

A Genetic Algorithm for Multi-component Optimization Problems: The Case of the Travelling Thief Problem

Daniel K.S. Vieira^{1,4}(✉), Gustavo L. Soares², João A. Vasconcelos³,
and Marcus H.S. Mendes⁴

¹ Instituto de Ciências Exatas, Federal University of Minas Gerais,
Belo Horizonte, Brazil
danielv@dcc.ufmg.br

² Pontifical Catholic University of Minas Gerais, Belo Horizonte, Brazil
gsoares@pucminas.br

³ Evolutionary Computation Laboratory, PPGE, Federal University of Minas
Gerais, Belo Horizonte, Brazil
jvasconcelos@ufmg.br

⁴ Instituto de Ciências Exatas e Tecnológicas, Universidade Federal de Viçosa,
Florestal, Brazil
marcus.mendes@ufv.br

Abstract. Real-world problems are often composed of multiple inter-dependent components. In this case, benchmark problems that do not represent that interdependence are not a good choice to assess algorithm performance. In recent literature, a benchmark problem called Travelling Thief Problem (TTP) was proposed to better represent real-world multi-component problems. TTP is a combination of two well-known problems: 0-1 Knapsack Problem (KP) and the Travelling Salesman Problem (TSP). This paper presents a genetic algorithm-based optimization approach called Multi-Component Genetic Algorithm (MCGA) for solving TTP. It aims to solve the overall problem instead of each sub-component separately. Starting from a solution for the TSP component, obtained by the Chained Lin-Kernighan heuristic, the MCGA applies the evolutionary process (evaluation, selection, crossover, and mutation) iteratively using different basic operators for KP and TSP components. The MCGA was tested on some representative instances of TTP available in the literature. The comparisons show that MCGA obtains competitive solutions in 20 of the 24 TTP instances with 195 and 783 cities.

Keywords: Genetic Algorithm · Multi-component problem · Travelling Thief Problem

1 Introduction

Classic problems in computer science have been proposed and have recently been studied to define strategies to obtain a good solution for real-world problems.

Many of these problems belong to the NP-hard class, which means that they are combinatorial problems to which we do not have algorithms that can find the best solution in a polynomial time nowadays. Therefore, it is not possible to obtain the best solution in large instances due to the necessary time to process each possible solution. For this reason, heuristics have been developed to obtain a satisfactory solution in an acceptable time.

According to Bonyadi et al. (2013), real-world problems often have interdependent components and these should not be solved separately. This correlation should be considered for the achievement of better solutions to the overall problem.

In order to cover the complexity of a real-world problem, a new problem called Travelling Thief Problem (TTP) was proposed (Bonyadi et al. 2013). It is a combination of two well-known problems: the 0-1 Knapsack Problem (KP) and the Travelling Salesman Problem (TSP).

Some strategies have been proposed to solve TTP. Faulkner et al. (2015) presents algorithms that focus on manipulating the KP component and obtaining the TSP component from the Chained Lin-Kernighan algorithm (CLK) (Applegate et al. 2003). Bonyadi et al. (2014) use a co-evolutionary approach called CoSolver where different modules are responsible for each component of the TTP, which in turn communicates with others and combines solutions to obtain an overall solution to the problem. In this way, the CoSolver attempts to solve the TTP by manipulating both components at the same time, instead of obtaining a solution for one component using the solution of other component. Mei et al. (2014) seek large-scale TTP instances proposing complexity reduction strategies for TTP with fitness approximation schemes and applying these techniques in a Memetic Algorithm, which outperforms the Random Local Search and Evolutionary Algorithm proposed by Polyakovskiy et al. (2014). Wagner (2016) proposes a swarm intelligence approach based ant colony that builds efficient routes for TTP instead of TSP, and combines it with a packing heuristic. This strategy outperforms state-of-the-art heuristics on instances with up to 250 cities. Finally, an Evolutionary Algorithm that tackles both sub-problems (KP and TSP) at the same time is presented by Lourenço et al. (2016). The computational experiments in this last study used six instances with a number of cities ranging from 51 to 100.

This paper proposes an algorithm based on Genetic Algorithm (GA) concept called Multi-component Genetic Algorithm (MCGA) to solve small and medium instances of TTP. MCGA has four basic steps in each iteration: evaluation of the solution (individual) to know how good this solution is; selection of solutions based on their performance (fitness); crossover the solutions to create new solutions (children) based on the features (chromosome) of existing solutions (parents); disturbance of the solutions by applying some mutation operator that changes their features (change some alleles of the genes of their chromosome). After some iterations (generations) the algorithm tends to achieve good solutions to the problem.

