This article can be cited as A. Barros Miranda, D. Jabba Molinares, C. J. Ardila Hernandez, L. Guzman Reyes and J. Ruiz-Rangel, Adaptation of Parallel Framework to Solve Traveling Salesman Problem Using Genetic Algorithms and Tabu Search, International Journal of Artificial Intelligence, vol. 19, no. 1, pp. 123-137, 2021.

Adaptation of Parallel Framework to Solve Traveling Salesman Problem Using Genetic Algorithms and Tabu Search

Alber Barros Miranda[1], Daladier Jabba Molinares[1], Carlos Julio Ardila Hernandez[1], Luis Guzman Reyes[1] and Jonathan Ruiz-Rangel[2]

[1]Department of Systems Engineering
Universidad del Norte
Barranquilla, Colombia
alberb@uninorte.edu.co
djabba@uninorte.edu.co
lgguzman@uninorte.edu.co
cardila@uninorte.edu.co

[2]Systems Engineering Program
Universidad Simón Bolívar
Barranquilla, Colombia
jruiz1@unisimonbolivar.edu.co

ABSTRACT

The modeling of combinatorial optimization problems has acquired great importance both for research in mathematical processes and for companies starting new projects and seeking to identify the most efficient, beneficial and economic implementation. One of the most investigated combinatorial problems in optimization studies is the traveling salesman problem (TSP). Here we describe a parallel strategy to solve TSP problem using genetic algorithm and Tabu search based on framework(Guzman, N. Ruiz, Ardila, Jabba and Nieto, 2016) which make a solution through parallel processing within a Master-Slave structure. In addition, the proposed approach is compared with an existing algorithm.

Keywords: TSP problem, Parallel program, Genetic algorithm, Tabu search, Combinatorial optimization problems.

2010 ACM Computing Classification System: 11K45,65C05,65M75,68Q10
2012 ACM Computing Classification System: 90C35

1 Introduction

The modeling of combinatorial optimization problems has been widely researched in mathematics. In addition, such models bear important practical applications. For example, modeling of optimization problems is also relevant to companies initiating new projects and interested in selecting optimal alternatives regarding efficiency, benefits and costs(Precup, David, Petriu, Szedlak-Stinean and Bojan-Dragos, 2016; Precup and Tomescu, 2015; Niño, Ardila, Perez and Donoso, 2010). One of the most studied combinatorial problems in optimization analysis is the traveling salesman problem (TSP), which allows the simulation of many current problems in industrial processes or mathematical processes, among others(Luo, Lin and Feng, 2016; Scholz, 2019). Therefore, the study of solutions or tools providing optimal solutions to the TSP is of significant theoretical and applied importance. Many methods and attempts have been proposed to identify optimal and fast solutions to the TSP. Some current proposals, namely those classified as metaheuristic methods, have performed very well at the task. Of such methods, the tabu search and the genetic algorithm(Caballero-Morales, Martinez-Flores and Sanchez-Partida, 2018; Kin-Ming, Wei-Ying, Pak-Kan, Kwong-Sak, Yee and Sui-Tung, 2018) particularly stood out.

Tabu search is an algorithm designed to identify satisfactory solutions to combinatorial optimization problems, being recognized as one of the best-known metaheuristics based on the quality of provided solutions(Karamcheti and Malek, 1991). In addition, the genetic algorithm has been a efficient approach to large-scale optimization problems and mathematical models that allow finding solutions to complex problems linked to other fields of science such as Psychology, Biology and Artificial Intelligence(Ruiz-Rangel, Ardila Hernández, Jabba Molinares and Maradei Gonzalez, 2018). The genetic algorithm produces close approximation solutions under restricted calculations in comparison to other heuristic methods including simulated annealing(Kondo and Watanabe, 2011; Mohammad and Lixin, 2018). For this reason, we believe that the search for optimal solutions to the TSP can be improved through implementation of an algorithm combining the two metaheuristics(Niño, 2012; Nino-Ruiz, Ardila and Capacho, 2018), or specifically, by defining genetic algorithm as the master, and Tabu search as the slave algorithm. The outcome would master algorithm producing a random and initial solution, followed by slave algorithm fine-tuning the initial solution in order to generate a local optimum. The parallel processing technique is also included for the sake of higher efficiency and enhanced quality of the final solution.

