Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 362 – 376

doi:10.3906/elk-1804-56

A novel metaheuristic optimization algorithm: the monarchy metaheuristic

Ibtissam AHMIA*, **Méziane AÏDER**
Laboratory of Operations Research and Mathematics of Decision, Faculty of Mathematics,
University of Sciences and Technology Houari Boumediene, Bab Ezzouar, Algeria

Received: 07.04.2018 • **Accepted/Published Online:** 04.10.2018 • **Final Version:** 22.01.2019

Abstract: In this paper, we introduce a novel metaheuristic optimization algorithm named the monarchy metaheuristic (MN). Our proposed metaheuristic was inspired by the monarchy government system. Unlike many other metaheuristics, it is easy to implement and does not need a lot of parameters. This makes it applicable to a wide range of optimization problems. To evaluate the efficiency of the proposed algorithm, we examined it on the traveling salesman problem (TSP) using some benchmark from TSPLIB online library of instances for the TSP. The experimental results indicate that the monarchy metaheuristic is competitive with the other methods that exist in the literature for finding approximate solutions.

Key words: Metaheuristics, the monarchy metaheuristic, hybrid-methods, combinatorial optimization

1. Introduction

Our proposed method, the monarchy metaheuristic, is a metaheuristic inspired by a form of government called monarchy. A monarchy is a form of government in which supreme power is absolutely or nominally lodged in an individual, who is the head of state, until his death or abdication. The head of a monarchy is called a monarch or king. It was a common form of government across the world during the ancient and medieval times. Nowadays, more than forty sovereign nations in the world have a monarch acting as head of state. Monarchies can be classified according to how their monarch is selected, for instance, an elective monarchy is a monarchy ruled by an elected monarch. On the other hand, for the hereditary monarchy, the throne is passed down as a family inheritance but elective monarchies can transform into hereditary ones over time and hereditary ones can have occasional elective aspects.

2. Definitions needed to understand the proposed method

2.1. Hereditary monarchy

A hereditary monarchy is the most common style of monarchy; it is the form that is used by almost all of the world's existing monarchies. Under a hereditary monarchy, all the monarchs come from the same family, and the throne is passed down from one family member to another. The hereditary system has the advantages of stability, continuity, and predictability.

*Correspondence: iahmia@usthb.dz

2.2. Dynasty

A dynasty is a family or line of rulers, a succession of sovereigns of a country belonging to a single family or tracing their descent to a common ancestor. The persons in line to become monarchs are called dynasts.

2.3. Order of succession

An order of succession or line of succession is the sequence of those people eligible to come to the throne. In a hereditary monarchy, the position of monarch is inherited according to an established order of succession, this provides immediate replacement of the monarch after an unexpected vacancy. All possible monarchs are legally recognized as born into or descended from the reigning dynasty or a previous one.

Different systems of succession have been used to select the next monarch such as, primogeniture, seniority, and tanistry. For primogeniture, the eldest child of the monarch is first in line to become monarch. In some monarchies, succession to the throne uses seniority which usually first gives the throne to the monarch's next eldest brother. Tanistry is a semielective and gives weight to merit while choosing the next monarch. A self-proclaimed monarchy is established when a person claims the monarchy without any historical ties to a previous dynasty.

3. The traveling salesman problem

We used the traveling salesman problem (TSP) for benchmarking and testing our method. It is commonly used in investigating the performance and behavior of metaheuristics. It is one of the most studied problems of combinatorial optimization. The TSP is very easy to describe, yet very difficult to solve. It is an NP-Hard problem; this means that a method that ensures an optimal solution for all instances of this problem within reasonable execution time does not exist yet.

The TSP can be described as follows: Given N cities and the distances between each pair of them, the task of the salesman is to visit each and every city once so that the overall tour-length is minimal. The solutions are coded using path representation which is the most natural representation which represents a tour as a list of N cities. If city j is k^{th} element of the list, city j is k^{th} city to be visited. Hence, the tour 2 0 1 3 4 is simply represented by (2, 0, 1, 3, 4).

4. Description of the monarchy metaheuristic

The monarchy metaheuristic inspired by the monarchy government system starts first by finding two solutions to the problem randomly or by using any known heuristic, one of these two solutions will be the king, the second will help to make the first dynasty list which contains the possible successors to the throne. For this king solution, we will create two lists: the children list L_1, and the relative list L_2. Hence, the dynasty list will be D: the union of L_1 and L_2.

The next king will be chosen using one of three possible ways of succession chosen among the set of different ways of succession that exist: primogeniture, seniority, and tanistry. We added a fourth way of selection named intruder which represents the situation of self-proclaimed monarchy, where the next king is not chosen from a dynasty list. The algorithm keeps running from one iteration to another, each iteration starts with a new king solution to which we associate a dynasty list, this list is updated at each iteration. The algorithm terminates when it reaches a prespecified number of iterations.

4.1. Finding the first king

The first king solution can be generated randomly or created using any known heuristic. For instance, for the TSP, the heuristic that we have used is a constructive heuristic based on the nearest neighbor algorithm, it is one of the first algorithms used to determine a solution to the TSP. It starts with a randomly chosen city, as long as there are cities that have not yet been visited, visits the nearest city that still has not appeared in the tour. The algorithm terminates once all cities have been visited. It quickly finds a tour, but usually not the best one.

4.2. Creating the L_1 list

Every solution in the L_1 list is obtained by applying crossover between the king solution and another solution (found by a constructive heuristic, or generated randomly). Many kinds of crossover methods exist but it is essential that we find and apply a crossover method that produces valid results for our problem. In our case, we used the ordered crossover since this crossover method produced valid solutions for the TSP.

The ordered crossover was proposed by Davis in 1985 [1], this operator builds the children tour by using a subtour of one of the parents and maintaining the order of the cities of the remaining parent.

For example, consider the following two parent tours:

$(2,0,1,3,4)$, $(0,1,3,4,2)$. Two cut points identical for both parents are chosen. Assuming that these cut points are located between the positions two and three and between three and four: $(2,0|1|3,4)$, $(0,1|3|4,2)$. The resulting children tours will have the subtour between the cut points the same as the parent tours as follows: $(**|1|**)$, $(**|3|**)$.

