Genetic algorithms on winner determination problems

Javier Guillén Martí
UPV
Valencia, España
jguimar@inf.upv.es

Hector Mártinez Cabanes
UPV
Valencia, España
hmarcab@inf.upv.es

Sergio Viana Llovell
UPV
Valencia, España
sviallo@inf.upv.es

Óscar Lucas Blázquez
UPV
Valencia, España
oslubla@inf.upv.es

***Abstract*— In this paper we propose a solution to the winner determination problem in a combinatorial auction platform for business-to-business (B2B) e-commerce using a genetic algorithm. Our solution, a biased random-key genetic algorithm, is based on the random keys technique and includes the option of weighting the random numbers of the chromosome by the ratio of each bid. We also implemented different crossover operators, elite population fraction, parent selection strategy and mutation operators to optimize the algorithm. Finally, we performed a thorough hyperparameter tuning using the WandB framework, resulting in promising results when tested on various datasets. The best combination has been selected based on the score and the runtime and it has achieve an average normalized score of 0.5907 with a standard deviation of 0.99 and an average runtime of 76.81 seconds.**

Keywords— Genetic Algorithm, Winner Determination Problem, Optimization, Python, Metaheuristics

I. INTRODUCTION

In this work, we tackle the problem of determining the winners in a combinatorial auction platform for business-to-business (B2B) e-commerce. Each day, producers offer a number of different products that can be purchased by consumers on the platform, and consumers submit offers to buy these products. At the end of the day, the platform must decide which transactions will be carried out in order to maximize its own profit, which corresponds to 5% of the total monetary amount provided by the auctions. To do this, the platform must select a subset of offers that are all compatible with each other, meaning that they do not share any common products. Additionally, offers are indivisible, meaning that only an entire offer can be selected as a winner, not just a part of it. The task is therefore to select a set of compatible offers that maximizes the profits for the platform. This problem, known as the winner determination problem, may seem simple at first but presents several challenges, including the need to consider compatibility and indivisibility among offers, as well as the potentially large number of offers to evaluate. In this work, we propose a solution to the winner determination problem using a genetic algorithm, which is a type of optimization technique inspired by the natural selection process. Specifically, we design a biased random-key genetic algorithm that is able to efficiently and effectively select the winning offers while considering the constraints of compatibility and indivisibility. Our algorithm is tested on a variety of datasets and demonstrates promising results.

II. STATE OF THE ART

The scientific articles presented in this work deal with the application of genetic algorithms in the context of combinatorial auctions, where resources are assigned to users who offer a price for some of these resources. These auctions can be complex and finding an optimal solution can be challenging. Therefore, several authors have proposed different methods for solving this type of problem using genetic algorithms.

In the first article [1], Schwind, Stockheim, and Rothlauf propose three methods for solving the problem of combinatorial auctions, including a greedy algorithm and two metaheuristics. One of the metaheuristics is a genetic algorithm whose main advantage is the use of a chromosome representation based on the "random keys" technique. This technique only works with valid solutions, so it is not necessary to implement correction techniques after applying the crossover or mutation operators. Going deeper into the random keys technique, J. Gonçalves and M. Resende present in [2], a specification of the random keys technique in genetic algorithms called biased random keys (BRKGA), which consists of applying a parent selection strategy in which one parent is selected from the subset of best solutions in the population (elite population) and the other parent from the rest of the population. This can be beneficial because it reduces the computational complexity of the algorithm. Therefore, from [1] and [2] we will implement both the proposed random keys representation, as well as rely on certain points of the structure that the genetic follows at each iteration. These points will be discussed further in the section of the paper where the whole structure of the algorithm we have developed is laid out.

In a similar way of finding high quality solutions quickly, as in [1], V. Avasarala, H. Polavarapu and T. Mullen develop in [3] a Seeded Genetic Algorithm that uses a novel chromosome representation based on random keys that produces only feasible solutions. In this case, a chromosome is represented by a string of numbers where each number corresponds to the rank of the corresponding bid assigned randomly (from 1 to the number of bids). However there is a main difference between [1] and [3], so in the second one, standard crossover, mutation and tournament selection are used and moreover, after producing a chromosome the authors added an operator called regenerator. They expose that this regenerator is necessary since the mutation and crossover

operators lead to having multiple copies of the same bid in a chromosome, thus, the regenerator takes a degenerate chromosome and returns a healthy one in which no bid is repeated.

