УДК 519.6, 519.8

Ткаченко Дмитро Анатолійович
студент
Національний технічний університет України
«Київський політехнічний інститут»

Ткаченко Дмитрий Анатольевич
студент
Национальный технический университет Украины
«Киевский политехнический институт»

Tkachenko D. A.
student
National Technical University of Ukraine
"Kyiv Polytechnic Institute"

ПОРІВНЯЛЬНЕ ДОСЛІДЖЕННЯ ДЕЯКИХ МЕТАЕВРИСТИЧНИХ АЛГОРИТМІВ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА

СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ НЕКОТОРЫХ МЕТАЭВРИСТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЕРА

A COMPARATIVE STUDY OF SOME METAHEURISTIC ALGORITHMS FOR SOLVING TRAVELLING SALESMAN PROBLEM

Анотація. *Досліджено застосування трьох метаевристичних алгоритмів для розв'язання задачі комівояжера: імітації відпалу, табу пошуку та мурашиної колонії. Проведено експериментальні дослідження продуктивності програмної реалізації алгоритмів на чотирьох тестових задачах комівояжера з відомими оптимальними маршрутами. В результаті експерименту оцінені характеристики розв'язків, отриманих кожним алгоритмом в серії вимірів за однакову фіксовану кількість часу. Детально описано застосовані в експерименті параметри алгоритмів і особливості реалізації. Візуалізовано процес оптимізації кожним із алгоритмів. Зроблено висновки щодо ефективності досліджених алгоритмів для різних розмірів задач, використаних у експерименті.*

Аннотация. *Исследовано применение трех метаэвристических алгоритмов для решения задачи коммивояжера: имитации отжига, табу поиска и муравьиной колонии. Проведены экспериментальные исследования производительности программной реализации алгоритмов на четырех тестовых задачах коммивояжера с известными оптимальными маршрутами. В результате эксперимента оценены характеристики решений, найденных каждым из алгоритмов в серии измерений за одинаковое фиксированное количество времени. Детально описаны использованные в эксперименте параметры алгоритмов и особенности реализации. Визуализирован процесс оптимизации каждым из алгоритмов. Сделаны выводы об эффективности исследованных алгоритмов для разных размеров задач, использованных в эксперименте.*

Summary. *Application of the following three metaheuristic algorithms to Travelling Salesman Problem (TSP) were explored: Simulated Annealing (SA), Tabu Search (TS), and Ant Colony System (ACS). The performance of software implementation of these approaches was experimentally studied using four test instances of TSP with known optimal solutions. As a result of the experiment, features of solutions found in a set of trials by each algorithm in the same fixed amount of time were assessed. The implementation details of the algorithms and the parameters used for the experiment were thoroughly described. Optimization process was visualized for every algorithm. Conclusions were made regarding the effectiveness of studied algorithms for the different sizes of problem instances used in the experiment.*

Ключові слова: *задача комівояжера, метаевристичні алгоритми оптимізації, алгоритм імітації відпалу, алгоритм табу пошуку, алгоритм мурашиної колонії.*

Ключевые слова: *задача коммивояжера, метаэвристические алгоритмы оптимизации, алгоритм имитации отжига, алгоритм табу поиска, алгоритм муравьиной колонии.*

Key words: *travelling salesman problem, metaheuristic optimization algorithms, simulated annealing algorithm, tabu search algorithm, ant colony system algorithm.*

INTRODUCTION

Travelling Salesman Problem (TSP) [1] has many applications in different fields. In its classical formulation, this problem appears in logistics, planning, and microchip manufacturing. Also this NP-hard [2] problem is a very popular testbed for various combinatorial optimization algorithms. The high computational complexity of seeking an optimal tour in TSP has facilitated the development of many heuristic and metaheuristic algorithms. Such algorithms are widely used because they can find a solution that is very close to the optimal solution, even for problems with very large numbers of cities [3].

The purpose of this paper is to explore the application of the following three metaheuristic algorithms to the Travelling Salesman Problem (TSP): Simulated Annealing (SA), Tabu Search (TS) and Ant Colony System (ACS). Another objective is to experimentally determine the algorithm that is able to produce the best solution when given some constant amount of computational resources.

PROBLEM FORMULATION

TSP is the problem of finding the shortest closed tour through a given set of cities, when distances between them are known, in such a way that each city is visited exactly once and the tour ends at the start city. In other words, the task is to find a Hamiltonian cycle with the least weight, given a complete weighted graph.

SIMULATED ANNEALING ALGORITHM

Description. Simulated Annealing algorithm [4], [5] consists of the following steps:

1. Start with a random tour through the given set of cities.

2. Create a neighbour solution.

3. If the candidate tour is shorter than the current tour, accept it. Otherwise, still accept it, according to some probability.

4. Go back to Step 2 and repeat until the stopping criteria are met, lowering the temperature at each iteration according to some cooling schedule.

This algorithm is inspired by annealing in metallurgy. Annealing is the process of heating and controlled cooling of a system.

The key to this algorithm is in Step 3. The higher the system temperature, greater is the probability of accepting the worse candidate tour. This probability decreases as the system cools down. The point of accepting the longer tour is to try to avoid getting stuck in a local minimum.