Then, using the other parent tour, starting with the second cut point and omitting the cities that have already been presented in the child tour, the remaining nodes tour are inserted in the same order they appear in it. When the end of the parent tour is reached, it continues from its first position. The children tours resulting from this example would be $(0,3|1|4,2)$ and $(0,1|3|4,2)$.

4.3. Creating the L_2 list

The L_2 list is formed by using the king solution neighbors; three types of neighborhood have been used: one-point neighborhood, two-point neighborhood, and two-opt neighborhood.

The neighborhood of a given solution is the set of feasible solutions that somehow are similar to the given solution and their objective function values are not very different.

1. **One-point neighborhood:** Given a tour, we randomly select a city and then relocate it to all other positions of the tour. For instance, having a tour $(0,1,2,3,4)$ and the selected city 2, the next new tours are $(2,0,1,3,4)$, $(0,2,1,3,4)$, $(0,1,3,2,4)$, $(0,1,3,4,2)$.

2. **Two-point neighborhood:** Given a tour, we randomly select a city and then swap the position of the selected city with another city. For instance, having a tour $(0,1,2,3,4)$ and the selected city 2, the next new tours are $(2,1,0,3,4)$, $(0,2,1,3,4)$, $(0,1,3,2,4)$,$(0,1,4,3,2)$.

3. **Two-opt neighborhood:** Each two-opt neighbor is constructed with the well known two-opt algorithm, which was first proposed by Croes in 1958 [2]. The two-opt move deletes two edges and thus breaks the tour into two paths and then reconnects those paths in the other possible way.

One-point neighborhood (↓ T: tour, K: integer, pos: integer,↑ Neighbors: a list of tours)

```
Input T: a tour, K: a number of neighbors, pos: integer.
Output Neighbors: a list of K neighbors.
Begin
city:=Tour.get(pos); j := 0; Neighbors:=Null;
while (j ≤ K) do
    if (j ≠ pos) then
        i := 0;
        while i ≤ (Tour.size() − 1) do
            if (i ≠ pos) then Neighbor.set(i)=Tour.get(i);
            end if
            i := i + 1;
        end while
        // The function add(pos, city) will add city at the position pos of the list Neighbor.
        Neighbor.add(pos,city);
    end if
    Neighbors:=Neighbors.add(Neighbor);
    j := j + 1;
end while
End
```

Two-point neighborhood (↓ T: tour, K: integer, pos: integer, ↑ Neighbors: a list of tours)

```
Input T: a tour, K: a number of neighbors, pos: integer.
Output Neighbors: a list of K neighbors.
Begin
j := 0; Neighbors:=Null;
while (j ≤ K) do
    Neighbor:=Tour.copy;
    Neighbor.switch(pos,j);
    Neighbors:=Neighbors.add(Neighbor)
    j := j + 1
end while
End
```

4.4. Selection of the next king

As mentioned before, four ways of selection were used.

1. **Primogeniture:** Primogeniture is a succession law where the eldest son (or daughter, under certain conditions) inherits all titles. In the absence of any children, brothers succeed. In the monarchy metaheuristic, if the way of selection of the next king is primogeniture, the next king will be chosen from the L_1 list.
2. **Seniority:** For the seniority way of succession, the order of succession to the throne prefers the king's younger brother over the king's own sons. A king's children (the next generation) succeed only after the males of the elder generation have all been exhausted. In the monarchy metaheuristic, if the way of selection of the next king is seniority, the choice of the next king will be from the L_2 list.
3. **Tanistry:** Tanistry is a rare succession method where the king is chosen from the dynasty list by election. In the monarchy metaheuristic, if the way of selection of the next king is tanistry, the next king will be the best solution in list D which is the dynasty list that contains all members of both L_1 and L_2 lists.

Two-opt neighborhood (↓ T: tour, pos1: integer, ↑ Neighbors: a list of tours)

Input T: a tour, pos1: integer.
Output Neighbors: a list of neighbors.
Begin
$pos2 := pos1 + 1$;
$i := 0$; Neighbors:=Null;
while $(i \leq Tour.size() - 3)$ **do**
  $pos3 := pos2 + 1 + i$;
  $pos4 := pos3 + 1$;
  //Reverse the path between pos2,pos3.
  Neighbor:=Neighbor.ReversePath(pos2, pos3);
  Neighbors:=Neighbors+Neighbor;
  $i := i + 1$;
end while
End

Procedure primogeniture (↓ L_1 : a list of tours, ↑ T : a tour)

Input ↓ L_1: a list of tours.
Output ↑ T : the best tour of L_1.
Begin
// BestFitness (↓ L_1: a list of tours) is a function that calculate the fitness value of each tour in L_1 and return the tour that have the best value of fitness.
T= BestFitness (L_1);
End

Procedure seniority (↓ L_2 : a list of tours, ↑ T : a tour)

Input ↓ L_2: a list of tours.
Output ↑ T : the best tour of L_2.
Begin
// BestFitness (↓ L_2: a list of tours) is a function that calculate the fitness value of each tour in L_2 and return the tour that have the best value of fitness.
T= BestFitness (L_2);
End

Procedure tanistry (↓ L_1, L_2 : a list of tours, ↑ T : a tour)

Input ↓ L_1, L_2 : two lists of tours.
Output ↑ T : the best tour of $L_1 \cup L_2$.
Begin
Tour D= $L_1 \cup L_2$;
// BestFitness (↓ D: a list of tours) is a function that calculates the fitness value of each tour in D and returns the tour that have the best value of fitness.
T= BestFitness (D);
End

4. **Intruder:** A self-proclaimed monarchy is established when a person claims the monarchy without any historical ties to a previous dynasty. In the monarchy metaheuristic, if the way of selection is intruder, the next king solution will not be chosen from the dynasty list. It will be created using any known metaheuristic. In our case, simulated annealing (SA) [3] was used to create an intruder king.