The article is organized as follows. Section 2 describes the TTP and its specifications. In Sect. 3, the Multi-component Genetic Algorithm (MCGA) is defined. Section 4 explains the methodology. Finally, we present the conclusion in Sect. 5, as well as the possibilities for further research.

2 Travelling Thief Problem

According to Polyakovskiy et al. (2014), TTP is defined as having a set of cities $N = \{1, \dots, n\}$ where the distance d_{ij} between each pair of cities i and j is known, with $i, j \in N$. Every city i , except the first, has a set of items $M_i = \{1, \dots, m_i\}$. Each item k in a city i is described by its value (profit) p_{ik} and weight w_{ik} . The candidate solution must visit each city only once and return to the starting city.

Additionally, items can be collected in cities while the sum of the weight of the collected items does not exceed the knapsack maximum capacity W . A rent rate R must be paid by each time unit that is used to finish the tour. v_{max} and v_{min} describes the maximum and minimum speed allowed along the way, respectively.

$y_{ik} \in \{0, 1\}$ is a binary variable equals to 1 if the item k is collected in the city i . W_i specifies the total weight of the collected items when the city i is left. Hence, the objective function for a tour $\Pi = (x_1, \dots, x_n)$, $x_i \in N$ and a picking plan $P = (y_{21}, \dots, y_{nm_i})$ is defined as:

$$Z(\Pi, P) = \sum_{i=2}^n \sum_{k=1}^{m_i} p_{ik} y_{ik} - R \left(\frac{d_{x_n x_1}}{v_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{max} - \nu W_{x_i}} \right) \quad (1)$$

where $\nu = \frac{v_{max} - v_{min}}{W}$. The aim is to **maximize** $Z(\Pi, P)$. The equation is summarized in penalize the profit gains from collected items with a value that represents the total travel time multiplied by the renting rate R .

Figure 1 shows an example of TTP where the number of items for each city is equal to the test instances provided by Polyakovskiy et al. (2014). This case has 2 items to each city (except the first city, that does not have items) and 3 cities in total. Each item $I_{ij}(p_{ij}, w_{ij})$ is described as being the item j of the city

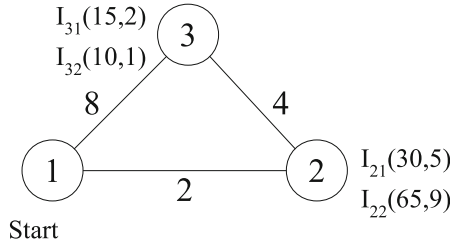


Fig. 1. TTP example.

i that has a value p_{ij} and a weight w_{ij} . Let us assume the knapsack capacity $W = 10$, the renting rate $R = 1$, $v_{max} = 1$ and $v_{min} = 0.1$. A feasible solution for this problem is $P = (0, 1, 0, 1)$ and $\Pi = (1, 2, 3)$, which describes that the items I_{22} , I_{32} which are in the cities 2 and 3, respectively, will be collected, making a tour starting from the city 1 to the city 2, and then to the city 3, and finally returning to the city 1. This solution results in $Z(\Pi, P) \approx -28.053$.

Note that, in this example, if only the TSP component of the problem is considered, all the possible solutions have the same cost, since all connections between the cities will be used anyway. However, considering the overall problem, it can be seen that the order of the tour affects the solution cost, due to the variation of the time that an item is kept in the knapsack. In other words, a heavy item picked at the start of the tour affects the travelling speed for a longer time, compared to the same item picked at the end of the tour, which slows the solution and increases the cost.

3 Multi-component Genetic Algorithm

The proposed MCGA¹ has a different encoding type for each component of the TTP. The TSP component is encoded by enumerating the city indices in order, starting from the city 1 and ending the tour at the same city. The KP component is encoded using a binary array with the size of existing items. The items are ordered according to the city to which they belong. If the problem has 5 items per city, the first five genes make reference to the items of the city two (since the first city, by definition, does not have items). Figure 2 shows a graphical representation of the chromosomes that compose the individual.