In [4], by Carlos Eduardo, Rodrigo Franco, Mauricio G., and Flávio Keidi, proposes a version of BRKGAs for solving the problem of combinatorial auctions. This algorithm has two key features: a chromosome encoding that uses a vector with t uniformly drawn random keys (range [0,1]) and an evolutionary process that uses parameterized uniform crossover for exploitation instead of the mutation operator. Crossover is performed between an individual from the elite set and one from the remainder population, while exploration is achieved through the introduction of mutants at each generation. This scheme prevents infeasibility because the resulting chromosomes, both offspring and mutants, are always vectors of random keys over [0,1].

Finally, in [5] Avasarala, Polavarapu, and Mullen discuss the use of BRKGAs to solve the distributed load-balancing scheduling problem (DLSP). This problem involves finding an efficient schedule for tasks to be completed on a set of processors in order to minimize the makespan, or the time required to complete all tasks. The authors propose a "seeded genetic algorithm" that uses a novel chromosome representation based on random keys that produces only feasible solutions. In this case, a chromosome is represented by a string of numbers where each number corresponds to the rank of the corresponding bid assigned randomly (from 1 to the number of bids).

III. GENETIC ALGORTIHM DESIGN

In this section we will expose all the operators and strategies implemented in our genetic algorithm, which will then be evaluated to obtain the optimal version of our proposal.

A. Chromosome representation

We have opted for 2 different representations of the chromosome, both based on the random keys technique explained in [insert bibliographic references]. This technique consists of assigning a random number between 0 and 1 to each bid, i.e. the chromosome is a list of length equal to the number of bids, where each value is a random number between 0 and 1. When obtaining the score of a chromosome, the indexes of the vector are ordered according to the value of the random number they contain (from smallest to largest), and the bids are accepted in that order as long as the items they contain do not overlap with those that have already been sold in the previously accepted bids. Figure 1 represents an example of random keys encoding.

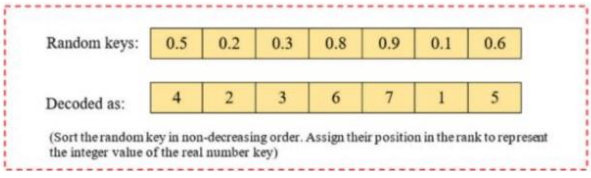

Figure 1. Random keys scheme

In this way, any solution created by crossing or mutation will be valid, and it is not necessary to carry out any type of check or operation to fix the vectors resulting from applying the operators, since when modifying a vector all we are doing is altering the order in which we are going to select the bids.

This is the basic representation based on random keys, apart from this option, we have implemented another type of solution based on this, which consists of weighting the random numbers of the vector by the ratio (number of objects/amount) of each bid. We do this weighting because by ordering the vector from smallest to largest when obtaining the solution, we want the smallest numbers to be those corresponding to the bids that yield the highest profit by selling the fewest objects. In a way, we are subtracting randomization from the process by focusing it on the offers that a priori we are most interested in accepting.

These 2 representations would be 2 different values of a hyperparameter, and in the experimentation phase we will decide which one gives us better solutions to our problem.

B. Parent selection strategy

In this problem, we use a specific strategy for parent selection in a genetic algorithm. At the beginning of each generation, we divide the population into two sets: an elite population, which includes the best solutions, and a non-elite population, which includes the worst. The size of the elite population is always less than half the total population size. Then, to select the parents, there are two possible strategies: choose one parent from the elite population and one from the non-elite population or choose one parent from the elite population and one from the entire population, with the possibility that the two parents are the same individual due to replacement. The size of the elite population, as well as the 2 selection strategies, are hyperparameters that we will optimize through experimentation. This strategy of parent selection applied with the random keys representation is known as 'Biased random key strategy' as is explained in [2].

