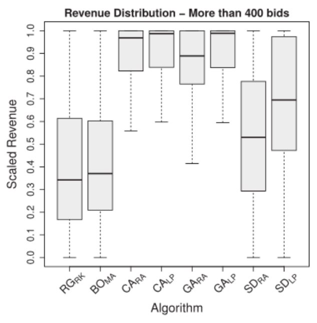
The 4 graphs below are a comparison between the aforementioned six different variants of the BRKGA, Boughaci et al. Memetic Algorithm (BO_{MA}) [23], and Raidl and GottliebWeight-Biased Genetic Algorithm (RG_{RK}) [24]. As explained before, the CA_{RA} outperforms the other variants without LP-relaxation.



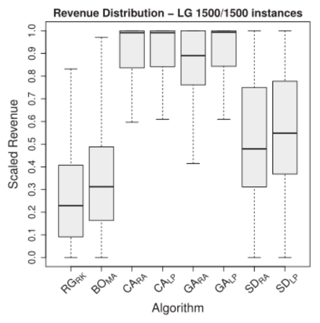
(a) All instances



(b) Small instances



(c) Large instances



(d) LG 1,500/1,500instances

Figure 20: Dispersion of revenue for each heuristic using 100 generations at most, from the work of de Andrade et. al.

6.6.2 GOMEA

In order to ensure and promote diversity within the individual of the populations, when an offspring or a new individual result in an individual that is already part of the population, that individual is then discarded. If it was the case of a newly generated individual of the population, a new individual will be created, if it was the results of recombination, then the parent returns to occupy his position. To ensure that the algorithm makes sure that the random key of an individual does not match the ones belonging to another individual already present in the population. This mechanism is not enough to ensure the diversity of the population and avoid premature convergence.

In the winner determination problem, there is the constraint that a bid can be accepted in the solution only if there is no clash between the goods to which it is associated and the goods associated with the bids already in the solution. One good can only be present in only one bid. This constraint results in the fact that just a portion of the individual becomes part of the actual solution, hence individuals having a different random key can still produce the same solutions and in this way reducing the diversity of the population.

To avoid that an additional control, foreign to the original implementation of the algorithm has been implemented. Not only the bids of the offspring, or new individual, are checked, but also the solution associated with the individual is compared to the solutions present in the current population and if the same solution is found, the individual is discarded. This way redundancy is eliminated which results in less probability of premature convergence of the algorithm.

6.7 Final remarks

There is a substantial difference in how the BRKGA and the GOMEA algorithms work on solving the winner determination problem. The first utilizes a big population size and parametrized uniform crossover while the latter uses a very small population size with greedy recombination in accordance with a FOS model.
The BRKGA reaches a higher fitness value compared to GOMEA on simpler problem instances while GOMEA performs better on harder problem instances.

Comparing the two algorithms using the same population size wouldn't be fair, due to the high number of fitness evaluations per generation performed by the GOMEA algorithm. As an example, taking the first problem instance of the smaller instances group, L3-100-300, the GOMEA algorithm with a 50 individual population size performs 29900 fitness evaluations per generation, while the BRKGA with a population size of 50 individuals still performs 50 fitness evaluations per generation.
At each fitness evaluation a new solution is explored hence GOMEA would visit 598 times the number of solutions visited by the BRKGA per generation. To make this even, the BRKGA algorithm should run with a population of 29900 individuals which is unrealistic and let us visualize the difference between the two algorithms' mechanism. The BRKGA is a smaller and faster algorithm, able to reach sub-optimal results with a 1000 individual population size, as shown in figures 16 and 17. GOMEA instead is a more complex algorithm that requires considerable computational power and still maintaining a small population size, 30 individuals, is able to excel in harder problem instances.

The fact that GOMEA is not able to achieve the same results as the BRKGA on a small problem instance is an indicator of the linkage tree not being able to capture the problem structure. In addition, the random version of the linkage tree proves the fact that there is no structure being learned from the model.
The comparison between the Univariate and the LT versions of GOMEA although proved the latter FOS model capable to achieve better results, and therefore the usefulness of having bigger building blocks on which performing the recombination.
The building of the linkage tree has its foundation on the dependency matrix which is calculated using the two deltas, as explained before. The procedure behind these calculations has been thoughtfully examined and has been proven successful in the previous work with different permutation problems [11].

All these indications lead to the conclusion that these generated problem instances lack structure. That explains why it is impossible for the linkage tree model to work efficiently and moreover, it explains the lack of differences in the results between the standard version and the two random versions of the linkage tree.

