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

Daniel K. S. Vieira
Federal University of Viçosa, Brazil
daniel.sa@ufv.br

Marcus H. S. Mendes
Federal University of Viçosa, Brazil
marcus.mendes@ufv.br

ABSTRACT

Real-world problems many times are characterized by being composed by multiple interdependent components. In this case, benchmark problems that do not represent that interdependency are not a good choice to assess algorithms performance. Recently in the literature a benchmark problem called Travelling Thief Problem 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. The ideia is solve the overall problem instead of each sub-component separately. Starting from 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 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 for TTP instances with a number of cities between 51 and 783.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Constrained optimization
; I.2.8 [**Problem Solving, Control Methods, and Search**]: Multi-component combinatorial problem

General Terms

Algorithms

Keywords

Travelling Thief Problem; Genetic Algorithm; Combinatorial problem optimization

1. INTRODUCTION

Classic problems in computer science have been proposed and these days are studied to define strategies that obtain a good solution efficiently on real-world problems. Many of these problems belong to NP-hard class, which means that they are combinatorial problems that you cannot find the best solution in a polynomial time, implying that in large instances it is not possible to get the best solution due to the necessary time to process each possible solution. Because of that heuristics are developed to obtain a satisfactory solution in an acceptable time.

According to Bonyadi et al. [2] real-world problems often have interdependent components and these should not be solved separately. This correlation should be considered to get 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) [2] was proposed. It is a combination of two well-known problems that are the Knapsack Problem (KP) and the Travelling Salesman Problem (TSP).

Some strategies have been proposed with the aim to solve TTP. Faulkner et al. [4] presents algorithms that focus on manipulating the KP component with the TSP component being obtained by the Chained Lin-Kernighan algorithm (CLK) [1]. Bonyadi et al. [3] uses a co-evolutionary approach called CoSolver where different module are responsible for each component of the TTP and these communicate with each other combining solutions to get a overall solution to the problem. In this way the CoSolver attempts to solve the TTP manipulating both components at the same time instead of get a solution for one component given the solution of other component. Mei et al. [5] seeks large-scale TTP instances proposing complexity reduction strategies for TTP with fitness approximation schemes and apply this techniques in a Memetic Algorithm which outperforms the Random Local Search and Evolutionary Algorithm proposed by Polyakovskiy et al. [7]. Oliveira et al. [6] developed a Tabu Search (TS) approach with a 2-OPT local search that shows competitive performance for very small instances of TTP.

This paper proposes a new algorithm based on Genetic Algorithm (GA) concept called Multi-component Genetic Algorithm (MCGA) to solve small and medium sizes of TTP instances which 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); disturb the solutions applying some mutation operator that change 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. The Section 2 describes the TTP and its specifications. In Section 3 the Multi-component Genetic Algorithm (MCGA) is defined. The Section 4 shows the methodology. Finally, we present the conclusion in Section 5 as well as the possibilities for future work.

2. TRAVELLING THIEF PROBLEM

According to Polyakovskiy et al. [7], 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 but 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. υ_{max} and υ_{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:

$$\begin{aligned} Z(\Pi, P) &= \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} \\ &- R \left(\frac{d_{x_n x_1}}{\upsilon_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\upsilon_{max} - \nu W_{x_i}} \right) \end{aligned} \tag{1}$$

where $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$. 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.

The figure 1 shows an example of TTP where there is the same number of items for each city as the test instances provided by Polyakovskiy et al. [7]. 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 i that has a value p_{ij} and a weight w_{ij}. Assuming the knapsack capacity $W = 10$, the renting rate $R = 1$, $\upsilon_{max} = 1$ e $\upsilon_{min} = 0.1$. A feasible solution for this problem is $P = (0, 1, 0, 1)$ and $\Pi = (1, 2, 3)$, describing 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 goes to the city 2, and then goes to the city 3, and finally returns to the city 1. This solution results in $Z(\Pi, P) \approx -28.053$.

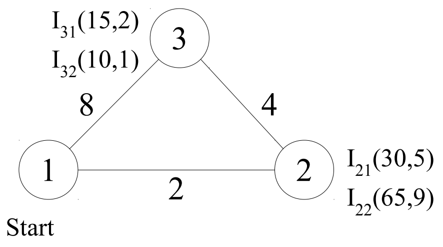

Figure 1: TTP example.

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. But considering the overall problem, it can be seen that the order of the tour makes influence in the solution cost due to the variation of the time that an item is kept into the knapsack. In other words, a heavy item picked at the start of the tour influences the travelling speed for a longer time compared to the same item picked at the end of the tour, making the solution slower and increasing the cost.

