Developing Tools for the Team Orienteering Problem
A Simple Genetic Algorithm

João Ferreira[1], José A. Oliveira[1], Guilherme A. B. Pereira[1], Luis Dias[1]
Fernando Vieira[2], João Macedo[2], Tiago Carção[2], Tiago Leite[2] and Daniel Murta[2]
[1]*Centre Algoritmi, Universidade do Minho, Braga, Portugal*
[2] *Graduation in Informatics Engineering, Universidade do Minho, Braga, Portugal*

Keywords: Routing Problems, Team Orienteering Problem, Optimization, Metaheuristics, Genetic Algorithm.

Abstract: Presently, the large-scale collection process of selective waste is typically expensive, with low efficiency and moderate effectiveness. Despite the abundance of commercially available software for fleet management, real life managers are only minimally helped by it when dealing with resource and budgetary requirements, scheduling activities, and acquiring resources for their accomplishment within the constraints imposed on them. To overcome these issues, we intend to develop a solution that optimizes the waste collection process by modelling this problem as a vehicle routing problem, in particular as a Team Orienteering Problem (TOP). In the TOP, a vehicle fleet is assigned to visit a set customers, while executing optimized routes that maximize total profit and minimize resources needed. In this work, we propose to solve the TOP using a genetic algorithm, in order to achieve challenging results in comparison to previous work around this subject of study. Our objective is to develop and evaluate a software application that implements a genetic algorithm to solve the TOP. We were able to accomplish the proposed task and achieved interesting results with the computational tests by attaining the best known results in half of the tested instances.

1 INTRODUCTION

In the last few decades, waste separation and recycling have become critically important. They are performed via an easy to accomplish daily routine due to the existence of collection points, which are usually placed near residential areas, with greater concentration in city centres and more spread out in the suburbs and rural areas.

Among waste collection companies, a common problem is finding efficient methods for performing the collection of separated waste, while using a limited fleet of vehicles and obtaining the highest possible profit. This problem may be addressed as a variation of the well-known Vehicle Routing Problem (VRP), described in the literature as the problem of designing the least-costly routes from a depot to a set of customers of known demand. The routes must be designed so that each customer is visited exactly once, without violating capacity constraints and while aiming to minimize the number of vehicles required and the total distance travelled. However, the majority of real-world applications require systems that are more flexible in order to overcome some imposed constraints that may lead to the selection of customers. To deal with these changes, the Team Orienteering Problem (TOP) models can be used. In the TOP, each customer has an associated profit, and the routes have maximum durations or distances. The choice of customers is made by balancing their profits and their contributions for the route duration or distance. The objective is to maximize the total reward collected by all routes while satisfying the time limit.

The TOP is a fairly recent concept, first suggested by Butt and Cavalier (1994) under the name Multiple Tour Maximum Collection Problem. Later, Chao et al. (1996) formally introduced the problem and designed one of the most frequently used sets of benchmark instances. In 2006, Archetti et al. achieved many of the currently best-known solutions for the TOP instances by presenting two versions of Tabu Search, along with two

Ferreira J., Oliveira J., Pereira G., Dias L., Vieira F., Macedo J., Carção T., Leite T. and Murta D..
Developing Tools for the Team Orienteering Problem - A Simple Genetic Algorithm.
DOI: 10.5220/0004273801340140
In *Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems* (ICORES-2013), pages 134-140
ISBN: 978-989-8565-40-2

metaheuristics based on Variable Neighbourhood Search (VNS). Other competitive approaches were carried out by Ke et al. (2008), with two Ant Colony Optimization (ACO) variations; Vansteenwegen et al. (2009), with a VNS-based heuristic; and more recently, Souffriau et al. (2010) designed two variants of Greedy Randomized Adaptive Search Procedure with Path Relinking.

The work presented in this paper is part of experiments that are integrated in the R&D project named Genetic Algorithm for Team Orienteering Problem (GATOP), which was approved by the Portuguese Foundation for Science and Technology (Fundação para a Ciência e Tecnologia – FCT). It involves five combined tasks to accomplish the desired goal which is the development of a more complete and efficient solution for several real-life multi-level Vehicle Routing Problems (VRP), with emphasis on the waste collection management. This should be achieved by the implementation and testing of heuristic and optimization strategies, in close collaboration with demand forecasting, transportation problems, simulation, and multi-criteria decision models.