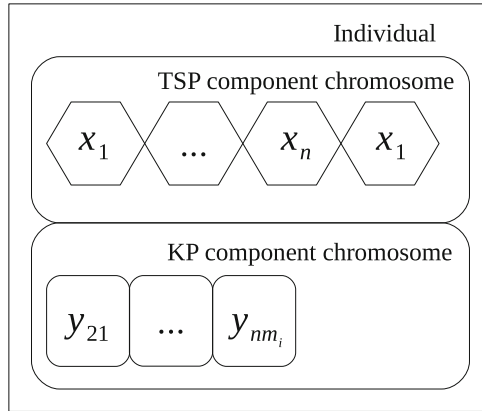


Fig. 2. Graphical representation of the MCGA individual encoding.

¹ The source code can be found at <https://github.com/DanielKneipp/GeneticAlgorithmTravelingThiefProblem>.

The initial population is obtained using the Chained Lin-kernighan (Applegate et al. 2003) (CLK) for the TSP component and all items unpacked (KP component). After generating the initial population, an iterative process starts with the selection of the individuals who will pass through an evolutionary process, based on their fitness. In this step, it was used the tournament method, in which, given a tournament size k , k individuals are selected to be compared with each other and the best individual is selected to the next step. This procedure is repeated until the size of the selected population reaches the size of the original population. Algorithm 1 shows a k -tournament where s^* individuals are selected from population H .

Algorithm 1. Tournament (H, s^*, k)

```

1:  $H\_subset \leftarrow \{\}$ 
2: for  $i := 0$  to  $s^* - 1$  do
3:    $H\_subset \leftarrow k$  random individuals from  $H$ 
4:   add to  $H^*$  the best individual of  $H\_subset$ 
5: end for
6: return  $H^*$ 

```

The crossover step creates new individuals (children) based on the chromosomes of the existing individuals (parents). Due to the multi-component characteristic of the TTP, a different crossover method was used on each component of the problem. For the KP component, a crossover operator called N-point was used. This operator combines the picking plan of two parents in two children in a way that one child receives the alleles of one parent until a certain point is reached. Then, it receives the alleles from the other parent. In the crossover, while the child c_1 receives the alleles from a parent p_i , the child c_2 receives the alleles from a parent p_j , where $\forall i, j \in \{1, 2, \dots, s\} : i \neq j$, and s is the number of parents. The Algorithm 2 shows the N-point crossover operator.

For the TSP component, a crossover operator called Order-based is used to combine the tours of two parents without generating invalid tours (Davis 1985). It uses a random-generated binary mask. When the value in a position of the mask is equal to 1, the gene that is in that position of the children 1 and 2 are filled with the genes of the parents 1 and 2, respectively. The remaining genes in parent 1 (with the mask value equal to 0) are sorted in the same order as they appear in parent 2 and the alleles of those sorted genes are used to fill in the child 1 genes which are still empty. The same happens to child 2, although in this case, the genes of the parent 2 are sorted according to parent 1 order. This operator is detailed in Algorithm 3.

After the application of the crossover operators, the population size doubles due to the creation of two children for each two parents. Hence, a selection procedure is applied to decrease the population back to its original value. This second selection step also uses the Tournament method, as shown in Algorithm 1, using the number of individuals before the crossover step as the population size.

Algorithm 2. N-point crossover (P_{p_1}, P_{p_2})

```

1:  $s \leftarrow$  size of the chromosome.
2:  $Points \leftarrow n$  different loci (positions) in the chromosome sorted in increasing order
3:  $j \leftarrow 0$ 
4: for  $i := 0$  to  $s - 1$  do
5:   if  $j < n, i = Points[j]$  then
6:      $j \leftarrow j + 1$ 
7:   end if
8:   if  $j$  is an even number then
9:      $P_{c_1}[i] \leftarrow P_{p_1}[i]$ 
10:     $P_{c_2}[i] \leftarrow P_{p_2}[i]$ 
11:   else
12:     $P_{c_2}[i] \leftarrow P_{p_1}[i]$ 
13:     $P_{c_1}[i] \leftarrow P_{p_2}[i]$ 
14:   end if
15: end for
16: return  $P_{c_1}, P_{c_2}$ 