2 Combinatorial optimization problems

Different researchers have worked several combinatorial problems; below there is a list of the most studied combinatorial problems in computer science:

2.1 Knapsack problem

The knapsack problem is a complete N-P problem, which refers to a backpack with a certain weight capacity. Each of N objects must be placed in the knapsack taking into account its weight

and value. The aim is to fit as many objects as possible in the knapsack to maximize value and minimize weight, given the weight capacity of the knapsack. Formally(Hajarian, Shahbahrami and Hoseini, 2016), the problem includes the following parameters:

$$W_k = weight\ of\ each\ object\ X_k,\ \ for\ k = 1, 2, .., N. \tag{2.1}$$

$$R_k = value\ associated\ with\ each\ object\ X_k,\ \ for\ k = 1, 2, .., N. \tag{2.2}$$

$$c = maximum\ weight\ capacity\ of\ knapsack. \tag{2.3}$$

Thus, the problem can be formulated as:

$$\max \sum_{k=1}^{N} (R_k X_k) \tag{2.4}$$

Subject to:

$$\sum_{k=1}^{N} (W_k X_k \leq c) \tag{2.5}$$

Various studies have works proposed methodologies to solve this problem(Ohlsson, Peterson and Sderberg, 1993; Chen, Weng and Li, 2010; Ji, Huang, Liu, Liu and Zhong, 2007; Niu and Bi, 2014), and some approximate solutions to real-life problems through modeling of the knapsack problem.

2.2 Minimum coloring problem

The minimum coloring problem consists in finding the smallest integer number of colors $\chi(G)$ required to color a graph G so that there are no two adjacent vertices. The graph G consists of n vertices forming a set $V(G) = v_1, v_2, v_3, ..., v_n$ and m edges forming the set $E(G) = e_1, e_2, e_3, ..., e_m$. Each e_k is incident uniquely with an unordered pair of final vertices v_i and v_j, with $(v_i, v_j) = e_k \in E(G)$ for any given i, j, k. $A(G)$ is an $n * n$ symmetric binary adjacency matrix, where $A_ij = 1$ if $(v_i, v_j) \in E(G)$; and $A_ij = 0$ otherwise(Marappan and Sethumadhavan, 2016); see figure 1(Bensouyad and Saidouni, 2016).

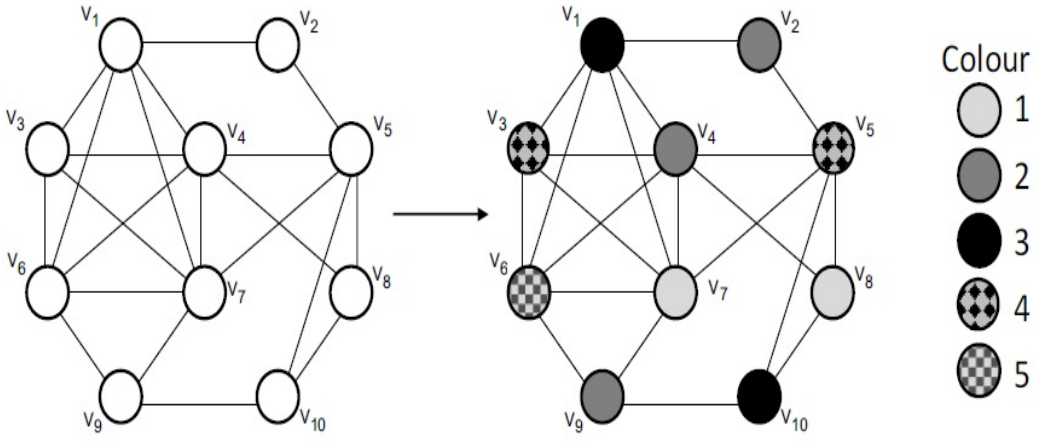

Figure 1: **Minimum coloring problem**.

Different algorithmic approaches have attempted to find the optimal solution to the minimum coloring problem, also bearing in mind the importance of the speed of the algorithm. Multiple solutions based on metaheuristic algorithm have been proposed(Dadaneh, Markid and Zakerolhosseini, 2015; Marappan and Sethumadhavan, 2013; Fister and Brest, 2011; Hong, Vaidya, Lu and Shafiq, 2011; Azlan and Hussin, 2013) and results show that it represents a satisfactory candidate for its solution.