3. MULTI-COMPONENT GENETIC ALGORITHM

The proposed MCGA[1] has a different encoding type for each component of the TTP. The TSP component is encoded 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 that the items belong to. 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). The Figure 2 shows a graphical representation of the chromosomes that composes the individual.

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

The crossover step performs the creation of 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

[1]The source code can be found at `https://github.com/DanielKneipp/GeneticAlgorithmTravelingThiefProblem`

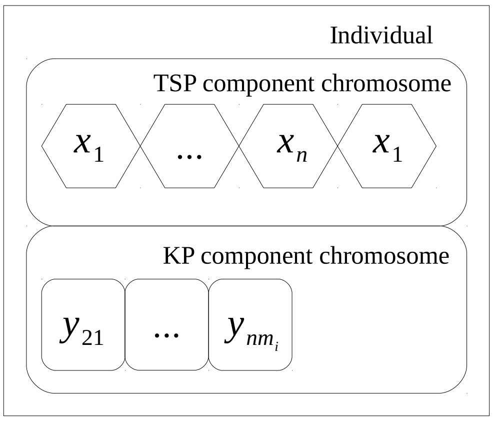

Figure 2: Graphical representation of the MCGA individual encoding.

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

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

on each component of the problem. For the KP component a crossover operator with N-points was used (the number of points changes for each configuration tested and will be defined in Section 4). This operator combine the picking plan of two parents in two children in a way that one child receives the alleles of one parent until a point is reached, after that 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 $\forall\ i,\ j \in \{1, 2, \ldots, s\} : i \neq j$, with s being the number of parents. The Algorithm 2 shows the N-points crossover operator.

Algorithm 2 N-points 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 $P_{c_1},\ P_{c_2}$
```

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

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 $\Pi_{p_1}$ that are in a position $i$ that
   the $Mask[i] = 0$
10: $L_2 \leftarrow$ list of genes in $\Pi_{p_2}$ that are in a position $i$ that
   the $Mask[i] = 0$
11: Sort $L_1$ so that the genes appear in same order as in
   $\Pi_{p_2}$
12: Sort $L_2$ so that the genes appear in 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 $\Pi_{c_1},\ \Pi_{c_2}$
```

After the crossover operators are applied 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 as population size the number of individuals before the crossover step.

As well as 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, which given an item i and a random value $r_i \in [1, 0]$, $i := 0$ if $i = 1$ and $r > p$, p being the probability of item be removed from oo added to the knapsack. Of course for each item a different r_i is generated.

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

In the TSP the order of the tour does not impact the solution cost since the distance from a city c_1 to another city c_2 is the same of c_2 to c_1. However, in TTP this change affect the time that an item will be carried, consequently affecting the cost. Therefore, the 2-OPT operator can cause a huge change on the individual. For this reason the mutation oper-