Within the GATOP project, the main task is to solve the TOP and the development of heuristic solutions based on a genetic algorithm (GA) is suggested. The simplicity of a GA in modelling more complex problems and its easy integration with other optimization methods were the factors considered for its choice. Therefore, we believe it can be applied to solve the TOP, since it was also used for the Orienteering Problem by Tasgetiren (2002).

In this work we propose to solve medium-to-large-scale TOP instances considering a time constraint. We intend to verify whether it is possible to develop a method, based on a GA, that optimizes the TOP by achieving equal or better results as presented in previous studies.

2 PROBLEM FORMULATION

The aim of the present study is to solve the TOP, which means to develop a method that determines P paths which start in the same location and have the same destination, in order to maximize the total profit made in each path, while respecting a time constraint. Then, the generated paths are assigned to a limited vehicle fleet, usually one path to each available vehicle.

In this work we followed the mathematical formulation presented in the work done by Ke et al., (2008). The objective function for the TOP is given in equation 1, where n is the total number of vertices, m is the number of vehicles available, the value y shows if vertex i is visited or not by a vehicle k, and finally, r is the reward associated to a certain vertex i. The objective function consists of finding m feasible routes that maximizes the total reward or profit.

$$\max \sum_{i=2}^{n-1} \sum_{k=1}^{m} r_i \cdot y_{ik} \quad (1)$$

3 DEVELOPING TOOLS

3.1 The Genetic Algorithm

The Genetic Algorithm (GA) is a search heuristic that imitates the natural process of evolution as it is believed to happen to all the species of living beings. This method uses nature-inspired techniques such as mutation, crossover, inheritance and selection, to generate solutions for optimization problems. The success of a GA depends on the type and complexity of the problem to which it is applied.

In a GA, the chromosomes or individuals are represented as strings which encode candidate solutions for an optimization problem, that later evolve towards better solutions.

The GA evolutionary process starts off by initializing a population of solutions (usually randomly), which will evolve and improve during three main steps:

- Selection: a portion of each successive generation is selected, based on their fitness, in order to breed the new, and probably better fit, generations.
- Reproduction: the selected solutions produce the next generation through mutation and/or crossover, propagating the most crucial changes to the future generations by inheritance.
- Termination: once a stopping criteria is met, the generational process ends.

3.2 Algorithm Details

The GA we developed to solve the TOP takes into account the main elements of the problem: the customers and the vehicles.

There is a set of n customers that must be visited by at most once and only by one vehicle. The customers correspond to the vertices in a network

graph.

A fleet of m vehicles is available, and each vehicle is given a priority list, randomly generated, representing the preferred visiting order of customers, figure 1.

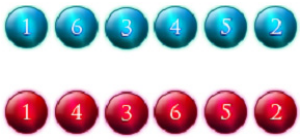

Figure 1: Priority lists for 2 vehicles in an instance with 6 vertices.

In this GA, every chromosome is composed by m sub-chromosomes, each one corresponding to a different priority list of customers (figure 1). The sub-chromosomes are composed by n genes, one for each customer. The i gene of a chromosome corresponds to the i-th (i^{th}) customer to be visited. In other words, it is the customer in the i-th position in a priority list. The allele is the index of a customer in the service network and therefore indicates which customer is visited in a given gene or position. Consequently, the total number of genes in a chromosome is equal to $m \times n$ (one priority list for each vehicle).

The representation of a valid solution is the result of a constructing process, where the vertices are added to the routes by following the order they appear in priority lists for each vehicle. During this process, each vertex (or customer) v is chosen for a route if:

- It is possible to go from the last added vertex to v and from v directly to the final vertex, without violating the time limit.
- Vertex v does not belong to another vehicle's route.

In figure 2 it is presented a graphic representation of a feasible solution for a 6-vertices instance with two vehicles, A (red line) and B (blue line). Therefore, two routes, one for each vehicle, were calculated:

- Route A – The solution contains vertices 1-6-3-4, and these vertices become automatically unavailable for vehicle B. Vertices 5 and 2 are not visited because the time limit for the route is exceeded.
- Route B – Since the route determined for vehicle A already picked up four vertices from the priority list of vehicle B, the next available vertex to be added is vertex 5, concluding the route since adding vertex 2 means disrespecting the time constraint.