2.3 The traveling salesman problem (TSP)

This problem consists in finding the shortest route, with the minimum travel costs across a list of cities that need to be visited by a salesman(Sharma and Gupta, 2015; Hussain, Shad Muhammad, Nauman Sajid, Hussain, Mohamd Shoukry and Gani, 2017); see Figure 2(Hussain et al., 2017). The cost of the route formula is determined as follows:

$$cost = \sum_{k=1}^{N} (C_{ij}) \tag{2.6}$$

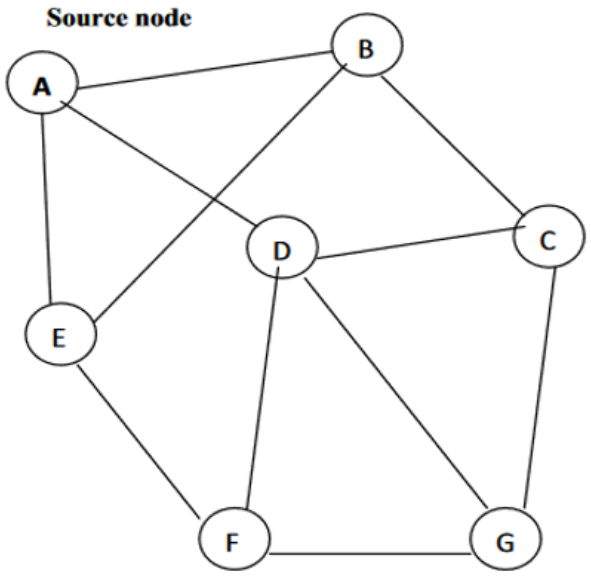

Figure 2: **Representation of the TSP**.

Many real-life the problems can be seen as variants of the TSP, and therefore significant effort has been dedicated to finding optimal solutions to the TSP. The m-crossover operator(Mudaliar and Modi, 2013; Anaya Fuentes, Hernández Gress, SeckTuoh Mora and Medina Marín, 2018) was proposed as a new genetic algorithm to solve this problem. This model produces 18 different descendant chromosomes derived from either of the two possible parental chromosomes, and selects two newly generated chromosomes from them. The crossing operator results in a faster search for best solutions in comparison to alternative models.

Another hybrid scheme was recently proposed(Khan, Khan and Iqbal, 2012) as a way of identifying the shortest traveling distance across each city until the return of the salesman to the

starting point. The scheme tackles the problem by creating a multilevel partition of the graph and then individually solving each partition through the application of the Lin and Kernighan's algorithm. The procedure generates an optimal solution to the TSP and can be applied to other complex problems.

The TSP was also used to test the performance of a new hybrid particle swarm optimization approach(Bouzidi and Riffi, 2014). This method combines particle swarm optimization with the harmonic search algorithm, which improved the swarm. This algorithm showed higher efficiency than any existing metaheuristics. A case study of Anti-Ceva Automotive Logistics Ltd(Liu, Zhang and Du, 2014) simulated the TSP by using relative location coordinates as the parameters of the problem. The simulated annealing algorithm was used to obtain the optimal solutions, with a circular trajectory being generated as a final solution. An agent-based evolutionary search algorithm (AES) was presented to solve the problem of dynamic salesman(Dazhi and Shixin, 2010). The algorithm uses a cooperative learning mechanism in which agents in the current population coevolve to perform optimal dynamic monitoring. In addition, a local update rule is induced and is very similar to the learning permutation scheme utilized by the application to maintain the diversity of dynamic environments. Experimental results show that this method is efficient and has great potential of application to other optimization problems of dynamic combinatory.

3 Solution of the TSP problem through metaheuristic algorithms

Based on previous work,(Guzman et al., 2016) this article proposes a new strategy to solve the combinatorial TSP. The strategy takes advantage of existing metaheuristics by merging them into a parallel processing master-slave structure. Essentially, the proposed algorithm is composed as follows:

Initially, a master (or main) computing node transmits an initial condition to its workers (slaves), and this condition represents a possible initial solution to the TSP. In the first round, the initial solution is randomly generated.

Next, each slave node uses the genetic algorithm and Tabu search to create a scheme including master algorithm and slave algorithm. Within the scheme, the genetic algorithm (master algorithm) identifies a random solution and sends it to the Tabu search (slave algorithm) that processes the delivered solution and returns an improved one. This process is performed in parallel.

Finally, each slave node sends its optimal solution to the master node once optimization has been finalized. Then, the master node chooses the best solution among all slaves. The best solution is returned to the slaves as a new initial condition, and the process is repeated; see Figure 3.

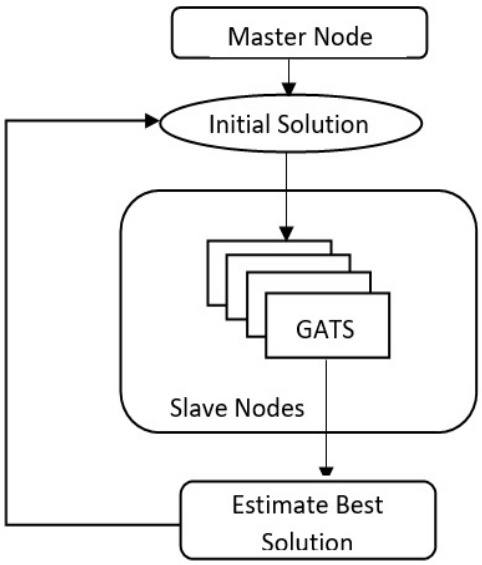

Figure 3: **Diagram with proposed algorithm**.