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 $\Pi_c$
```

ators are not applied always for all individuals. The Bit-flip has 65% chance to be applied, 2-OPT has $17,5\%$ and it has $17,5\%$ chance of both operators to be applied.

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 steps before the procedure starts the set of best individuals are ensured to be part of H^*, note that this best individuals are included in the tournament.

The Equation 1 is used to evaluate the individuals. If an individual is invalid (sum of the weight of picked items exceed the knapsack capacity) a correction procedure is applied which removes the worst items of the individual until he becomes valid. The worst items are characterized as having 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 combine all these operator as shown in the Algorithm 5 where K and Q stores the MCGA configuration and the stop conditions, respectively. Inside K 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 number of elite solutions 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-points crossover operator; the probability p for Bit-flip mutation operator.

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

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

4. METHODOLOGY AND RESULTS

A subset of 9720 TTP benchmark instances, proposed by Polyakovskiy et al. [7], was used in the experiments. These 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 the 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 has similar weights) and *bounded strongly correlated* (items with values strongly correlated with its weights and it is possible to exist multiple items with the same characteristics).

Different configurations for the MCGA was defined due to the variety of the problem sizes. They are:

C1: 150 individuals, Tournament size of both selection procedures equal to 2, number of elites of both selection steps and the mutation step equal to 15, number of points for the N-point crossover operator equal to 1, and the probability p for the Bit-flip mutation operator equal to 0.5%. The stop conditions are 10 minutes of runtime or 10000 generations without improvement. The CLK heuristic has no runtime limit;

C2: 200 individuals, Tournament size of both selection procedures equal to 2, number of elites of both selection steps and the mutation step equal to 12, number of points for the N-point crossover operator equal to 3, and the probability p for the Bit-flip mutation operator equal to 0.2%. The stop condition is 10 minutes of runtime. The CLK heuristic has no runtime limit;

C3: 80 individuals, Tournament size of both selection procedures equal to 2, number of elites of both selection steps and the mutation step equal to 6, number of points for the N-point crossover operator equal to 3, and the probability p for the Bit-flip mutation operator equal to 0.2%. The stop conditions are 50 minutes of runtime. The CLK heuristic has limit of the runtime $t_{CLK} = \frac{0.6 \times t_{MCGA}}{s}$, where t_{MCGA} is the total runtime (50 minutes 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 order to compare the MCGA results with the TS based heuristic presented by Oliveira et al. [6] we ran the experiments using a computer with CPU Intel Core i7 4810MQ, 8 GB of RAM and Windows 8.1 OS. The MCGA was implemented using the C++ programming language.

The Table 1 shows the mean of the objective function values considering 30 runs of the algorithms: (1 + 1) Evolutionary Algorithm (EA) approach presented by Polyakovskiy et al. [7]; MCGA with C1 configuration; a TS heuristic [6] that according to the authors it has a deterministic characteristic, in other words, the results of different runs in the same conditions are always the same. The instances used have relative small sizes, with the number of cities (n) varying from 50 to 100, a item factor of 1, 3, 5 and 10, all KP types

t and with the knapsack capacity always being of category 10 totaling 60 instances.

The results show that MCGA outperforms the TS in 40% and EA in 98.3% of the tested instances. TS had a notable performance in instances with 53 cities However, it showed a strange behavior in some instances of 99 cities. In the instances with 98 items TS had significantly better results (mean of 7.477 times compared to MCGA) while it had a very poor performance in instances with 490 and 980 items (MCGA achieved results 5.092 times higher on average).