C. Crossover operators

About the crossover operators implemented, we have developed 4, all of them receive: one elite father, one sub elite father and an elite probability (p_e>0.5), and all of them produces one offspring chromosome.

C.1. Biased crossover

The first one is the biased crossover, which generates a random probability for each gen of the new chromosome. Each gen will be the one from the elite father if the random probability is lower than the elite probability and will be the one from the other father otherwise.

C.2. Complete arithmetic recombination crossover

This crossover operator produces one chromosome where each gen is computed giving a higher chance to the elite father's gen with the elite probability, using the following formula:

$$Gen_{i,child} = p_e \times Gen_{eliteParent} + (1 - p_e) \times Gen_{subeliteParent}$$

Equation 1 Arithmetic Recombination

C.3. Simple arithmetic recombination crossover

The third one is the simple arithmetic recombination crossover where a cut point is chosen randomly, the new

chromosome will inherit the same genes that the elite father has before this cut point. On the other hand, the gens after the cut point, including it, will be calculated using Equation 1.

C.4. Unique arithmetic recombination

Applying this crossover operator, the new chromosome will inherit all the genes from the elite father except one that will be selected randomly and will be calculated following the Equation 1.

D. Mutation operators

In relation with the mutation operators, 2 are the ones that have been implemented. On one hand, we have developed the simple swap operator which consists of choosing two genes randomly and swapping them. On the other hand, we have implemented a variation of the previous operator, in this case we sort the chromosome in an ascendent way, and we exchange one gen from the first half with another from the second half. Applying this variation of the simple swap mutation to our problem, we intend to force a bid that was very unlikely to be selected in the final solution to be much more likely to be selected by assigning it a low random key value.

E. Replacement and diversity maintenance strategies

Because of the way we have implemented the genetic algorithm, both strategies are implicit in the iteration process of the algorithm. Firstly, there is no defined replacement strategy, but in each iteration a percentage of the population pop_e passes to the next generation (pop_e<0.5), another percentage pop_a is composed of new random solutions (pop_a<0.5), thus implementing in the algorithm a diversity maintenance strategy in each iteration. Finally, the rest of the population that will pass to the next iteration comes from applying the crossover and mutation operators explained in the previous sections.

In short, the population that passes to the next generation is formed by a set of the best individuals of the current population, a set of new random individuals and a set of individuals resulting from applying the crossover operator (in addition to the mutation operator in some cases) on the individuals of the current population. With this structure, we ensure that the best individuals from each generation are passed on to the next, we maintain diversity with random solutions, and the offspring resulting from applying the crossover operator has genetic material from both an individual that is passed on to the next generation and one that is not, thus adding more diversity to the population.

F. General scheme followed by the Genetic Algorithm

As it is common in genetic algorithms, ours initializes by creating an initial random population of solutions, following the random keys technique explained in *section A.* Then the algorithm starts and continues while the stopping criterion is not triggered, that is, the number of generations that don't improve the results equals to 100 or the time of execution surpasses 180 seconds.

Firstly, our population is divided into elite and sub-elite groups by sorting the solutions by its fitness, being the elite less than 50% of the total. The elite population continues to the next generation without modifications. Then, a random number (between 1 and half of the size of the initial population) is selected, and new solutions are created and added to the next generation set. The population size of the next generation is not equal to the first one yet, so the remaining solutions will be created by the crossing of the elite and sub-elite solutions of the previous generation, depending on which cross operator is selected. Furthermore, there is a probability of mutation for these crossed individuals, that can occur before being added to the next generation.

When the next generation is complete, we evaluate the best solution obtained obtaining its fitness and representation. If the best solution of the generation improves the current best solution found, the best solution variable is updated and the stopping criterion variable of number of generations without improving the solution restarts to 0, if not, the best solution is not modified and the stopping criterion variable increases in 1. The algorithm continues operating from the new generation until the stopping criterion is met.