Procedure intruder ($\uparrow Solution$: a tour)

Output $\uparrow Solution$: a tour found using SA
Begin
// FirstSolution() is a function that find a first solution to start a simulated annealing algorithm.
Tour CurrentTour=FirstSolution();
// Initialize the simulated annealing parameters
$T_{Initial}$= Random(Int); T_{Final}= Random(Int);
T= $T_{initial}$;
I_{Max}= Random(Int);
while $i \leq I_{Max}$ **do**
  while $T \geq T_{Final}$ **do**
    // Pick a random neighbor of CurrentTour.
    $NextTour \in N(CurrentTour)$;
    $\Delta E = Fitness(NextTour) - Fitness(CurrentTour)$;
    if $\Delta E \leq 0$ **then**
      CurrentTour= NextTour;
    else if $\Delta E > 0$ **then**
      Set (CurrentTour=NextTour) with probability $\exp^{-\frac{\Delta}{T}}$
    end if
    $T = T \times CoolingRate$;
  end while
  $i = i + 1$;
end while
Solution= CurrentTour;
End

4.5. The monarchy metaheuristic framework

As any other metaheuristic, the monarchy metaheuristic can be defined within a generic framework. One of the principal advantages of this metaheuristic is that it can be viewed as single-solution metaheuristic if we decide to keep the king solution only at each iteration or it can be used as population-based metaheuristic if all solutions of the dynasty list associated to the king solution of each iteration are memorized. Algorithm 1 presents the framework of the proposed method.

4.6. The monarchy metaheuristic parameter

The monarchy metaheuristic (MN) needs one parameter: the length of the two lists, L_1 and L_2. Different ways can be used to set this parameter. The user can choose to set the length of these two lists at the beginning of the algorithm and then for every king solution at each iteration, the length of the two lists, L_1 and L_2, will be the same. Another way which gives more importance to the quality of the solution selected as a King at each iteration consists of choosing the length of the two lists according to the percentage of improvement of the solution. The length of the lists can be changed from one iteration to another, and the length of L_1 can also be different from the length of L_2.

4.7. What if the next king solution does not bring an improvement?

We tested two alternatives in case the next selected king solution did not bring an improvement: The first one consists of keeping this solution and the size of L_1; L_2 constructed for this solution will decrease according to the value of percentage of improvement. This alternative has been denoted as MN_1. The second one is to

Algorithm 1 Pseudocode of the monarchy metaheuristic

Initialization

- Construct the first king solution K_1 randomly or using any known heuristic.
- Initialize LK with K_1. LK is the list that contains all king solutions found at all iterations.
- Set an initial size for L_1 and L_2.
- Generate L_1, L_2 associated to K_1.
- Set $delta \leftarrow 0$ ($delta$ is the percentage of improvement of solution).
- Set a value for the number of maximum iterations ($MaxIterations$).
- Initialize the iteration counter $N \leftarrow 1$.

while ($N < MaxIterations$) **do**

a) Choose a way of selection of the next king solution.

if (way of selection is Primogeniture) **then**

- Evaluate each solution of L_1.
- Select the next king solution from L_1 (the best solution in L_1).

else if (way of selection is Seniority) **then**

- Evaluate each solution of L_2.
- Select the next King from L_2 (the best solution in L_2).

else if way of selection is Tanistry) **then**

- Build the dynasty list D which is $L_1 \cup L_2$.
- Evaluate each solution of D.
- Select the next king solution from D.

else if (way of selection is Intruder) **then**

- Generate the next king solution by any other metaheuristic.

end if

b) Calculate $delta$, the percentage of improvement of the solution.

if $delta < 0$ **then**

1. Use one of the following alternatives:

Alternative 1:

- Put the actual king solution into the LK list and generate L_1, L_2.
- The size of L_1, L_2 will decrease according to the value of $delta$.

Alternative 2:

- Take the best king solution found so far and use its L_1, L_2 lists.

2. $N \leftarrow N + 1$.

else if $delta > 0$ **then**

1. Put the actual king solution into LK.

2. The size of L_1, L_2 will increase according to the value of $delta$.

3. Generate L_1 and L_2.

4. $N \leftarrow N + 1$.

end if

end while

Termination criteria: The algorithm stops if it reaches a prespecified number of iterations.

Finalization: Report the best king solution obtained.

neglect this solution, get back to the best king solution found so far and use the dynasty list of this best solution to choose the next king solution, this alternative has been denoted as MN_2.

5. Experimental results and discussion

The MN metaheuristic was tested on several instances of euclidian symmetric TSP, the instances were collected from TSPLIB which is an online library of TSP. Most of the instances included in TSPLIB have already been solved to optimality and they have been used in many research studies. Each instance presents a specific number of cities (e.g., the data in instance $eil51$ includes 51 cities and instance $st70$ includes 70 cities and so on). The Monarchy Metaheuristic was operated throughout 500 iterations and each TSP problem was repeated 50 times using integer distances and 20 times using real distances. The MN algorithm was implemented using Java on 2.5-GHz Dell computer with 4 GB RAM running on windows 7. The parameter setting for the MN metaheuristic is the length of the two lists L_1 and L_2, for the first iteration the initial lengths of L_1 and L_2 were set to 50. The results of the other methods were directly taken from the literature when available. The results presented are the best, the average, the worst tour lengths, and the relative error values obtained according to the optimal values. The relative error RE indicates how far the obtained solution is from the optimal solution. The relative percent error value was calculated with the following equation:

$$RE(\%) = \frac{(LRT - LOT)}{LOT} \times 100 \tag{1}$$

LRT: length of the resulting tour, LOT: length of the optimal tour.

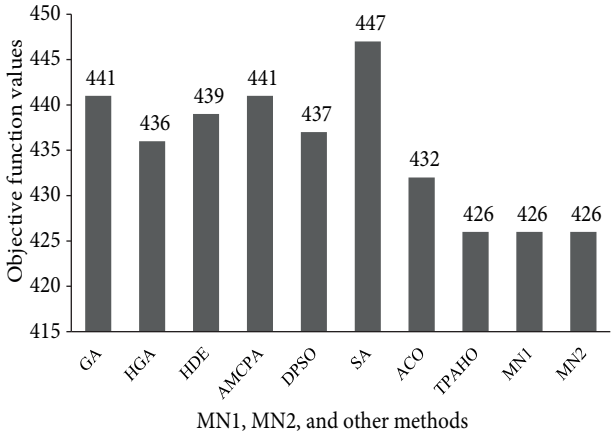


Figure 1. Comparison of objective function values for Eil51 instance of TSP (using integer distances).