3.1 Adaptation of genetic algorithm

First, the genetic algorithm defines a population $P(0)$ with a set of initial solutions, and those are then evaluated. Next, a generic operator creates an offspring set through crossing and mutation operations. A new randomly generated population is then added, and the population is reevaluated so that the strongest chromosomes are selected for the next generation. A population $P(t+1)$ with the same number of individuals is sampled, and those steps are repeated until convergence is achieved. See Algorithm 1.

Algorithm 1: GENETIC-ALGORITHM-TSP-PARALLEL

Input: $costM$, $nCity$, $inputcities$, $mutrate$, $xoverrate$

Output: $bestSolution$

$population \leftarrow GenerateInitialPopulation(nCity, inputcities)$
$k \leftarrow 0$
$bestSolution \leftarrow evaluatePopulation(population)$
while $k < nCity$ **do**
    if $ProbabilityCrossover < xoverrate$ **then**
        $population \leftarrow CrossOver(population)$
    end
    if $ProbabilityMutation < mutrate$ **then**
        $population \leftarrow Mutate(population)$
    end
    $bestSolution \leftarrow evaluatePopulation(population)$
    $k \leftarrow k + 1$
end
return$(bestSolution)$

3.2 Adaptation of Tabu search

Initially, the Tabu search algorithm builds a Tabu list with the solution delivered by the genetic algorithm and labels it as the best solution. At each iteration, a new possible solution is created through a search around the neighbors of the current solution, by exchanging one element of the path with another and comparing the cost of each exchange. The exchange engendering the lowest cost then becomes the new compared solution. See Algorithm 2(Guzman et al., 2016). If the new solution is not on the Tabu list and is superior to the current best solution, it replaces the old solution and the Tabu list is updated. The update of the Tabu list consists in the new solution being appended to the list, and the age of each item on the list being calculated. If a solution is old enough, it is removed from the list(Guzman et al., 2016).

Algorithm 2: TABU-SEARCH-TSP-PARALLEL

Input: $costM$, $nCity$, $inputcities$

Output: $bestTour, bestCost$

$previousDistance \leftarrow TourCost(inputcities, costM)$
$bestTour \leftarrow inputcities$
$tempCities \leftarrow inputcities$
$bestCost \leftarrow previousDistance$
$tabuList \leftarrow initialTabu(inputcities)$
$numIter \leftarrow nCity$
$i \leftarrow 1$
while $i \leq numIter$ **do**
    $tempCities \leftarrow GenerateBestNeighborhood(tempCities)$
    if *Not* $IsIn(tempCities, tabuList)$ **then**
        $minDist \leftarrow TourCost(inputcities, costM)$
        if $minDist < bestCost$ **then**
            $bestCost \leftarrow minDist$
            $bestTour \leftarrow tempCities$
        end
    end
    $updateTabuList(tempCities)$
    $i \leftarrow i + 1$
end
return$(bestTour, bestCost)$

3.3 Master-slave algorithmic structure

The proposed master-slave algorithmic structure receives parameters from the main algorithm and enters them into the genetic algorithm, which searches for a random solution for the whole domain. Next, the outcome is entered into the Tabu search algorithm, which performs a local search to refine the solution; see Algorithm 3.

Algorithm 3: MASTER-SLAVE ALGORITHMIC STRUCTURE

Input: $costM$, $nCity$, $inputcities$,$mutrate$,$xoverrate$

Output: $TabuSearchTSPParallel(costM, nCity, inputcities)$

$inputcities \leftarrow GeneticAlgorithmTSPParallel(costM, nCity, inputcities, mutrate, xoverrate)$

return$(TabuSearchTSPParallel(costM, nCity, inputcities))$

3.4 Main algorithm

The main algorithm initially loads the data necessary to solve a specific TSP. Then, a random possible solution is generated and used as the initial parameter of metaheuristic instances, which are created and executed in parallel in each core of the master-slave algorithmic structure. See Algorithm 4. The best solution from each instance is stored on a list, and the best one is selected as the new initial solution in the next iteration for each metaheuristic instance. In the tests, the number of iterations was 10.

Algorithm 4: MAIN ALGORITHM.