IV. METHODOLOGY

Once we have explained all the aspects of the genetic, we must determine which is the best configuration to solve our problem, for this, we must adjust the different hyperparameters that we have generated in the different operators and strategies. Recapitulating all the information from the previous section, the hyperparameters to be adjusted together with their possible values are the following:

- Population size: the size of the initial population takes one of the following values: 50, 150, 300 or 500.
- Solution type: the type of the representation: random or weighted solutions.
- Elite population fraction: portion of the population size that will constitute the size of the elite subset. This hyperparameter can take the values: 0.1, 0.25, 0.3 and 0.4.
- Parent selection strategy: if the second parent is chosen from the non-elite subset or from the entire population.
- Crossover operator: the crossover operator applied between biased, complete arithmetic recombination, simple arithmetic recombination, or unique arithmetic recombination
- Elite probability: probability to inherit information from the elite parent, values available are 0.6, 0.7 and 0.8
- Mutation operators: mutation operator applied between simple swap, and our simple swap modification
- Mutation probability: probability of mutate an offspring chromosome, available values are 0.05, 0.15 and 0.3

To perform the tuning of these hyperparameters we will use the WandB framework, which allows us to launch multiple

runs of the algorithm with different hyperparameter settings and evaluate their results.

V. EXPERIMENTATION

Once the genetic algorithm and the framework are implemented in Python, we start to evaluate the efficiency of our model. In the first moment, we proposed to run a grid search for finding the best parameter combination, however, this grid search had to try 4.608 possible combinations for 8 different files which implied a large computational cost. Therefore, we thought of using a Bayesian search that would maximize the fitness function, which is already implemented in WandB, with a specific number of executions. In this case, we have decided to run 100 executions for each *population size* for each file, except for *population size* equal to 500 that we have run 80 executions for file.

After doing these experimentations we have 3.040 results. In this section we will study if statistical differences exist between the different combination of parameters.

First, and before doing any plot or test, we have normalized the score variable in the following way: to each score obtained for a file, the mean has been subtracted and divided by the standard deviation of the scores obtained for this file. Remember that we have 8 different files with different length and different difficulty.

Once the results have been normalized, we compute the mean score of each hyperparameter combination. Here we have observed that there are 6 combinations that achieve the highest score:

- *Conf1*: {*Population size*: 150, *Solution type:* weighted, *Elite population fraction*: 0.1, *Parent selection strategy*: entire population, *Crossover operator*: biased, *Elite Probability*: 0.8, *Mutation operator*: simple swap modification, *Mutation probability*: 0.15}.

- *Conf2*: {*Population size*: 150, *Solution type:* weighted, *Elite population fraction*: 0.25, *Parent selection strategy*: non-elite subset, *Crossover operator*: biased, *Elite Probability*: 0.8, *Mutation operator*: simple swap, *Mutation probability*: 0.15}.

- *Conf3*: {*Population size*: 300, *Solution type:* weighted, *Elite population fraction*: 0.1, *Parent selection strategy*: non-elite subset, *Crossover operator*: complete arithmetic recombination, *Elite Probability*: 0.7, *Mutation operator*: simple swap modification, *Mutation probability*: 0.05}.

- *Conf4*: {*Population size*: 150, *Solution type:* weighted, *Elite population fraction*: 0.1, *Parent selection strategy*: non-elite subset, *Crossover operator*: biased, *Elite Probability*: 0.7, *Mutation operator*: simple swap, *Mutation probability*: 0.05}.

- *Conf5*: {*Population size*: 150, *Solution type:* weighted, *Elite population fraction*: 0.4, *Parent selection strategy*: non-elite subset, *Crossover operator*: complete arithmetic recombination, *Elite Probability*: 0.8, *Mutation operator*: simple swap modification, *Mutation probability*: 0.15}.

- *Conf6*: {*Population size*: 150, *Solution type:* weighted, *Elite population fraction*: 0.1, *Parent selection strategy*: entire population, *Crossover operator*: complete arithmetic recombination, *Elite Probability*: 0.6, *Mutation operator*: simple swap, *Mutation probability*: 0.05}.