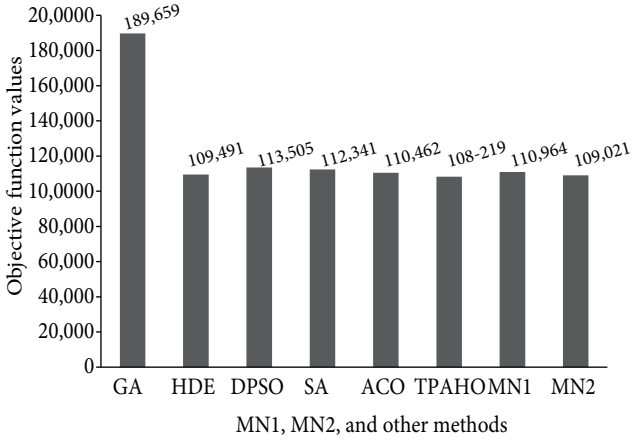


Figure 2. Comparison of objective function values for Pr76 instance of TSP (using integer distances).

Figures 1 and 2 show a comparison of objective function values of the proposed method with other existing metaheuristics that have used integer distances for Eil51 and Pr76 instances of TSP.

Table 1 shows a comparison of the two proposed alternatives of the monarchy metaheuristic MN_1 and MN_2 and other methods in the literature that have used integer distances, the results of GA (genetic algorithm) and AMCPA (adaptive multicrossover population algorithm) were taken from [4], the results of HGA (hybrid genetic algorithm) were taken from [5], the results of HDE (hybrid differential evolution algorithm) and DPSO (discrete particle swam optimization) were taken from [6], and the results of SA (simulated annealing), ACO

Table 1. The comparison of MN algorithm and other algorithms in the literature (using integer distances).

Problem	Method	Best	Average	Worst	RE	OPT
Eil51	GA	441	445	-	3.52	426
	HGA	436	440	-	2.35	
	AMCPA	441	465.7	-	3.52	
	HDE	439	444	-	3.05	
	DPSO	437	445	-	2.58	
	SA	447	467	-	4.92	
	ACO	432	439	-	1.40	
	TPASHO	426	431	-	0	
	MN_1	**426**	**432.88**	**442**	**0**	
	MN_2	**426**	**435.2**	**468**	**0**	
Berlin52	GA	7745	8040.1	-	2.69	7542
	AMCPA	7542	7805.2	-	0	
	HGA	7544	7559	-	0.03	
	HDE	7544	7816	-	0.02	
	DPSO	7700	7960	-	2.09	
	MN_1	**7863**	**8071.88**	**8183**	**4.25**	
	MN_2	**7703**	**7947.32**	**8295**	**2.13**	
St70	GA	707	750.2	-	4.74	675
	AMCPA	692	706.5	-	2.51	
	HGA	683	686	-	1.19	
	HDE	684	691	-	1.33	
	DPSO	712	733	-	5.48	
	MN_1	**705**	**721.32**	**735**	**4.44**	
	MN_2	**675**	**698.48**	**733**	**0**	
Eil76	GA	558	610.6	-	3.71	538
	AMCPA	566	578.1	-	5.20	
	HGA	552	559	-	2.60	
	HDE	558	568	-	3.71	
	DPSO	580	587	-	7.80	
	MN_1	**573**	**580.4**	**589**	**6.50**	
	MN_2	**540**	**558.54**	**585**	**0.37**	
Pr76	GA	189,659	-	-	75.35	108,159
	HDE	109,491	110,539	-	1.23	
	DPSO	113,505	115,144	-	4.94	
	SA	112,341	113,523	-	3.86	
	ACO	110,462	111,037	-	2.13	
	TPASHO	108,219	110,031	-	0.05	
	MN_1	**110,964**	**114,522.66**	**116,355**	**2.59**	
	MN_2	**109,021**	**111,116.08**	**115,384**	**0.79**	
Kroa100	GA	21,566	22,270.4	-	1.33	21,282
	AMCPA	21,608	22,125.3	-	1.53	
	HGA	21,733	21,811	-	2.12	
	MN_1	**23,178**	**23,625**	**23,844**	**8.90**	
	MN_2	**22,363**	**23,398.08**	**23,674**	**5.08**	
Eil101	GA	696	725.8	-	10.65	629
	AMCPA	657	678.1	-	4.45	
	HGA	651	661	-	3.50	
	MN_1	**654**	**667.38**	**674**	**3.97**	
	MN_2	**630**	**649.68**	**670**	**0.15**	

Table 2. The comparison of MN algorithm with other algorithms in the literature (using real distances).