```

Algorithm 3. Order-based crossover (Π_{p_1}, Π_{p_2})

```

1:  $s \leftarrow$  size of the chromosome.
2:  $Mask \leftarrow$  random array of bits with size  $s$ 
3: for  $i := 0$  to  $s - 1$  do
4:   if  $Mask[i] = 1$  then
5:      $\Pi_{c_1}[i] \leftarrow \Pi_{p_1}[i]$ 
6:      $\Pi_{c_2}[i] \leftarrow \Pi_{p_2}[i]$ 
7:   end if
8: end for
9:  $L_1 \leftarrow$  list of genes in  $\Pi_{p_1}$  which are in a position  $i$  that the  $Mask[i] = 0$ 
10:  $L_2 \leftarrow$  list of genes in  $\Pi_{p_2}$  which are in a position  $i$  that the  $Mask[i] = 0$ 
11: Sort  $L_1$  so that the genes appear in the same order as in  $\Pi_{p_2}$ 
12: Sort  $L_2$  so that the genes appear in the same order as in  $\Pi_{p_1}$ 
13: Fill in the still empty genes of  $\Pi_{c_1}$  with the  $L_1$ 
14: Fill in the still empty genes of  $\Pi_{c_2}$  with the  $L_2$ 
15: return  $\Pi_{c_1}, \Pi_{c_2}$ 

```

Similarly to the crossover step, the mutation step has a different operator for each component of the problem. To mutate the KP component, a simple Bit-flip operator was used. Each item i has a probability p of being removed in case of i already be selected, or added to the knapsack, otherwise.

2-OPT was used to mutate the TSP component (Watson et al. 1998). This mutation operator swaps a pair of edges in the tour. A way to do that is given two vertexes v_1 and v_2 (they cannot be the first or the last vertex, which reference the same city), it made a copy of the tour until v_1 , then the sub-tour $[v_1, v_2]$ is copied in reverse order. Next, the remainder of the tour is copied. It can be seen in the Algorithm 4, assuming that the index range of the chromosome starts from 0.

Algorithm 4. 2-OPT mutation (Π_p)

```

1:  $s \leftarrow$  size of the chromosome.
2:  $v_1 \leftarrow$  random integer number  $\in [1, s - 3]$ 
3:  $v_2 \leftarrow$  random integer number  $\in [v_1 + 1, s - 2]$ 
4: for  $i := 0$  to  $v_1 - 1$  do
5:    $\Pi_c[i] \leftarrow \Pi_p[i]$ 
6: end for
7: for  $i := v_1$  to  $v_2$  do
8:    $\Pi_c[i] \leftarrow \Pi_p[v_2 - (i - v_1)]$ 
9: end for
10: for  $i := v_2 + 1$  to  $s - 1$  do
11:    $\Pi_c[i] \leftarrow \Pi_p[i]$ 
12: end for
13: return  $\Pi_c$ 

```

In the TSP, the order of the tour does not affect the solution cost, since the distance from a city c_1 to another city c_2 is equal to that from c_2 to c_1 , for example. However, in TTP, this change affects the time that an item will be carried, and consequently, the cost. Therefore, the 2-OPT operator can dramatically change the individual. For this reason, the mutation operators are used with parsimony. The 2-OPT has a 17.5% chance of being chosen, Bit-flip has 65% and both operators have 17.5%. Note that the probability of 82.5% (65% + 17.5%) of Bit-flip being used do not necessarily mean that several items are going to be removed or added because there is the configuration parameter p of the Bit-flip. For example: with $p = 0.2\%$, an item i has a $0.002 \times 0.825 = 0.00165 = 0.165\%$ chance of being added or removed.

We use elitism to maintain a set of the best individuals (also called elites) in each generation to the next generation. In the mutation step, the best individuals are not included in the procedure. In the crossover step, before the procedure starts, a set of the best individuals is ensured to be part of H^* . Note that these best individuals are included in the tournament.

The Eq. 1 is used to evaluate the individuals. If an individual is invalid (the sum of the weight of picked items exceeds the knapsack capacity), a correction procedure is applied and removes the worst items of the individual until it becomes valid. The worst items are those with the lowest values of $\frac{p_i}{w_i}$, where p_i is the value of the item i and w_i its weight.

The MCGA combines all these operators as shown in the Algorithm 5, where K and Q stores the MCGA configuration and the stop conditions, respectively. Inside K , it is specified the population size s ; two different tournament sizes t_s , t_c , for the selection step and the tournament after the crossover, respectively; three different sizes of elite set e_s , e_c , e_m , for the selection step, tournament procedure after the crossover and the mutation step, respectively; a number of points n_c for the N-point crossover operator; the probability p for Bit-flip mutation operator.