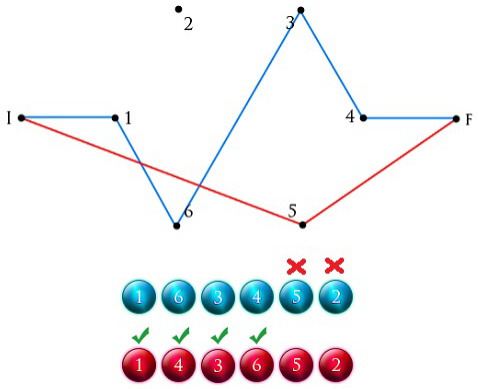

Figure 2: Representation of a possible solution for 2 vehicles in a 6-vertices instance.

A fitness function is used in the genetic algorithm to evaluate the overall fitness of a chromosome, which corresponds to the total reward collected in a reached solution. The quality of this information highly influences the reproductive process.

The implemented GA follows a very simple and common operation structure:

```
Begin
   P ← Initialize (population);
   S ← Selection (P);
   While stop criteria not met do:
      O ← Reproduction (S);
      P ← O + ΔP
         S ← Selection (P);
End
```

Where P is the population of potential solutions (chromosomes) at the moment, S is the group of solutions within P, selected to produce the offspring O, which is the next generation of solutions. As for ΔP, it represents a residual part of P that contains non-selected solutions, used to promote diversification in the reproductive process and avoid early convergence to local optimal solutions.

During the reproduction phase, two procedures take place: crossover and mutation. The crossover operation involves the combination of two chromosomes in order to obtain a better one by inheriting the good genes from each parent. In our algorithm the process goes by removing a block of consecutive genes from one parent solution and placing it in a random position within the other parent solution. The block of genes has a random size, and the insertion position may vary randomly. After the insertion, the repeated vertices in the solution are removed to maintain the correct size of the sub-chromosomes.

The schematic in figure 3 shows how the crossover procedure is done. In this example, the crossover occurs between two chromosomes, *A* and *B*, with two sub-chromosomes each. At first, a block of consecutive genes is removed from B. Then, the same block is inserted in *A* within a randomly chosen sub-chromosome, in an arbitrary position. The removal of repeated vertices starts in the position where the sub-chromosome of A was modified, therefore preventing the removal of the new added vertices. The new generated solution is then ready to struggle for a spot in the next generation.

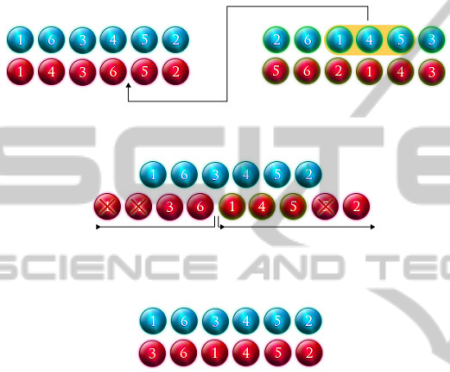

Figure 3: The crossover procedure.

In the other reproduction method, mutation, a valid solution can alter its own genetic code by randomly modifying the genes order. In this case we decided to perform a simple position switch by first selecting one vertex (or gene) from one of the priority lists and exchange with the vertex right after that selected one, in the same list. An example on how this process works can be observed in figure 4.

The individuals are selected to be reproduced by Crossover and Mutation according to a triangular distribution, where the higher the fitness of a chromosome, the higher the probability it has to take part in the reproduction process. In our algorithm there are parameters that set the number of mutations and crossovers that may occur during each reproduction phase (optimization cycle).

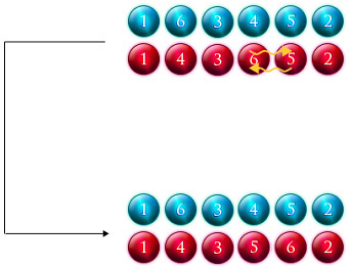

Figure 4: The mutation procedure.

Other parameters allow changing the number of randomly generated chromosomes at each new iteration, and the number of immune chromosomes (unchangeable between iterations). It is also possible to set the algorithm to just accept crossovers and/or mutations that improve the group of solutions. In the case of just accepting improving crossovers, the offspring of two chromosomes is measured in terms of fitness, and if it is more fit than one of its parents, then the new chromosome is kept, otherwise it is discarded.