Problem	Method	Best	Average	Worst	RE	OPT	CPU(s)
Eil51	ACO	450.59	457.86	463.55	5.06		112.11
	ABC	563.75	590.49	619.44	31.45		2.16
	HA	431.74	443.39	454.97	0.67	428.87	58.33
	MN_2	**431.17**	**437.25**	**446.89**	**0.54**		**71.1**
Berlin52	ACO	7548.99	7659.31	7681.75	0.06		116.67
	ABC	9479.11	10390.26	11021.99	25.64		2.17
	HA	7544.37	7544.37	7544.37	0	7544.37	60.64
	MN_2	**7774.24**	**7952.09**	**8065.97**	**3.05**		**69.25**
St70	ACO	696.05	709.16	725.26	2.79		226.06
	ABC	1162.12	1230.49	1339.24	71.63	677.11	3.15
	HA	687.24	700.58	716.52	1.49		115.65
	MN_2	**682.66**	**706.11**	**724.59**	**0.82**		**103.8**
Eil76	ACO	554.46	561.98	568.62	1.66		271.98
	ABC	877.28	931.44	971.36	60.85		3.49
	HA	551.07	557.98	565.51	1.04	545.39	138.82
	MN_2	**563.66**	**573.68**	**583.43**	**3.35**		**123.85**
Pr76	ACO	115166.66	116321.22	118227.41	6.48		272.41
	ABC	195198.9	205119.61	219173.64	80.47		3.50
	HA	113798.56	115072.29	116353.01	5.21	108159.44	138.92
	MN_2	**109140.82**	**111049.24**	**114868.83**	**0.91**		**125.9**
Kroa100	ACO	22455.89	22880.12	23365.46	5.49		615.06
	ABC	49519.51	53840.03	57566.05	132.64		5.17
	HA	22122.75	22435.31	23050.81	0.04	21285.44	311.12
	MN_2	**22415.47**	**23116.88**	**23703.12**	**5.31**		**204.65**
Eil101	ACO	678.04	693.42	705.65	5.56		527.42
	ABC	1237.31	1315.95	1392.64	92.63		5.17
	HA	672.71	683.39	696.04	4.73	642.31	267.08
	MN_2	**664.258**	**682.935**	**700.497**	**3.42**		**210.05**

(ant colony optimization), and TPASHO (two-phase hybrid optimization algorithm) were taken from [7]. As seen in Table 1, the results produced by the proposed method are better than those of all the other methods for Eil101, Eil76, Eil51, and St70 test problems, and the optimum solutions are obtained for the Eil51 and St70 instances. For the Pr76 instance, the quality of the obtained solution is better than those of GA, HDE, DPSO, SA, ACO, and the solution is not far from the optimum solution ($RE = 0.79$).

We noticed that the results of the MN_2 are better than those of MN_1. Therefore, we did more tests using real distances to compare MN_2 with the methods in the literature that used real distances.

Table 2 shows a comparison between the proposed method MN_2 and ACO (ant colony optimization), ABC (artificial bee colony), and HA (hierarchic approach: ACO with ABC); these three methods used real distances and their results were taken from [8].

In Table 2, we can see that the proposed method (MN) produced results that are better than those of ACO in terms of time and quality. The quality of the obtained solutions of the MN method are better than those of the ABC method and it is very competitive with the HA method.

Figures 3 and 4 show a comparison of objective function values of the proposed method with other existing metaheuristics that use integer distances for some TSP instances.

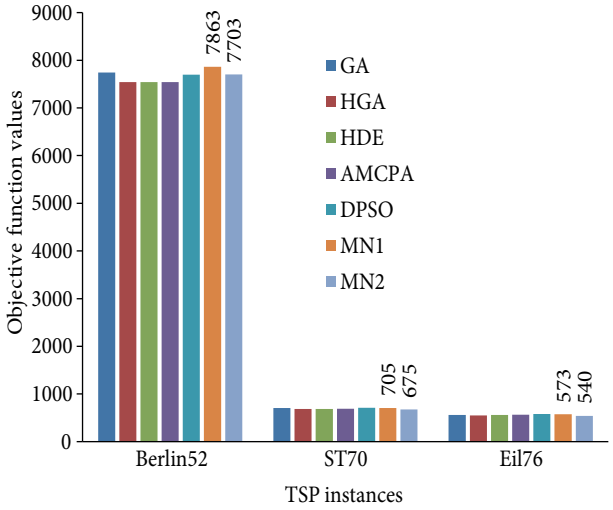


Figure 3. Comparison of objective function values for Berlin52, Eil76, and St70 instances of TSP (using integer distances).

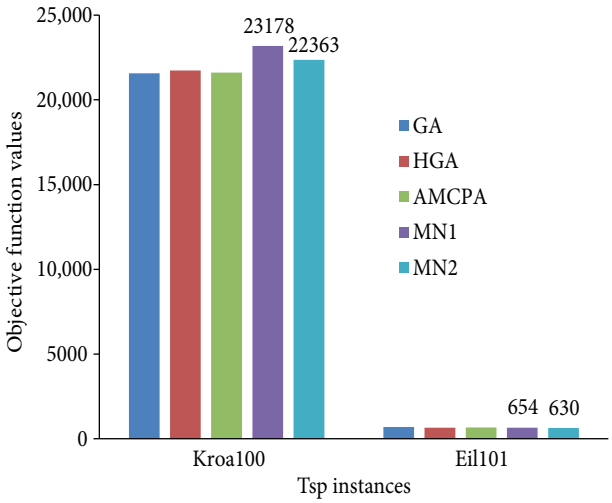


Figure 4. Comparison of objective function values for Kroa100 and Eil101 instances of TSP (using integer distances).

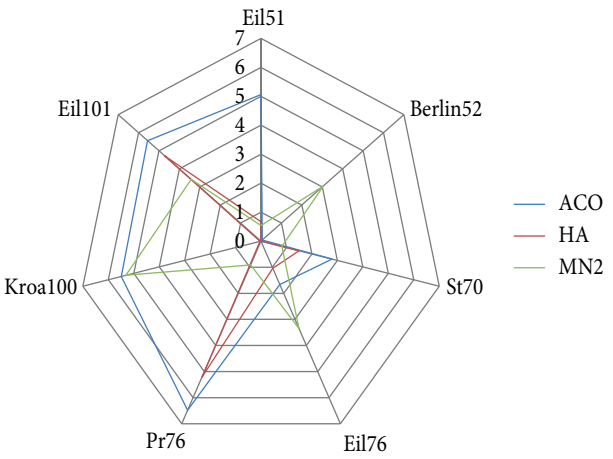


Figure 5. RE% comparison of MN2 with ACO and HA (using real distances).