A coefficient c, for each algorithm, was calculated for comparing their performance on the 60 instances presented in Table 1. The calculation of these coefficients was made to avoid the influence of the magnitude variations among different TTP instances in the overall comparison results. The coefficient c is defined by $c = \frac{v}{z}$, where v is the objective value obtained by the algorithm to a given instance and z is the best (highest) objective value to that instance. It makes $c \in [a, 1]$, where a is the coefficient of the worst (smallest) objective value comparing the three algorithms. c will be negative if $v < 0$.

Table 1 shows the mean of the coefficients c (considering all 60 instances) for each algorithm. We can note that MCGA has a more consistent performance in the experiments (mean value of c for MGCA is more close to 1).

To compare MCGA with some heuristics proposed by Faulkner et al. [4] changes on the experiment conditions were made. The computer used has two Intel Xeon E5-2630 v3 totaling 32 cores, 128 GB of RAM and Ubuntu 14.04 LTS OS. The GA configuration used is the C2. The tests conducted by Faulkner et al. [4] show results of the mean of 30 independent runs, different of the tests made using the MCGA and presented in the Table 3 that are the mean of 10 independent runs. A subset of the instances selected by Faulkner et al. [4] was used which consists 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.

The results presented in the Table 3 compares MCGA to a local search algorithm called S5 (showed be highly competitive in many tested instances) that run CLK and then an iterative greedy method heuristic PackIterative until time is up and a mixed integer programming (MIP) based approach.

It can be seen that MCGA outperforms the other algorithms in almost every instance derived from the TSP problem component `rat` with 783 and 195 cities. On the other hand for the problem with 3038 cities it has clearly underperforming making explicit that the MCGA performance is visible linked with the TSP component characteristics, which can be the distances pattern or the problem size or maybe both. The overall result shows that MCGA obtained better averages in 52.78% of the presented instances, the S5 outperforms the other algorithms in 36.11% (due to its results in the instances with 3038 cities) and the MIP based approach was the best in 11.11% of the benchmark problems.

An analyses of the algorithm evolution was performed to

Table 1: MCGA performance compared to the Tabu Search approach [6] and EA [7]. The instances with 100 cities refers to the `kroA100` instances.

n	m	t	MCGA	EA	TS
51	50	bsc	**11396.3**	9604.35	11180.05
		unc	6999.51	6130.09	**7167.77**
		usw	5926.52	5418.48	**5959.69**
	150	bsc	**27481.2**	24121.29	26997.77
		unc	**21949.6**	18780.23	21522.96
		usw	**18362.2**	14843.1	17927.85
	250	bsc	44628.4	33708.71	**49781.12**
		unc	33452.7	29508.08	**34000.52**
		usw	**32826.8**	26483.19	31747.54
	500	bsc	14641.9	7874.7	**15331.3**
		unc	8355.08	7979.65	**9151.72**
		usw	7863.13	7231.54	**8655.27**
52	51	bsc	41971.7	31767.99	**49373.15**
		unc	22171.9	20754.83	**23589.93**
		usw	24238.2	23208.87	**26940.18**
	153	bsc	82137.8	55942.48	**87235.28**
		unc	39862.4	39386.03	**42608.34**
		usw	41719.6	40536.51	**47572.81**
	255	bsc	160041	117278.61	**179376.75**
		unc	87094	86049.77	**94354.89**
		usw	81237.9	79072.25	**93972**
	510	bsc	99301.2	75874.84	**103220.98**
		unc	72161.9	62588.19	**72825.86**
		usw	63153.4	52793.76	**64960.24**
70	69	bsc	**12132.6**	10212.02	12048.55
		unc	13712.9	12234.81	**14356.49**
		usw	**10677.5**	9077.16	10626.42
	207	bsc	**52973.9**	42385.48	49422.32
		unc	**35069.6**	30231.71	34612.74
		usw	**32349**	28591.82	29398.59
	345	bsc	89053.1	69353.76	**89075.65**
		unc	**54735.7**	48568.99	49533.55
		usw	**51838.3**	44832.2	51504.4
	690	bsc	**181097**	142550.43	159478.11
		unc	**110046**	94962.56	96015.28
		usw	**103793**	89263.78	103315.15
99	98	bsc	22637	19910.91	**159018.64**
		unc	15287.2	13975.62	**119652.97**
		usw	13942.3	12527.14	**105671.57**
	294	bsc	66754.1	57418.5	**82861.53**
		unc	43824.9	39675.44	**69423.99**
		usw	41874.2	37202.45	**58207.53**
	490	bsc	**117066**	93366.59	17335.31
		unc	**77495.2**	70168.3	12365.18
		usw	**67510**	60821.47	12972.58
	980	bsc	**232309**	196269.46	51109.27
		unc	**149729**	138636.78	34560.47
		usw	**137806**	126800.27	39954.74
100	99	bsc	19747.5	14749.02	**20353.12**
		unc	14773.2	14149.22	**15432.37**
		usw	13958.9	13879.76	**14133.86**
	297	bsc	54174.1	39072.07	**58010.61**
		unc	42989.5	41997.23	**44628.82**
		usw	40913.1	39463	**43294.08**
	495	bsc	**93475.2**	73185.28	79311.13
		unc	74733.2	73997.67	**77778.99**
		usw	66709.8	66751	**71705.39**
	990	bsc	**176962**	167126.14	171872.05
		unc	**150619**	148490.64	143345.51
		usw	135309	133913.51	**141737.38**