Lastly, for having the same number of observations for each configuration and therefore, not perform wrong interpretations of the results, we have executed 2 runs for each configuration for all the files. To these last results, 16 observations for configuration (96 in total), we have applied the normalization used previous.

VI. RESULTS

With the aim of selecting the best combination of hyperparameters, multiple test and analyses will be applied.

Firstly, we have checked if the variable score for each configuration follows a normal distribution to know which test we should apply for finding significant differences.

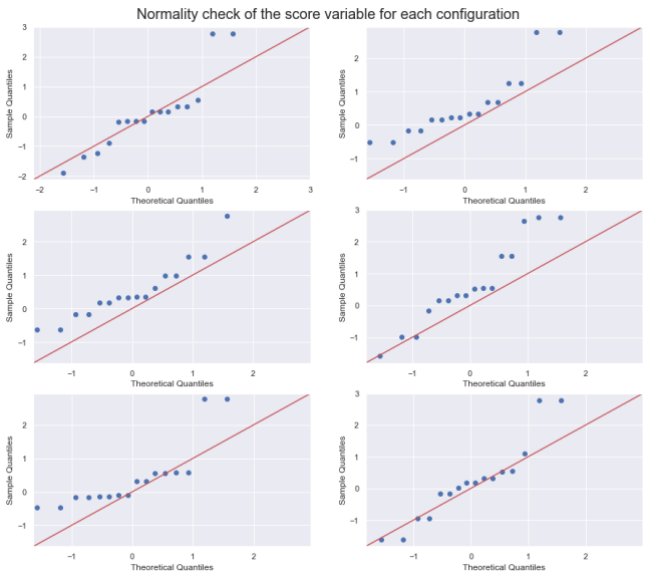

Figure 2. QQplots of the score variable for each configuration.

In this case, we can infer that any configuration follows a normal distribution, therefore we cannot apply ANOVA [6] for finding significant differences instead of using ANOVA we have decided to use the Kruskal-Wallis [7] test.

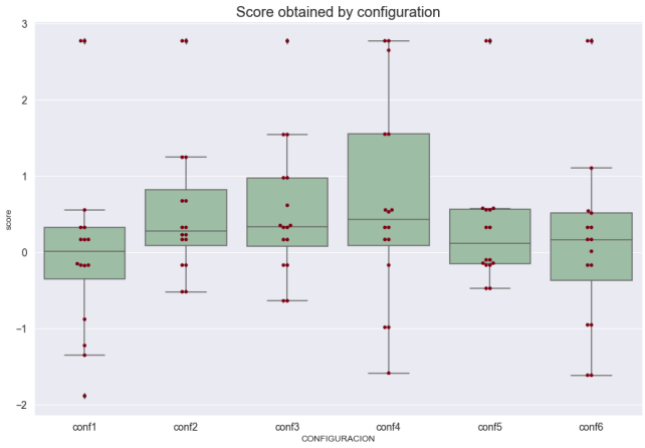

Figure 3. Boxplot of the score variable for each configuration.

The result provided by the Kruskal-Wallis test is that populations are equal since the *p-value* obtained is equal to 0.4561. In Figure 3, we can observe how the configurations have similar behavior but with different variability. Lastly, for deciding the best combination of parameters we have analyzed the run time of each one.

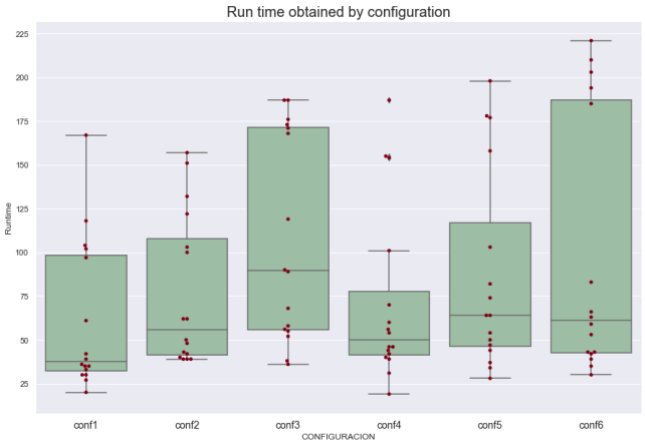

Figure 4. Boxplot of the run time variable for each configuration.