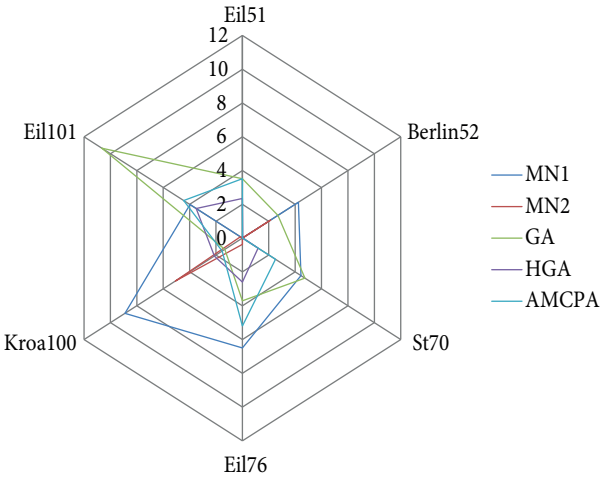


Figure 6. RE% comparison of MN1 and MN2 with ACO, ABC, HA (using integer distances).

Figure 5 shows a comparison of the relative percent error values of our metaheuristic with other existing metaheuritics that used real distances.

Figure 6 shows a comparison of the relative percent error values of our metaheuristic with other existing metaheuritics that used integer distances.

Figures 7 and 8 show a comparison of objective function values of the proposed method with other existing metaheuristics that used real distances. Figure 9 shows computing time comparison between the MN metaheuristic and other existing methods.

Figure 10 shows the convergence curve of best results for the proposed method (the second alternative MN_2) in single run (best run) for Eil51 TSP instance, in this case, the proposed algorithm started with solution found by greedy heuristic and reached the 426 value which is the optimal value for Eil51 instance. Figure 11

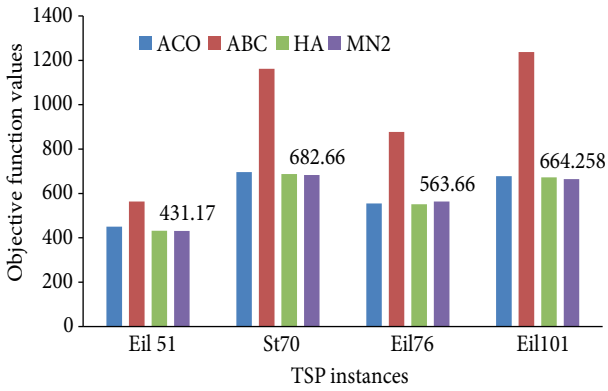


Figure 7. Comparison of objective function values for Eil51, St70, Eil76, and Eil101 instances of TSP (using real distances).

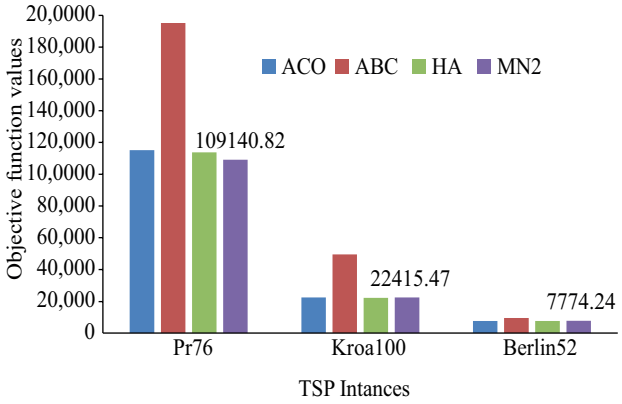


Figure 8. Comparison of objective function values for Pr76, Kroa100, and Berlin52 instances of TSP (using real distances).

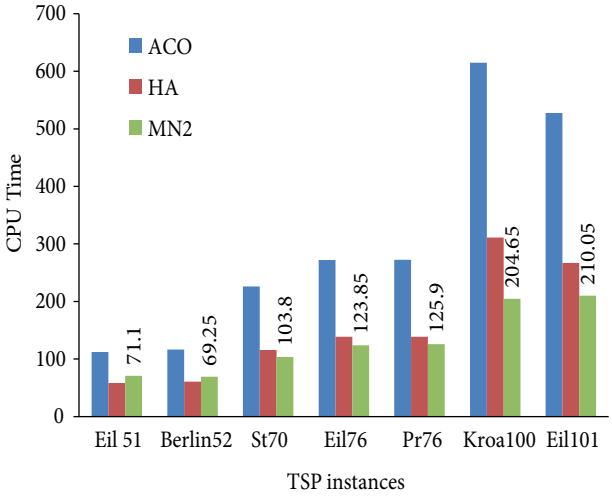


Figure 9. CPU comparison of the ACO and HA (ACO+ABC) with MN2 (using real distances).

shows the convergence curve of best results for the proposed method (the second alternative MN_2) in single run for Pr76 instance, the proposed algorithm started with solution found by greedy heuristic.

Figures 12a and 12b show convergence curves of the proposed method, and the genetic algorithm (respectively), for the Eil51 instance. Figures 13a and 13b show convergence curves of the proposed method, and the genetic algorithm, respectively, for the Pr76 instance.

Figures 12 and 13 show a comparison of convergence curves of best results for the proposed metaheuristic and genetic algorithm both started with random solutions for Eil51 instance and Pr76 instance (respectively), the proposed algorithm converges faster than the genetic algorithm and needs less iterations than the genetic algorithm to reach the global optimum.

As it can be seen in Figures 12 and 13, the proposed method (MN metaheuristic) outperforms the genetic algorithm in terms of convergence and quality of solution.

The results obtained for TSP instances prove that the proposed method (the Monarchy Metaheuristic) is competitive with the existing methods in the literature. It is worth mentioning that the MN also has the possibility of improving its performance since it can use various algorithms as subalgorithms.

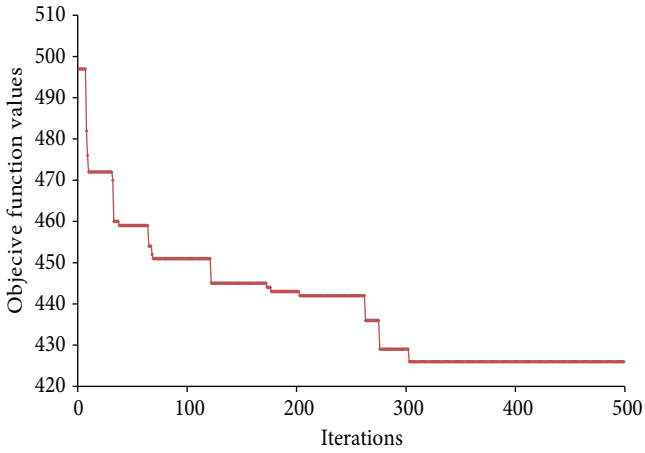


Figure 10. Convergence curve of best results for MN_2 starting with solution found by greedy heuristic for Eil51 instance.