7 Conclusion

In this study, it has been proved the effectiveness of the Random Key encoding for combinatorial optimization problems. The Biased Random Key Genetic Algorithm proves once more successful in solving the winner determination problem in multi-combinatorial auctions. The GOMEA algorithm demonstrated high capabilities in solving harder problem instances for the winner determination problem, compared to the BRKGA. CATS has been demonstrated to be an optimal and reliable tool in generating problem instances. The comparison between the two FOS models, the Univariate and the Linkage Tree model, demonstrated the superiority of the latter in this specific domain. Although GOMEA was able to find optimal solutions, the linkage tree did not perform as expected, as in being able to fully represent the problem's structure. This study shows that the problem instances generated using CATS don't have a clear exploitable structure that the linkage tree can utilize. At the same time, the linkage tree performs better than the univariate FOS model, which proves the recombination of groups of bits beneficial even though no structure was learned.

References

[1] Wikipedia contributors. Auction — Wikipedia, the free encyclopedia, 2020. [Online; accessed 06-November-2020].

[2] Michael H Rothkopf, Aleksandar Pekeč, and Ronald M Harstad. Computationally manageable combinational auctions. *Management science*, 44(8):1131–1147, 1998.

[3] Daniel Lehmann, Rudolf Müller, and Tuomas Sandholm. The winner determination problem. *Combinatorial auctions*, pages 297–318, 2006.

[4] Carlos Eduardo de Andrade, Rodrigo Franco Toso, Mauricio GC Resende, and Flávio Keidi Miyazawa. Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary computation*, 23(2):279–307, 2015.

[5] Robert C Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 57–66. Springer, 2001.

[6] Jella Pfeiffer and Franz Rothlauf. Analysis of greedy heuristics and weight-coded eas for multidimensional knapsack problems and multi-unit combinatorial auctions. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1529–1529, 2007.

[7] Terence Kelly. Generalized knapsack solvers for multi-unit combinatorial auctions: Analysis and application to computational resource allocation. In *International Workshop on Agent-Mediated Electronic Commerce*, pages 73–86. Springer, 2004.

[8] Dalila Boughaci. Metaheuristic approaches for the winner determination problem in combinatorial auction. In *Artificial intelligence, evolutionary computing and metaheuristics*, pages 775–791. Springer, 2013.

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

[10] José Fernando Gonçalves and Mauricio GC Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

[11] 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, 2016.

[12] GH Aalvanger, Ngoc Hoang Luong, Peter AN Bosman, and Dirk 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.

[13] Jiyin Liu and Colin R Reeves. Constructive and composite heuristic solutions to the p// ci scheduling problem. *European Journal of Operational Research*, 132(2):439–452, 2001.

[14] Wagner Emanoel Costa, Marco César Goldbarg, and Elizabeth G Goldbarg. New vns heuristic for total flowtime flowshop scheduling problem. *Expert Systems with Applications*, 39(9):8149–8161, 2012.

[15] Peter AN Bosman and Dirk Thierens. The roles of local search, model building and optimal mixing in evolutionary algorithms from a bbo perspective. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 663–670, 2011.

[16] Peter AN Bosman and Dirk Thierens. More concise and robust linkage learning by filtering and combining linkage hierarchies. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 359–366, 2013.

[17] Michal Witold Przewozniczek, Piotr Dziurzanski, Shuai Zhao, and Leandro Soares Indrusiak. Multi-objective parameter-less population pyramid for solving industrial process planning problems. *Swarm and Evolutionary Computation*, page 100773, 2020.

[18] 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, 2011.

[19] Dimitri Knjazew and David E Goldberg. Omega-ordering messy ga: Solving permutation problems with the fast messy genetic algorithm and random keys. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 181–188, 2000.

[20] Hillol Kargupta, Kalyanmoy Deb, and David E Goldberg. Ordering genetic algorithms and deception. In *PPSN*, pages 49–58. Citeseer, 1992.

[21] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76, 2000.

[22] Kevin Leyton-Brown and Yoav Shoham. A test suite for combinatorial auctions. *Combinatorial auctions*, 18:451–478, 2006.

[23] Dalila Boughaci, Belaïd Benhamou, and Habiba Drias. A memetic algorithm for the optimal winner determination problem. *Soft Computing*, 13(8-9):905, 2009.

[24] Günther R Raidl and Jens Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary computation*, 13(4):441–475, 2005.