Algorithm 5. Multi-component Genetic Algorithm (K, Q)

```

1:  $O \leftarrow \text{GenerateInitialPopulation}(K.s)$ 
2:  $\text{EvaluateIndividuals}(O)$ 
3: while  $\neg Q$  do
4:    $O \leftarrow \text{Select}(O, K.s, K.t_s, K.e_s)$ 
5:    $\Theta \leftarrow \text{Crossover}(O, K.n_c)$ 
6:    $\text{EvaluateIndividuals}(\Theta)$ 
7:    $O \leftarrow \{O, \Theta\}$ 
8:    $O \leftarrow \text{Select}(O, K.s, K.t_c, K.e_c)$ 
9:    $O \leftarrow \text{Mutate}(O, K.p, K.e_m)$ 
10:   $\text{EvaluateIndividuals}(O)$ 
11: end while
12: return  $\text{BestIndividual}(O)$ 

```

4 Methodology and Results

A subset of 9720 TTP benchmark instances, proposed by Polyakovskiy et al. (2014), was used in the experiments. These 9720 instances have the number of cities ranging from 51 to 85,900 (81 different sizes); the number of items per city $F \in \{1, 3, 5, 10\}$ (called item factor); ten capacity categories (varying knapsack capacity); three KP types: *uncorrelated* (the values of the items are not correlated with their weights), *uncorrelated with similar weights* (same as the uncorrelated, but the items have similar weights) and *bounded strongly correlated* (items with values strongly correlated with their weights and likely presence of multiple items with the same characteristics).

According to Faulkner et al. the 72 instances selected by them are a representative subset to cover small, medium, and large size instances with different characteristics of the original set of 9720 instances. In this paper, we selected a subset including 36 instances of those 72 selected by Faulkner et al. (2015). This subset consists of 3 different numbers of cities n between 195 and 3038, two item factors $F \in 3, 10$, all three types of knapsacks t and two capacity categories C being equal to 3 or 7.

Different configurations for the MCGA were defined (based on a simple empirical study) due to the variety of problem sizes, namely:

- C1: 200 individuals, Tournament size of both selection procedures equal to 2, number of elites of both selection steps and mutation step equal to 12, number of points for the N-point crossover operator equal to 3, and probability p for the Bit-flip mutation operator equal to 0.2%. The stop condition is 10 min of runtime. The CLK heuristic has no runtime limit;
- C2: 80 individuals, Tournament size of both selection procedures equal to 2, number of elites of both selection steps and mutation step equal to 6, number of points for the N-point crossover operator equal to 3, and probability p for the Bit-flip mutation operator equal to 0.2%. The stop conditions are 50 min of runtime. The CLK heuristic has runtime limit $t_{CLK} = \frac{0.6 \times t_{MCGA}}{s}$, where

Table 1. MCGA performance compared to the S5 heuristic and MIP approach on a subset (36 instances) of the 72 representative instances used by Faulkner et al. (2015).

n	m	t	C	MCGA	S5	MIP
195	582	bsc	3	84328.5 (± 2169.3)	86516.9	86550.9
			7	121166 (± 5597.43)	110107	110555
		unc	3	61666.25 (± 1629.41)	56510.6	56518
			7	76813 (± 1180.9)	70583.8	70728
		usw	3	31049.1 (± 721.32)	28024.7	28061.5
			7	53808 (± 1110.14)	48023	48332
	1940	bsc	3	229924 (± 2217.35)	227063	227965
			7	385503.5 (± 3464.83)	359614	359527
		unc	3	174049.5 (± 999.1)	157297	157532
			7	250635 (± 996.26)	227502	227637
		usw	3	115036 (± 830.88)	102568	103417
			7	193773 (± 1814.72)	168931	169168
783	2346	bsc	3	288460 (± 7089.17)	263725	263691
			7	478978 (± 11832.2)	435157	433814
		unc	3	207616 (± 3640.77)	189949	189671
			7	287819 (± 4739.09)	263367	263258
		usw	3	140840.5 (± 2420.11)	130409	130901
			7	233716 (± 3593.83)	213893	213943
	7820	bsc	3	942002 (± 7588.29)	940002	940141
			7	1501090 (± 16182.4)	1425821	1424650
		unc	3	581237 (± 5916.8)	637793	637487
			7	905915 (± 9955.25)	910032	909343
		usw	3	399076 (± 9569.79)	434180	435368
			7	707502 (± 15864.72)	698730	699101
3038	9111	bsc	3	798247 (± 19108.8)	1217786	1214427
			7	1312630 (± 35161.7)	1864413	1858773
		unc	3	37385.3 (± 24576.2)	782652	780574
			7	362986 (± 31487.3)	1093259	1090977
		usw	3	93402.2 (± 16183.7)	568509	567102
			7	298075 (± 25109.7)	873670	869420
	30370	bsc	3	1042960 (± 81288.8)	4023124	4006061
			7	1784080 (± 201566)	5895031	5859413
		unc	3	-1576520 (± 44662.3)	2595328	2589287
			7	-897125 (± 117563)	3603613	3600092
		usw	3	-729530 (± 40322.8)	1800448	1801927
			7	-679838 (± 94382.3)	2863437	2856140