$[MatrixCost, nCities] \leftarrow LoadData()$

$bestTour \leftarrow createCity(nCities)$

$minDist \leftarrow TourCost(tempCities, MatrixCost)$

$nNodes \leftarrow getNumberNodes$

$nInstances \leftarrow nNodes/2$

$bestTours[]$

$i \leftarrow 1$

while $i \leq numIter$ **do**

    $bestTours.Add(CallinParallel(MasterSlaveAlgorithmic, nInstances, MatrixCost,$

    $bestTour, nCities))$

    $bestTour \leftarrow getBest(bestTours)$

    $i \leftarrow i + 1$

end

$showBest(bestTour)$

4 Methodology and analysis

In this section, it is presented in more the detail the experimental methodology to test the accuracy of the algorithm proposed to solve the TSP. Results are described and compared in the light of algorithm proposed in (Guzman et al., 2016). Finally, positive and negative aspects of the results are discussed. In the following, section 4.1 describes the hardware specifications of test scenarios. Section 4.2 presents the software environments used to develop and test the method. Section 4.3 present the TSP libraries that execute the two algorithms.

4.1 Hardware Specifications

Implementation of the algorithms was tested in a virtual machine on a Hewlett-Packard server running the Windows 10 Pro 64bit operating system, with 36 Intel (R) Xeon (R) CPU E5-2690 3.00GHz 2.99 GHz processors, 64GB RAM, and 31.5GB hard drive. During the execution of the tests, the machine was not networked to avoid sharing resources with network-related tasks. Finally, the machine was configured to stop any other demanding resource service running on the computer, in order to channel any available resources into the system.

4.2 Software Specifications

Development and compilation of metaheuristic algorithms with parallel processing were implemented in the Matlab programming language. The number of iterations in the tests was 10 using 16 cores.

4.3 TSP libraries

In the execution of the experiments, a selection of instances with symmetric structure from the TSPLIB library is used, providing a large sample of data. The symmetric data structure means that the cost of traveling from city i to city j is similar to the cost of traveling from j to city i, where i and j are cities included in the TSP. Table 1(Guzman et al., 2016) shows names of the TSPLIB samples, information data structure, and number of cities.

Table 1: TSPLIB instances.

N	TSPLIB	Structure	Number of cities
1	a280.tsp	EUCLIDEAN2D	280
2	berlin52.tsp	EUCLIDEAN2D	52
3	bier127.tsp	EUCLIDEAN2D	127
4	ch130.tsp	EUCLIDEAN2D	130
5	ch150.tsp	EUCLIDEAN2D	150
6	d198.tsp	EUCLIDEAN2D	198
7	d493.tsp	EUCLIDEAN2D	493
8	d657.tsp	EUCLIDEAN2D	657
9	eil101.tsp	EUCLIDEAN2D	101
10	eil51.tsp	EUCLIDEAN2D	51
11	eil76.tsp	EUCLIDEAN2D	76
12	kroA100.tsp	EUCLIDEAN2D	100
13	kroA150.tsp	EUCLIDEAN2D	150
14	kroA200.tsp	EUCLIDEAN2D	200
15	kroB100.tsp	EUCLIDEAN2D	100
16	kroB150.tsp	EUCLIDEAN2D	150
17	kroB200.tsp	EUCLIDEAN2D	200
18	kroC100.tsp	EUCLIDEAN2D	100
19	kroD100.tsp	EUCLIDEAN2D	100

Table 1 continued from previous page

20	kroE100.tsp	EUCLIDEAN2D	100
21	lin105.tsp	EUCLIDEAN2D	105
22	lin318.tsp	EUCLIDEAN2D	318
23	linhp318.tsp	EUCLIDEAN2D	318
24	p654.tsp	EUCLIDEAN2D	654
25	pcb442.tsp	EUCLIDEAN2D	442
26	pr76.tsp	EUCLIDEAN2D	76
27	pr107.tsp	EUCLIDEAN2D	107
28	pr124.tsp	EUCLIDEAN2D	124
29	pr136.tsp	EUCLIDEAN2D	136
30	pr144.tsp	EUCLIDEAN2D	144
31	pr152.tsp	EUCLIDEAN2D	152
32	pr226.tsp	EUCLIDEAN2D	226
33	pr264.tsp	EUCLIDEAN2D	264
34	pr299.tsp	EUCLIDEAN2D	299
35	pr439.tsp	EUCLIDEAN2D	439
36	rat99.tsp	EUCLIDEAN2D	99
37	rat195.tsp	EUCLIDEAN2D	195
38	rat575.tsp	EUCLIDEAN2D	575
39	rat783.tsp	EUCLIDEAN2D	783
40	rd100.tsp	EUCLIDEAN2D	100
41	rd400.tsp	EUCLIDEAN2D	400
42	st70.tsp	EUCLIDEAN2D	70
43	ts225.tsp	EUCLIDEAN2D	225
44	tsp225.tsp	EUCLIDEAN2D	225
45	u159.tsp	EUCLIDEAN2D	159
46	u574.tsp	EUCLIDEAN2D	574
47	u724.tsp	EUCLIDEAN2D	724
48	gil262.tsp	EUCLIDEAN2D	262
49	fl417.tsp	EUCLIDEAN2D	417