All the above mentioned parameters will determine the global efficiency of the algorithm in respect to a specific problem, which in this case is the Team Orienteering Problem (TOP).

3.3 Software Developed

A JAVA software application was developed to implement the algorithm described previously. The application has a simple, yet functional, Graphical User Interface (GUI), with a handful of options that allows parameters of the GA to be adjusted and help obtain better results in a specific problem or TOP instance. Although aware of this, we decided to keep the same settings during all the tests so that a comparison with other research authors could be done, as well as an overall evaluation of our GA.

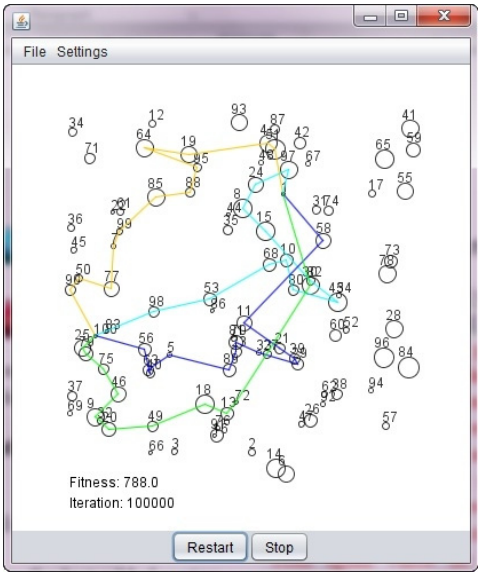

Figure 5: The “Solution Viewer” representing routes while the application solves an instance.

The developed software application can load different instances and it is possible to set the stopping condition. The other previously described parameters can also be changed through the menu.

There is also a visualization element that represents the current loaded instance (figure 5), where the circles correspond to customers in the TOP.

We consider the software is able to run a series of experiments on some well-known test instances.

4 COMPUTATIONAL RESULTS

A series of computational experiments were conducted in order to test the performance of our GA. The computational experiments were performed on 24 of the 320 benchmark instances published by Chao et al. (1996). The instances were chosen in a semi-random way within four different sets, in order to introduce diversity and different degrees of difficulty while testing our algorithm. We compared our results to the ones obtained by Chao et al. (1996), hereafter referred to as CGW, the results achieved by Tang and Miller-Hooks, hereafter referred to as TMH, and also the results produced by the algorithms presented by Archetti et al. (2006), hereafter referred to as AHS. The tests were run on a laptop computer with an Intel Pentium Core 2 Duo 2.53GHz processor and 4GB of RAM.

The chosen stopping criterion was the number of iterations to run the algorithm for each instance. This value was set to 100,000 iterations. There are 100 vertices in the first set, 66 in the second set, 64 in the third, and 102 in the fourth set of instances.

In each set of instances, the location and score of each vertex is identical. An instance is characterized by a number of vehicles, varying between 2 and 4, and by a time limit (Tmax). The best-known solution values for the tested instances were produced by the AHS algorithms.

In order to execute the tests, the following configuration for the GA was set:

- Total Population: 55
- 30 Crossovers per iteration
- 10 Mutations per iteration
- 10 Randomly Generated Individuals
- 5 Immune Individuals
- Only accepts crossovers with improvement
- Accepts all the results from mutations

We ran our algorithm ten times on each instance. The results obtained in the tested instances with our GA algorithm, hereafter referred to as GATOP-1, are presented in Table 1, with the best scores displayed in bold print. The values *fmin* and *fmax* are respectively denoted as the minimum and maximum value obtained with the fitness function.

In the TOP, the fitness value corresponds to the total reward collected or profit made with all the determined routes (one for each available vehicle). The value *fmin* can be considered a guaranteed value, reflecting the overall robustness of the algorithm, along with the average fitness value. The value *fmax* represents the ability of the algorithm to reach good solutions. It is obtained by running the algorithm more than once (ten times in our experiments) and taking benefit of the randomness, resulting in a larger computational time. Computational time of a run, with the default settings, rarely exceeds 8 minutes in the most difficult instances.