t_{MCGA} is the total runtime (50 min in this case) and s is the population size (80 in this case).

Note that the MCGA runtime includes the generation of the initial population procedure. Therefore, the time spent by the CLK is taken into account.

In the computational experiments we compare MCGA with two heuristics proposed by Faulkner et al. (2015): (1) a local search algorithm called S5, which runs CLK and then an iterative greedy heuristic (called PackIterative), until the time is up; and (2) a mixed integer programming (MIP) based approach. The computer used has two Intel Xeon E5-2630 v3 totaling 32 cores (only one was used for each run), 128 GB of RAM and Ubuntu 14.04 LTS OS.

The results presented in Table 1, using C1 configuration in MCGA, represent the mean of 30 independent runs (with standard deviation, in case of MCGA). It can be seen that MCGA outperforms the other algorithms in 83.33% (20 of the 24) instances derived from the TSP problem component **rat** with 195 and 783 cities. On the other hand, for the problem with 3038 cities, it has clearly underperformed and demonstrated that the MCGA performance is visibly linked with the TSP component characteristics, which can be the distance pattern or the problem size or maybe both. The overall result shows that MCGA obtained better averages in 55.56% (20 of the 36) of the presented instances; S5 outperforms the other algorithms in 36.11% (13 of the 36, mainly due to its results in the instances with 3038 cities) and the MIP based approach was the best in 8.33% (3 of the 36) of the benchmark problems.

The algorithm evolution was analyzed for further investigation of the MCGA behavior on instances with 3038 cities. We discovered that, for many instances such as **pcb3038_n30370_uncorr_07** (3038 cities, 10 items per city, uncorrelated type and capacity category 7), the performance was limited by time restriction. Figure 3 shows that MCGA convergence was not attained. Probably, a better solution can be found with more time available. In face of this, a configuration with more time available (C2) was used to analyze the MCGA behavior while solving the instance (**pcb3038_n30370_uncorr_07**). We limited the CLK

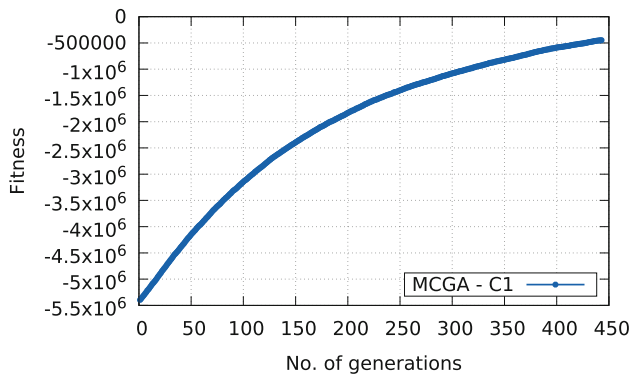


Fig. 3. MCGA with C1 configuration in the instance **pcb3038_n30370_uncorr_07**.

heuristic runtime (set to 37.5 s to generate each individual) in order to guarantee considerable runtime to MCGA.

Figure 4 shows that MCGA achieves significantly higher fitness value if more time is available. However, this value still remains below the S5 and MIP results. Therefore, the amount of time is not the only MCGA feature that requires improvement.