Data are represented in a 2D Euclidean structure, and therefore the cost associated with a given traveling distance is calculated with the Euclidean distance function (Guzman et al., 2016):

$$d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4.1}$$

4.4 Results

Here it is compared the results from the two TSP algorithms, which were executed in the same environment for each instance of the TSPLIB library. Table 2(Guzman et al., 2016) shows the optimum value of the library for each instance, the value obtained from the two algorithms, and the relative error of each. The relative error is obtained by subtracting the optimal value from the obtained solution.

Table 2: **Comparison between optimal value and obtained solutions.**

N	TSPLIB	Optimal value	Values of proposed algorithm	Errors of the proposed algorithm	Values from Algorithm	Errors from Algorithm
1	a280	2.579,00	4.987,30	93,38%	6.631,50	157,13%
2	berlin52	7.542,00	8.151,90	8,09%	9.023,90	19,65%
3	bier127	118.282,00	140.760,00	19,00%	161.770,00	36,77%
4	ch130	6.110,00	8.232,90	34,74%	10.757,00	76,06%
5	ch150	6.528,00	9.359,00	43,37%	12.451,00	90,73%
6	d198	15.780,00	20.978,00	32,94%	37.183,00	135,63%
7	d493	35.002,00	65.744,00	87,83%	82.927,00	136,92%
8	d657	48.912,00	127.000,00	159,65%	142.310,00	190,95%
9	eil101	629,00	761,81	21,11%	1.645,70	161,64%
10	eil51	426,00	461,99	8,45%	853,69	100,40%
11	eil76	538,00	608,95	13,19%	1.362,60	153,27%
12	kroA100	21.282,00	26.800,00	25,93%	38.583,00	81,29%
13	kroA150	26.524,00	37.300,00	40,63%	48.308,00	82,13%
14	kroA200	29.368,00	47.200,00	60,72%	57.510,00	95,83%
15	kroB100	22.141,00	27.600,00	24,66%	37.757,00	70,53%
16	kroB150	26.130,00	36.600,00	40,07%	45.250,00	73,17%
17	kroB200	29.437,00	47.300,00	60,68%	61.941,00	110,42%
18	kroC100	20.749,00	26.600,00	28,20%	36.000,00	73,50%
19	kroD100	21.294,00	26.300,00	23,51%	30.851,00	44,88%
20	kroE100	22.068,00	27.700,00	25,52%	37.249,00	68,79%
21	lin105	14.379,00	18.300,00	27,27%	24.638,00	71,35%
22	lin318	42.029,00	84.900,00	102,00%	116.570,00	177,36%
23	linhp318	41.345,00	85.400,00	106,55%	114.340,00	176,55%
24	p654	34.643,00	74.700,00	115,63%	341.980,00	887,15%
25	pcb442	50.778,00	118.000,00	132,38%	124.490,00	145,17%
26	pr76	108.159,00	123.000,00	13,72%	153.300,00	41,74%
27	pr107	44.303,00	56.100,00	26,63%	117.380,00	164,95%
28	pr124	59.030,00	82.000,00	38,91%	142.460,00	141,33%
29	pr136	96.772,00	136.000,00	40,54%	166.960,00	72,53%
30	pr144	58.537,00	89.100,00	52,21%	136.060,00	132,43%
31	pr152	73.682,00	101.000,00	37,08%	201.160,00	173,01%
32	pr226	80.369,00	141.000,00	75,44%	322.160,00	300,85%
33	pr264	49.135,00	80.400,00	63,63%	206.910,00	321,11%
34	pr299	48.191,00	97.600,00	102,53%	133.020,00	176,03%
35	pr439	107.217,00	225.000,00	109,85%	340.140,00	217,24%