As can be observed in Table 1, on 13 of the 24 instances, our algorithm achieved the same value as AHS. Comparing to the other authors, our scores were equal to TMH in 6 instances and equal to CGW in 4. In addition, GATOP-1 outperformed TMH eight times and CGW eleven times.

Table 1: Results achieved with GATOP-1 in the selected benchmark instances.

Instance	GATOP-1			AHS	TMH	CGW
	Avg	*fmin*	*fmax*			
p4.2.e	566.80	514	601	**618**	593	580
p4.2.n	978.40	894	1056	**1171**	1150	1112
p4.3.f	552.40	519	573	**579**	**579**	552
p4.3.i	696.00	624	778	**807**	785	798
p4.4.l	774.40	730	812	**880**	875	847
p4.4.q	941.90	876	1010	**1161**	1124	1084
p5.2.e	180.00	180	**180**	**180**	**180**	175
p5.2.m	829.50	800	**860**	**860**	**860**	855
p5.3.h	260.00	260	**260**	**260**	**260**	255
p5.3.v	1357.00	1310	1395	**1425**	1410	1400
p5.4.o	676.50	660	**690**	**690**	680	675
p5.4.t	1076.00	1035	**1160**	**1160**	1100	**1160**
p6.2.j	913.20	882	**948**	**948**	936	942
p6.2.n	1192.80	1164	1242	**1260**	1260	1242
p6.3.i	630.50	606	**642**	**642**	612	642
p6.3.l	961.80	936	**1002**	**1002**	990	972
p6.4.k	526.80	522	528	528	522	**546**
p6.4.n	1030.80	948	**1068**	**1068**	**1068**	**1068**
p7.2.e	290.00	290	**290**	**290**	**290**	275
p7.2.r	924.10	863	1027	**1094**	1067	1082
p7.3.g	344.00	344	**344**	**344**	**344**	338
p7.3.s	907.30	839	974	**1081**	1061	1064
p7.4.i	366.00	366	**366**	**366**	359	338
p7.4.t	920.40	839	953	**1077**	1067	1066

In respect to the best scores, GATOP-1 fell behind on 12 instances, with a maximum error of 151 and an average error of 57.7. The results produced with GATOP-1, in terms of consistency, are worse than the results achieved by AHS. In some instances, since there is a considerable gap between the *fmin* and *fmax* values, and also between *fmax* and the average values (*Avg* column in Table 1). In addition , in some of the tested benchmark instances, our GA scored considerably less than the best solution for each of those instances, marked in bold print (Table 1). We consider the gap between our scores and the best scores occurred for three reasons. One of them may relate to a non-optimal balance between the genetic operators (crossovers, mutations), the number of new solutions randomly generated, and the number of immutable solutions (auto-immune individuals).

We performed the tests with a fixed parameter configuration, but we also checked if, by proportionally increasing the parameters values, the algorithm would perform better, which in fact did occur but not in a significant way and at the expense of greater computational time. Therefore, improved results may be produced once a better configuration for the GA is determined.

Another reason that might explain some failures with GATOP-1 is the way of constructing feasible solutions. Before the algorithm evaluates the fitness of a chromosome, the process of solution construction takes place. It begins by generating the route for the first vehicle, following the priority list present in the first sub-chromosome. The customers are consecutively added to the route, respecting the ordered list. Once the time limit is reached, the next route starts to be generated. This method can restrict the evolution of the solution, and this process could be improved if it is performed in a parallelized system, so that all the routes are generated at the same time, allowing them to achieve better profits.

We concluded that the last reason for failure could be the lack of another genetic operator in the algorithm. During the tests with GATOP-1, we observed the best solution values achieved and also their graphical representations, and in some cases we noted optimization deficits in the generated routes. For example, we can consider a part of a route where six customers are visited, following the order given by the blue lines in figure 6. The time (*t)* between the vertices is also given. By calculating the total time of the current route, we get the value 15.6, but it is possible to lower this to 10.6 simply by following a different visiting order of customers, for example, going from vertex 1, then to vertex 3, then 2, and finally to vertex 4. So when rearrangement is possible and it decreases the total time consumption of the routes, then the new route should be kept instead of the initial one, since the fitness or total reward is maintained (same customers visited). It can also happen that the time saved by the improved route is enough to add one or more customers to it, and consequently increase the total profit to be collected.