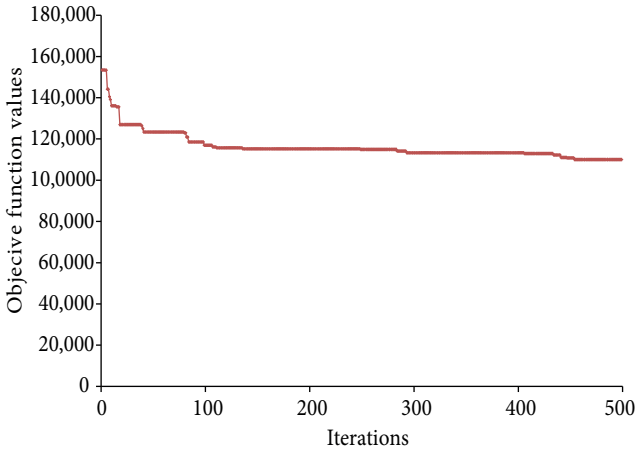
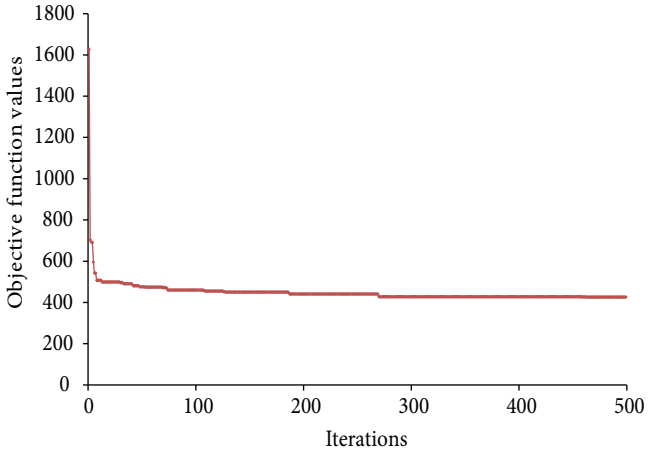
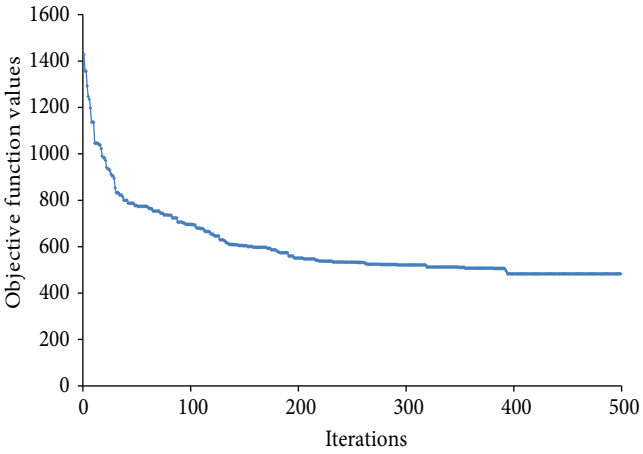


Figure 11. Convergence curve of best results for MN_2 starting with solution found by greedy heuristic for Pr76 instance.

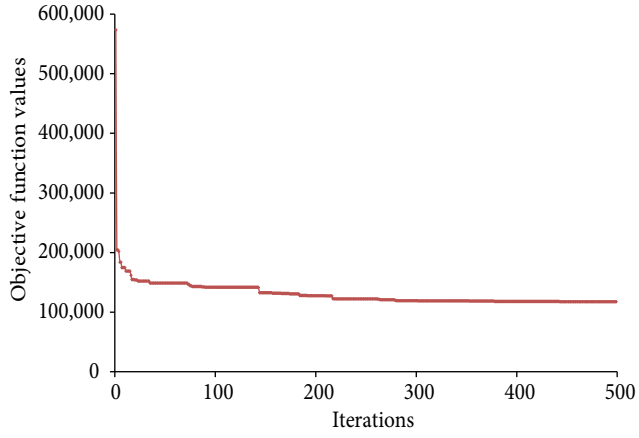


(a) Convergence curve of best results for MN2

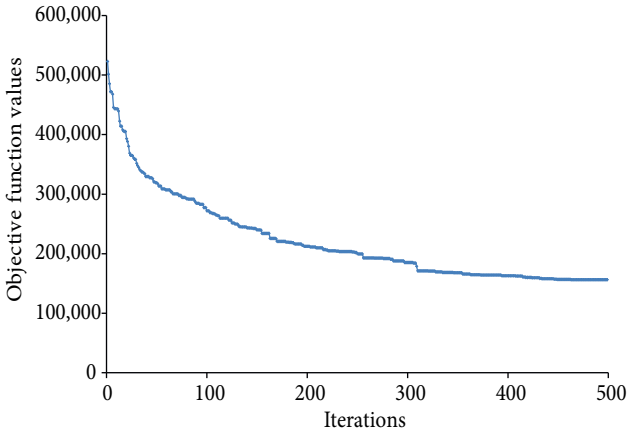


(b) Convergence curve of best results for GA

Figure 12. Comparison of convergence curves of MN_2 and GA both started with random solutions for Eil51 instance.



(a) Convergence curve of best results for MN2



(b) Convergence curve of best results for GA

Figure 13. Comparison of convergence curves between MN_2 and GA both started with random solutions for Pr76 instance.

6. Conclusion and future work

In this paper, a novel metaheuristic inspired by the monarchy government system and named the Monarchy Metaheuristic (MN) was introduced and investigated. The performance of the MN method was validated through a set of seven TSP (traveling salesman problem) benchmarks available on TSPLIB online library of instances of the TSP problem. Tests were done using integer distances and real distances. The results of the proposed method indicate that the MN method is competitive with the other methods that exist in the literature. In future works, we plan to reduce the computing time by changing the subalgorithms that were used and by testing others. We also aim to prove the effectiveness of the method on other types of problems such as the vehicle routing problem.