Table 2 continued from previous page

36	rat99	1.211,00	1.510,00	24,69%	2.942,00	142,94%
37	rat195	2.323,00	3.690,00	58,85%	5.687,50	144,83%
38	rat575	6.773,00	16.260,00	140,07%	17.356,00	156,25%
39	rat783	8.806,00	25.048,00	184,44%	26.376,00	199,52%
40	rd100	7.910,00	9.866,00	24,73%	11.942,00	50,97%
41	rd400	15.281,00	32.295,00	111,34%	38.770,00	153,71%
42	st70	675,00	734,66	8,84%	1.711,20	153,51%
43	ts225	126.643,00	256.330,00	102,40%	301.380,00	137,98%
44	tsp225	3.916,00	6.418,90	63,91%	8.386,20	114,15%
45	u159	42.080,00	64.611,00	53,54%	86.214,00	104,88%
46	u574	36.905,00	93.203,00	152,55%	99.177,00	168,74%
47	u724	41.910,00	120.080,00	186,52%	127.280,00	203,70%
48	gil262	2.378,00	4.214,50	77,23%	6.506,70	173,62%
49	fl417	11.861,00	23.483,00	97,98%	64.332,00	442,38%

Comparing the relative error of each of the algorithms against the optimal solution, it is seen that the results obtained by the proposed algorithm are closer to the optimal solution. In other words, it is achieved greater convergence using the proposed algorithm. On the other hand, it is inferred that as the iterations increase and the number of workers increases, the proposed algorithm arrives faster at a closer solution, without any possibility that the algorithm only finds a local optimum due to the random behavior of the genetic algorithm.

5 Conclusions

The results of the tests indicate that our parallel algorithm improves the TSP solution in comparison with (Guzman et al., 2016) framework, providing better convergence to the optimal value. In addition, it can be concluded that a greater number of iterations and nodes leads to improved solutions, with no danger of the proposed algorithm stagnating at a local optimum due to the randomness of the genetic algorithm. An increase in the number of iterations in each node may lead to a considerable improvement in the quality of solutions. The performance of the proposed algorithm should also be evaluated in more robust computational systems.

Acknowledgment

This work was supported by Yahwed God, the Lord Jesus Christ and Holy Spirit. He made this possible.

References

Anaya Fuentes, G. E., Hernández Gress, E. S., SeckTuoh Mora, J. C. and Medina Marín, J. 2018. Solution to travelling salesman problem by clusters and a modified multi-restart iterated local search metaheuristic, *PLoS ONE* (2): 1–20.

Azlan, A. and Hussin, N. M. 2013. Implementing graph coloring heuristic in construction phase of curriculum-based course timetabling problem, *2013 IEEE Symposium on Computers and Informatics (ISCI)* (1): 25–29.

Bensouyad, M. and Saidouni, D. 2016. A guide to graph colouring. algorithms and applications, *2016 Springer International Publishing* (1): 1–25.

Bouzidi, M. and Riffi, M. E. 2014. Discrete novel hybrid particle swarm optimization to solve travelling salesman problem, *2014 5th Workshop on Codes, Cryptography and Communication Systems (WCCCS)* (2): 17–20.

Caballero-Morales, S.-O., Martinez-Flores, J.-L. and Sanchez-Partida, D. 2018. Dynamic reduction-expansion operator to improve performance of genetic algorithms forthe traveling salesman problem, *Hindawi Mathematical Problems in Engineering* (1): 1–10.

Chen, M. R., Weng, J. and Li, X. 2010. A novel multiobjective optimization algorithm for 0/1 multiobjective knapsack problems, *2010 5th IEEE Conference on Industrial Electronics and Applications* (1): 1511–1516.

Dadaneh, B. Z., Markid, H. Y. and Zakerolhosseini, A. 2015. Graph coloring using intelligent water drops algorithm, *2015 23rd Iranian Conference on Electrical Engineering* (1): 595–600.

Dazhi, w. and Shixin, L. 2010. An agent-based evolutionary search for dynamic travelling salesman problem, *2010 WASE International Conference on Information Engineering* (2): 111–114.

Fister, I. and Brest, J. 2011. Using differential evolution for the graph coloring, *2011 IEEE Symposium on Differential Evolution (SDE)* (1): 1–7.

Guzman, L. G., N. Ruiz, E. D., Ardila, C. J., Jabba, D. and Nieto, W. 2016. A novel framework for the parallel solution of combinatorial problems implementing tabu search and simulated annealing algorithms, *2016 6th International Conference on Computers Communications and Control (ICCCC)* (1): 259–263.

Hajarian, M., Shahbahrami, A. and Hoseini, F. 2016. A parallel solution for the 01 knapsack problem using firefly algorithm, *2016 1st Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)* (1): 25–30.

Hong, Y., Vaidya, J., Lu, H. and Shafiq, B. 2011. Privacy-preserving tabu search for distributed graph coloring, *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing* (2): 951–958.

Hussain, A., Shad Muhammad, Y., Nauman Sajid, M., Hussain, I., Mohamd Shoukry, A. and Gani, S. 2017. Genetic algorithm for traveling salesman problem with modified cycle crossover operator, *HindawiComputational Intelligence and Neuroscience* (2): 1–7.