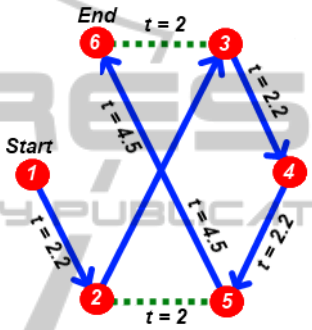

Figure 6: Example situation of an under-optimized route.

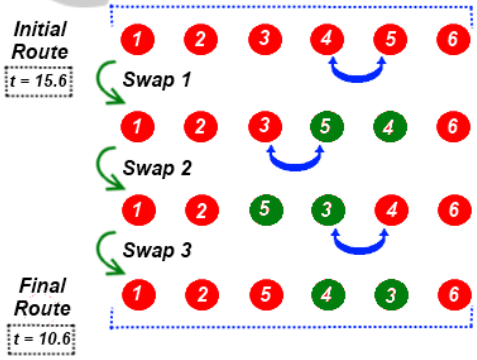

Figure 7: Improving an under-optimized route with a special mutation process.

Our algorithm, GATOP-1, is not able to optimize situations such as the given example with its current features. To overcome this problem, we promote the development of a new and specific mutation operator that can perform rearrangements to the routes between their validation (construction of a feasible solution) and their fitness evaluation. In order to perform properly, the new mutation operator should execute a series of swaps between a group of random yet consecutive genes in a route.

While executing the swaps, unfeasible solutions might be produced, but they should be kept and mutated until a maximum number of swaps are attained. In figure 7, the special mutation process is presented, where consecutive simple swaps take place to optimize the route in figure 6.

5 CONCLUSIONS

In this paper we presented a solution method for the TOP based on a genetic algorithm (GA). Based on our results, we can state that our GA is fairly efficient and attained the best-known solutions in half of the tested instances.

With these experiments, we were able to improve our knowledge in the TOP while aiming for the purposes of the GATOP project. Future research should focus on the development and testing of other methods to apply in a GA to solve the TOP.

The overall performance of the developed GA is fairly acceptable, but there are some small, yet observable, inconsistencies that lie specifically in the gap between the highest and the lowest scores we achieved during the tests. We believe that by avoiding the suggested sources of error, our results will improve significantly in our lowest scores. The usage of dynamic parameters to set the behaviour of the evolution process within the genetic algorithm can be interesting idea to explore. That way, the algorithm would be able to adapt to its own current performance and try to improve it actively.

ACKNOWLEDGEMENTS

This study was partially supported by the project GATOP - Genetic Algorithms for Team Orienteering Problem (Ref PTDC/EME - GIN/120761/2010), financed by national funds by FCT / MCTES, and co-funded by the European Social Development Fund (FEDER) through the COMPETE – Programa Operacional Fatores de Competitividade (POFC) Ref FCOMP-01-0124-FEDER-020609.

REFERENCES

Archetti, C., Hertz, A., Speranza, M. G., 2006. Metaheuristics for the team orienteering problem. In Journal of Heuristics, 13:49-76.

Butt, S. E., Cavalier, T. M., 1994. A heuristic for the multiple tour maximum collection problem. In Computers and Operations Research, 21:101-111.

Chao, I. M., Golden, B., Wasil, E. A., 1996. Theory and Methodology - The Team Orienteering Problem. In European Journal of Operational Research 1996a, 88, 464-474.

Ke, L., Archetti, C., Feng, Z., 2008. Ants can solve the team orienteering problem. In Computers and Industrial Engineering, 54, 648-665.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D., 2010. A Path Relinking Approach for the Team Orienteering Problem. In Computers & Operations Research, Metaheuristics for Logistics and Vehicle Routing, 37 (11), 1853-1859.

Tang, H., Miller-Hooks, E., 2005. A tabu search heuristic for the team orienteering problem. In Computers and Operations Research, 32, 1379-1407.

Tasgetiren, M. F., 2002, A Genetic Algorithm with an Adaptive Penalty Function for the Orienteering Problem. Journal of Economic and Social Research, 4 (2), 1-26.

Vansteewegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D., 2009. A guided local search metaheuristic for the team orienteering problem. In European Journal of Operational Research, 196(1), 118-127.