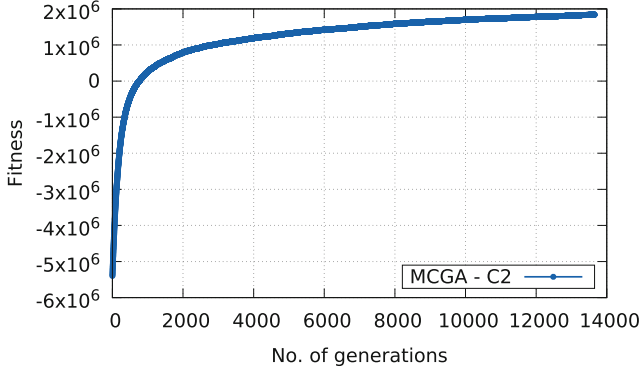


Fig. 4. MCGA with C2 configuration in the instance `pcb3038_n30370_uncorr_07`.

5 Conclusion and Future Work

In this paper, a Genetic Algorithm approach called Multi-component Genetic Algorithm (MCGA) was proposed in an attempt to solve a new multi-component combinatorial problem called Travelling Thief Problem (TTP). It applies basic mutation and crossover operators for each component of the problem. The interdependence observed in a multi-component problem proves challenging, since the optimal solution for a component does not imply in a good solution for the overall problem.

The experiments showed that MCGA can obtain a competitive solution in 20 of the 24 TTP representative instances with 195 and 783 cities when compared to other algorithms in the literature. Further investigation is required to improve its performance on larger instances. The initial ideas are the using of new strategies to evaluate the individuals, and different operators of mutation and crossover specifically developed for TTP that consider the interaction between the components. Moreover, we intended assess how the MCGA parameters influences the quality of the results.

Acknowledgments. This work has been supported in part by the Brazilian agencies CAPES, CNPq, and FAPEMIG.

References

- Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. *INFORMS J. Comput.* **15**(1), 82–92 (2003)
- Bonyadi, M.R., Michalewicz, Z., Przybyłóek, M.R., Wierzbicki, A.: Socially inspired algorithms for the travelling thief problem. In: *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, pp. 421–428. ACM (2014)
- Bonyadi, M., Michalewicz, Z., Barone, L.: The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In: *2013 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1037–1044, June 2013
- Davis, L.: Applying adaptive algorithms to epistatic domains. *IJCAI* **85**, 162–164 (1985)
- Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pp. 385–392. ACM (2015)
- Lourenço, N., Pereira, F.B., Costa, E.: An evolutionary approach to the full optimization of the traveling thief problem. In: Chicano, F., Hu, B., García-Sánchez, P. (eds.) *EvoCOP 2016. LNCS*, vol. 9595, pp. 34–45. Springer, Heidelberg (2016). doi:[10.1007/978-3-319-30698-8_3](https://doi.org/10.1007/978-3-319-30698-8_3)
- Mei, Y., Li, X., Yao, X.: Improving efficiency of heuristics for the large scale traveling thief problem. In: Dick, G., et al. (eds.) *SEAL 2014. LNCS*, vol. 8886, pp. 631–643. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-13563-2_53](https://doi.org/10.1007/978-3-319-13563-2_53)
- Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, GECCO 2014*, pp. 477–484. ACM, New York (2014). <http://doi.acm.org/10.1145/2576768.2598249>
- Wagner, M.: Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem. In: Dorigo, M., Birattari, M., Li, X., López-Ibáñez, M., Ohkura, K., Pinciroli, C., Stützle, T. (eds.) *ANTS 2016. LNCS*, vol. 9882, pp. 273–281. Springer, Heidelberg (2016). doi:[10.1007/978-3-319-44427-7_25](https://doi.org/10.1007/978-3-319-44427-7_25)
- Watson, J., Ross, C., Eisele, V., Denton, J., Bins, J., Guerra, C., Whitley, D., Howe, A.: The traveling salesrep problem, edge assembly crossover, and 2-opt. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) *PPSN 1998. LNCS*, vol. 1498, pp. 823–832. Springer, Heidelberg (1998). doi:[10.1007/BFb0056924](https://doi.org/10.1007/BFb0056924)

Evolutionary Computation in Combinatorial Optimization
17th European Conference, EvoCOP 2017, Amsterdam,
The Netherlands, April 19-21, 2017, Proceedings

Hu, B.; López-Ibáñez, M. (Eds.)

2017, XII, 249 p. 46 illus., Softcover

ISBN: 978-3-319-55452-5