References

[1] Davis L. Applying adaptive algorithms to epistatic domains. In: IJCAI-85 the 9th International Joint Conference on Artificial Intelligence; 18–23 August 1985; California, USA. San Francisco, CA, USA: Morgan Kaufmann Publishers. pp. 162-164.

[2] Croes GA. A method for solving traveling salesman problems. Oper Res 1958; 6: 791-812.

[3] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983; 220: 671-680.

[4] Osaba E, Onieva E, Carballedo R, Diaz F, Perallos A. An adaptive multi-crossover population algorithm for solving routing problems. In: Terrazas G, Otero FB, Masegosa AD, editors. Nature Inspired Cooperative Strategies for Optimization (NICSO 2013). Cham, Switzerland: Springer, 2014. pp. 113-124.

[5] Lin B, Sun XY, Salous S. Solving traveling salesman problem with an improved hybrid genetic algorithm. J Comput Commun 2016; 4: 98-106.

[6] Wang X, Xu G. Hybrid differential evolution algorithm for traveling salesman problem. Procedia Eng 2011; 15: 2716-2720.

[7] Bao H. A two phase hybrid optimization algorithm for solving complex optimization problems. Int J Smart Home 2015; 9: 27-36.

[8] Gündüz M, Kiran MS, Özceylan E. A hierarchic approach based on swarm intelligence to solve the traveling salesman problem. Turk J Electr Eng Co 2015; 23: 103-117.

[9] Osaba E, Díaz F. Comparison of a memetic algorithm and a tabu search algorithm for the traveling salesman problem. In: Computer Science and Information Systems (FedCSIS) 2012 Federated Conference on; 9–12 September 2012; Wrocław, Poland. New Jersey, USA: IEEE. pp. 131-136.

[10] Gao S, Wang W, Dai H, Li F, Tang Z. Improved clonal selection algorithm combined with ant colony optimization. Ieice T Inf Syst 2008; 91: 1813-1823.

[11] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. Acm Comput Surv 2003; 35: 268-308.

[12] Jacobson SH, Henderson D, Johnson AW. The theory and practice of simulated annealing. In: Glover F, Kochenberger GA, editors. Handbook of Metaheuristics. Dordrecht, SH, Netherlands: Kluwer Academic Publishers: Springer, 2003. pp. 287-319.

[13] Johnson DS, Aragon CR, McGeoch LA, Schevon C. Optimization by simulated annealing: an experimental evaluation part 1: graph partitioning. Oper Res 1991; 39: 378-406.

[14] Blum C. Hybrid metaheuristics in combinatorial optimization: a tutorial. In: International Conference on Theory and Practice of Natural Computing; 2–4 October 2012; Tarragona, Spain. Berlin, Heidelberg, Germany: Springer. pp. 1-10.

[15] Blum C, Puchinger J, Raidl GR, Roli A. Hybrid metaheuristics in combinatorial optimization: A survey. Appl Soft Comput 2011; 11: 4135-4151.

[16] Ting O, Yang XS, Cheng S, Huang KZ. Hybrid metaheuristic algorithms: past, present, and future. In: Yang XS, editor. Recent Advances in Swarm Intelligence and Evolutionary Computation. Cham, Switzerland: Springer, 2015. pp. 71-83.

[17] Hanif H, Idris I. Tree physiology optimization in constrained optimization problem. Telkomnika 2018; 16: 876-882.

[18] Potvin JY. Genetic algorithms for the traveling salesman problem. Ann Oper Res 1996; 63: 337-370.

[19] Blum C, Roli A. Hybrid metaheuristics: an introduction. In: Blum C, Aguilera MJB, Roli A, Sampels M, editors. Hybrid Metaheuristics. Berlin, Heidelberg, Germany: Springer, 2008. pp. 1-30.

[20] Applegate LD, Bixby RE, Chvatal V, Cook WJ. The traveling salesman problem. Princeton, NJ, USA: Princeton University Press, 2006.

[21] Siarry P. Metaheuristics. Cham, Switzerland: Springer, 2016.

[22] Goldberg DE. Genetic algorithms in search optimization and machine learning. Boston, MA, USA: Addison-Wesley Publishing Company, 1989.

[23] Glover FW, Kochenberger GA. Handbook of Metaheuristics. Dordrecht, SH, Netherlands: Kluwer Academic Publishers, 2003.

[24] Siarry P, Idoumghar L, Lepagnot J. Swarm Intelligence Based Optimization. Cham, Switzerland: Springer, 2016.

[25] Sivanandam SN, Deepa SN. Introduction to Genetic Algorithms. Berlin, Heidelberg, Germany: Springer, 2008.

[26] Jacobson L, Kanber B. Genetic Algorithms in Java Basics. New York, NY, USA: Apress, 2015.

[27] Ghanea-Hercock R. Applied Evolutionary Algorithms in Java. New York, NY, USA: Springer, 2013.

[28] Krzanowski RM, Raper J. Spatial Evolutionary Modeling. New York, NY, USA: Oxford University Press, 2001.

[29] Blum C, Raidl GR. Hybrid Metaheuristics Powerful Tools for Optimization. Cham, Switzerland: Springer, 2016.

[30] Gendreau M, Potvin JY. Handbook of Metaheuristics. New York, NY, USA: Springer, 2010.

[31] Siarry P, Michalewicz Z. Advances in Metaheuristics for Hard Optimization. Berlin, Heidelberg, Germany: Springer, 2008.

[32] Edelkamp S, Schrodl S. Heuristic Search Theory and Applications. Waltham, MA, USA : Elsevier, 2011.

[33] Wang SC. Interdisciplinary Computing in Java Programming. New York, NY, USA: Springer Science and Business Media, 2012.

[34] Hoos HH, Stützle T. Stochastic Local Search: Foundations and Applications. San Francisco, CA, USA: Elsevier, 2004.