Table 2: Mean of the coefficients calculated from the results presented in Table 1.

	MCGA	EA	TS
Mean coef.	**0.913212685**	0.795308198	0.906058209

Table 3: MCGA performance compared to the S5 heuristic and MIP approach on a subset of the 72 instances used by Faulkner et al. [4].

n	*m*	*t*	*C*	MCGA	S5	MIP
195	582	bsc	3	84385.5	86516.9	**86550.9**
			7	**117426**	110107	110555
		unc	3	**60017.3**	56510.6	56518
			7	**75303.3**	70583.8	70728
		usw	3	**30871.8**	28024.7	28061.5
			7	**54357**	48023	48332
	1940	bsc	3	**229432**	227063	227965
			7	**384304**	359614	359527
		unc	3	**174002**	157297	157532
			7	**249938**	227502	227637
		usw	3	**115337**	102568	103417
			7	**191539**	168931	169168
783	2346	bsc	3	**284904**	263725	263691
			7	**481395**	435157	433814
		unc	3	**206739**	189949	189671
			7	**286227**	263367	263258
		usw	3	**139672**	130409	130901
			7	**232023**	213893	213943
	7820	bsc	3	939702	940002	**940141**
			7	**1483880**	1425821	1424650
		unc	3	577930	**637793**	637487
			7	900138	**910032**	909343
		usw	3	393142	434180	**435368**
			7	**702622**	698730	699101
3038	9111	bsc	3	816356	**1217786**	1214427
			7	1360530	**1864413**	1858773
		unc	3	58605	**782652**	780574
			7	404174	**1093259**	1090977
		usw	3	121877	**568509**	567102
			7	375058	**873670**	869420
	30370	bsc	3	1113600	**4023124**	4006061
			7	2362480	**5895031**	5859413
		unc	3	-1531310	**2595328**	2589287
			7	-717599	**3603613**	3600092
		usw	3	-748034	1800448	**1801927**
			7	-434909	**2863437**	2856140

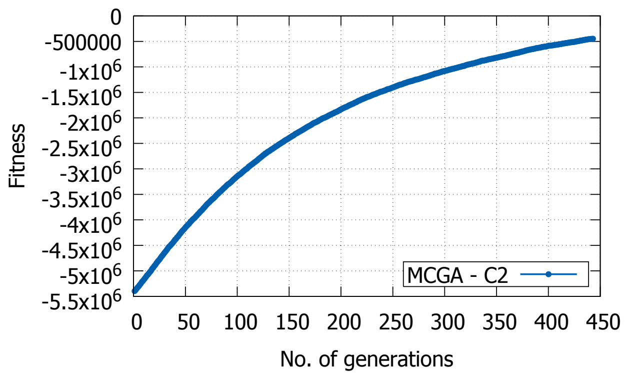

Figure 3: MCGA with C2 configuration in the instance `pcb3038_n30370_uncorr_07`.

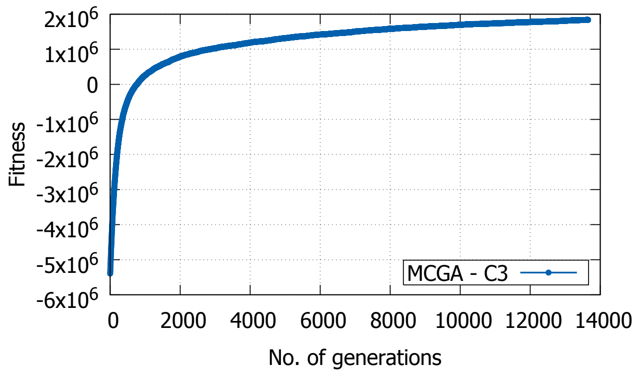

Figure 4: MCGA with C3 configuration in the instance `pcb3038_n30370_uncorr_07`.

further investigate the MCGA behavior on instances with 3038 cities. We discovered that for many instances like the `pcb3038_n30370_uncorr_07` (3038 cities, 10 items per city, uncorrelated type and capacity category 7) the performance was limited by the time restriction. It can be seen in Figure 3 that MCGA convergence was not attained. Probably, better solution can be found with more time. In face of this, a configuration with more time (C3) was used to analyze the MCGA behavior solving the instance (`pcb3038_n30370_uncorr_07`). We limited the CLK heuristic runtime (set to 37.5 seconds to generate each individual) in order to guarantee considerable runtime to MCGA. Moreover, we used the same 32 cores computer that ran the previous experiment.

The Figure 4 shows that MGCA 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 has to be improved.

5. CONCLUSION

The existing interdependence in a multi-component problem proves challenging due to the optimal solution for a component do not imply in a good solution for the overall problem.

In this paper a Genetic Algorithm approach called Multi-component Genetic Algorithm (MCGA) was proposed attempting to solve a new multi-component combinatorial problem called Travelling Thief Problem (TTP). It applies mutation and crossover operators for each component of the problem.

The experiments showed that MCGA can obtain competitive solution for TTP instances with a number of cities varying between 51 and 783 cities compared to other algorithms in the literature. Further investigation is required to improve its performance on larger instances. A possible improvement is reduce the time consumption using new strategies to evaluate the individuals or new operators of mutation and crossover specific developed for TTP that consider the interaction between the components. Tuning the MCGA parameters may lead to significantly higher objective values in instances with specific size range or type.

6. REFERENCES

[1] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[2] M. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1037–1044, June 2013.

[3] M. R. Bonyadi, Z. Michalewicz, M. Roman Przybyŏek, and A. Wierzbicki. Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 421–428. ACM, 2014.

[4] H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner. Approximate approaches to the traveling thief problem. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 385–392. ACM, 2015.

[5] Y. Mei, X. Li, and X. Yao. Improving efficiency of heuristics for the large scale traveling thief problem. In *Simulated Evolution and Learning*, pages 631–643. Springer, 2014.

[6] M. R. Oliveira, A. G. dos Santos, and M. de Freitas Araujo. Uma heurística busca tabu para o problema do mochileiro viajante. In *Simpósio Brasileiro de Pesquisa Operacional*, SBPO '15, 2015.

[7] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 477–484, New York, NY, USA, 2014. ACM.