Ji, J., Huang, Z., Liu, C., Liu, X. and Zhong, N. 2007. An ant colony optimization algorithm for solving the multidimensional knapsack problems, *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07)* (2): 10–16.

Karamcheti, V. and Malek, M. 1991. A tsp engine for performing tabu search, *Proceedings of the International Conference on Application Specific Array Processors* (2): 309–321.

Khan, A. A., Khan, M. U. and Iqbal, M. 2012. Multilevel graph partitioning scheme to solve traveling salesman problem, *2012 Ninth International Conference on Information Technology - New Generations* (1): 458–463.

Kin-Ming, L., Wei-Ying, Y., Pak-Kan, W., Kwong-Sak, L., Yee, L. and Sui-Tung, M. 2018. A genetic algorithm with new local operators for multiple traveling salesman problems, *2018 International Journal of Computational Intelligence Systems* **11**(1): 692–705.

Kondo, F. and Watanabe, T. 2011. A tsp engine for performing tabu search, *2011 IEEE International Conference on Systems, Man, and Cybernetics* (2): 675–680.

Liu, X., Zhang, B. and Du, F. 2014. Integrating relative coordinates with simulated annealing to solve a traveling salesman problem, *2014 Seventh International Joint Conference on Computational Sciences and Optimization* (2): 177–180.

Luo, W., Lin, D. and Feng, X. 2016. An improved ant colony optimization and its application on tsp problem, *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)* **2**(40): 136–145.

Marappan, R. and Sethumadhavan, G. 2013. A new genetic algorithm for graph coloring, *2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation* (2): 49–54.

Marappan, R. and Sethumadhavan, G. 2016. Solution to graph coloring problem using divide and conquer based genetic method, *2016 International Conference on Information Communication and Embedded Systems (ICICES)* (1): 1–5.

Mohammad, H. R. and Lixin, T. 2018. Parallelizing combinatorial optimization heuristics with gpus, *2017 International Symposium on Computer Science and Intelligent Controls (ISCSIC)* **3**(2): 265–280.

Mudaliar, D. N. and Modi, N. K. 2013. Unraveling travelling salesman problem by genetic algorithm using m-crossover operator, *2013 International Conference on Signal Processing , Image Processing and Pattern Recognition* (1): 127–130.

Niño, E., Ardila, C., Perez, A. and Donoso, Y. 2010. A genetic algorithm for multiobjective hard scheduling optimization, *International Journal of Computers Communications & Control* **5**(5): 825–836.

Niño, E. D. 2012. Samods and sagamods: Novel algorithms based on the automata theory for the multi-objective optimization of combinatorial problems, *International Journal of Artificial Intelligence* **12**(8): 147–165.

Nino-Ruiz, E., Ardila, C. and Capacho, R. 2018. Local search methods for the solution of implicit inverse problems, *Soft Comput 22, Springer* (22): 4819–4832.

Niu, B. and Bi, Y. 2014. Binary bacterial foraging optimization for 0/1 knapsack problem, *2014 IEEE Congress on Evolutionary Computation (CEC)* (1): 647–652.

Ohlsson, M., Peterson, C. and Sderberg, B. 1993. Neural networks for optimization problems with inequality constraints: The knapsack problem, *Neural Computation* **5**(2): 331–339.

Precup, R. E., David, R.-C., Petriu, E. M., Szedlak-Stinean, A.-I. and Bojan-Dragos, C.-A. 2016. Grey wolf optimizer-based approach to the tuning of pi-fuzzy controllers with a reduced process parametric sensitivity, *4th IFAC Conference on Intelligent Control and Automation SciencesICONS 2016 Reims, 13 June 2016, IFAC-PapersOnLine,France* **49**(5): 55–60.

Precup, R. E. and Tomescu, M. 2015. Stable fuzzy logic control of a general class of chaotic systems, *Neural Computing and Applications* **26**(3): 541–550.

Ruiz-Rangel, J., Ardila Hernández, C., Jabba Molinares, D. and Maradei Gonzalez, L. 2018. Ernead: Training of artificial neural networks based on a genetic algorithm and finite automata theory, *International Journal of Artificial Intelligence* **16**(1): 214–253.

Scholz, J. 2019. Genetic algorithms and the traveling salesman problem a historical review, *Neural and Evolutionary Computing (cs.NE)* (1): 1–8.

Sharma, P. and Gupta, M. 2015. Tsp problem using modified abc based on dynamically division of bees, *2015 International Conference on Computing Communication Control and Automation* (1): 427–431.