In this last figure we observe how the *Conf4* is the one that provides lower run times, however, looking at Figure 3 this configuration has the highest variability. To have a balance between a high score with low variability and a low run time with low variability, we have selected finally the *Conf2* as the best configuration for solving our problem. The statistics achieved by this configuration are an average normalized score of 0.5907 with a standard deviation of 0,99. In the other hand, regarding to the runtime, *Conf2* achieved an average run time of 76.81 seconds.

VII. Conclusions

In this work, we proposed a solution to the winner determination problem in a combinatorial auction platform for business-to-business (B2B) e-commerce using a genetic algorithm. Our approach, a biased random-key genetic algorithm, effectively selects the winning offers while considering the constraints of compatibility and indivisibility.

We based our algorithm on the random keys technique and included the option of weighting the random numbers of the chromosome by the ratio of each bid. This allowed us to obtain a more effective selection of the winning offers by focusing on the offers that yield the highest profit by selling the fewest objects.

We also implemented different crossover operators, elite population fraction, parent selection strategy, and mutation operators to optimize the algorithm. These different elements of the algorithm allowed us to explore different solution spaces and find the optimal combination of parameters. By performing a thorough hyperparameter tuning using the WandB framework, we were able to evaluate the performance of the algorithm on various datasets, and select the best configuration based on score and runtime. The best configuration selected was Conf2, achieving an average normalized score of 0.5907 with a standard deviation of 0.99 and an average runtime of 76.81 seconds.

It is important to note that our algorithm is a heuristic method, and therefore, it does not guarantee to find the global optimal solution, but it can find good solutions in a relatively short amount of time. Additionally, this algorithm can be sensitive to initial conditions and parameters, and thus, testing on different datasets, as well as fine-tuning the parameters could lead to better results.

In conclusion, the results obtained from our thorough hyperparameter tuning demonstrate the promising performance of our proposed algorithm on various datasets. Future work could include testing the algorithm on larger and more diverse datasets, as well as exploring other optimization techniques for solving the winner determination problem. Additionally, it would be interesting to evaluate the applicability of the algorithm in a real-world scenario and compare it with existing algorithms to see its effectiveness in comparison to those.

References

[1] M. Schwind, T. Stockheim y F. Rothlauf, «Optimization Heuristics for the Combinatorial Auction Problem» de The 2003 Congress on Evolutionary Computation, IEEE, 2003, pp. 1588-1595. doi: 10.1109/CEC.2003.1299862

[2] Gonçalves, J.F., Resende, M.G.C. (2018). Random-Key Genetic Algorithms. In: Martí, R., Pardalos, P., Resende, M. (eds) Handbook of Heuristics. Springer, Cham. https://doi.org/10.1007/978-3-319-07124-4_30

[3] V. Avasarala, H. Polavarapu and T. Mullen, "An Approximate Algorithm for Resource Allocation Using Combinatorial Auctions," 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2006, pp. 571-578, doi: 10.1109/IAT.2006.33.

[4] V. Avasarala, H. Polavarapu and T. Mullen, "An Approximate Algorithm for Resource Allocation Using Combinatorial Auctions," 2006 IEEE/WIC/ACM

International Conference on Intelligent Agent Technology, 2006, pp. 571-578, doi: 10.1109/IAT.2006.

[5] Brandão, J. S., Noronha, T. F., Resende, M. G. C. & Ribeiro, C. C. (2015). A biased random-key genetic algorithm for single-round divisible load scheduling. International Transactions in Operational Research, 22(5), 823-839. https://doi.org/10.1111/itor.12178

[6] Girden, E. R. (1992). ANOVA: Repeated measures. Sage.

[7] (2008). Kruskal-Wallis Test. In: The Concise Encyclopedia of Statistics. Springer, New York, NY. https://doi.org/10.1007/978-0-387-32833-1_216