Pseudocode:

```
Input: numberOfCities, time_max, T_0
Output: S_best

S_current = createRandomTour(numberOfCities)
S_best = S_current
time_start = currentTime()
i = 0;
while currentTime() - time_start < time_max
      S_i = generateNeighborSolution(S_current)
      T_current = calculateTemperature(i, T_0)
      if cost(S_i) <= cost(S_current)
            S_current = S_i
            if cost(S_i) <= cost(S_best)
                  S_best = S_i
            end
      else if exp((cost(S_current) - cost(S_i)) / T_current) > rand(0, 1)
            S_current = S_i
      end
      ++i
end
return S_best
```

Parameters: Initial temperature (T_0) and cooling schedule.

Implementation details. The following cooling schedule is used:

$$T_n = T_0 \alpha^n.$$

Neighbour solution is created using the *Stochastic 2-opt* algorithm [6], which picks two random indices c_1 and c_2 in such a way that $c_2 > c_1$ and $c_2 - c_1 > 1$, and then reverses the city order in the tour in $[c_1; c_2]$ interval.

TABU SEARCH ALGORITHM

Description. This algorithm [7]–[9] can be described as a sequence of the following steps:

1. Generate a random tour.

2. Create a set of candidate tours that do not have any of the features on the tabu list.

3. Pick the shortest tour.

4. Add features of a chosen tour to tabu list. Trim the list so that it will contain no more than the specified

number of entries. Go back to Step 2, and repeat until the stopping criteria are met.

The key to this approach is to prevent the search procedure from repeatedly checking the same areas of a search space.

Pseudocode:

```
Input: numberOfCities, time_max, tabuListSize_max (N_T), numberOfCandidates (N_C)
Output: S_best

S_current = createRandomTour(numberOfCities)
S_best = S_current
tabuList = createEmptyList(tabuListSize_max)
time_start = currentTime()
while currentTime() - time_start < time_max
      candidateList = []
      for j = 1 to numberOfCandidates
            candidate = generateNeighborSolution(S_current, tabuList);
            candidateList.add(candidate)
      end
      bestCandidate = findBestCandidate(candidateList)
      if cost(bestCandidate) < cost(S_current)
            S_current = bestCandidate
            if cost(bestCandidate) < cost(S_best)
                  S_best = bestCandidate
            end
            tabuList.add(features(bestCandidate))
            while size(tabuList) > tabuListSize_max
                  removeLastEntry(tabuList)
            end
      end
end
return S_best
```

Parameters. Maximum size of tabu list (N_T) and the number of candidates generated in Step 2 (N_C).

Implementation details. Candidate tours in Step 2 are generated using *Stochastic 2-opt* algorithm, as before.

Tabu features are the edges between the cities *i* and *j*. On Step 4, edges (from the tour which is modified by *Stochastic 2-opt* procedure) between cities $c_1 - 1$ and c_1, and between $c_2 - 1$ and c_2 are added to the tabu list.

ANT COLONY SYSTEM ALGORITHM

Description. This algorithm [10], [11] is initialised in the same way as before. Then the following actions are performed during each iteration:

1. For each ant:

1.1. Perform a stepwise construction of a candidate tour.

1.2. Update the pheromone trail of the candidate tour.

1.3. Compare the length of the candidate tour with the best solution. Update the best solution if a shorter tour is found.

2. Update the pheromone trail of the best tour.

The key to this algorithm is to exploit both history (information about explored tours) and heuristic information (coefficient inversely proportional to edge length).

Pseudocode:

```
Input: numberOfCities, time_max, numberOfAnts (N_a), decayFactor (ρ),
heuristicCoef (β), historyCoef (α), localCoef (σ), greedinessFactor (q_0)
Output: S_best

S_best = createRandomTour(numberOfCities)
initialPheromone = 1.0 / (numberOfCities * cost(S_best))
pheromone = initialisePheromoneMatrix(initialPheromone)
time_start = currentTime()
while currentTime() - time_start < time_max
      for i = 1 to numberOfAnts
            candidate = constructStepwise(pheromone, numberOfCities,
            historyCoef, heuristicCoef, greedinessFactor)
            if cost(candidate) < cost(S_best)
                  S_best = candidate
            end
            localUpdatePheromone(pheromone, candidate, localCoef,
            initialPheromone)
      end
      globalUpdatePheromone(pheromone, S_best, decayFactor)
end
return S_best
```

Parameters:

Number of ants (N_α)

History coefficient (α)

Heuristic coefficient (β)

Local pheromone coefficient (σ)

Decay factor (ρ)

Greediness factor (q_0)

Implementation details. The greediness factor determines when to use the probabilistic *Roulette Wheel Selection* method and when to greedily choose the tour with highest $\tau_{i,j}^{\alpha} \times \eta_{i,i}^{\beta}$ value ($\tau_{i,i}$ represents the pheromone for the edge (i,j); $\eta_{i,j} = \frac{1}{d(i,j)}$, where $d(i,j)$ is the distance between cities *i* and *j*). When *Roulette Wheel Selection* is used, the probability of picking a candidate is proportionate to

$$P_{i,j} \leftarrow \frac{\tau_{i,j}^{\alpha} \times \eta_{i,j}^{\beta}}{\sum_{k=1}^{c} \tau_{i,k}^{\alpha} \times \eta_{i,k}^{\beta}},$$

where *c* is the number of cities.

A local pheromone trail update (for each ant) is performed according to the following expression:

$$\tau_{i,j} \leftarrow (1-\sigma) \times \tau_{i,j} + \sigma \times \tau_{i,j}^{0},$$

where $\tau_{i,j}^{0}$ is the initial pheromone value.

At the end of each iteration, pheromone trail is updated and decayed according to the best solution found so far, as follows:

$$\tau_{i,j} \leftarrow (1-\rho) \times \tau_{i,j} + \rho \times \frac{1}{C(S)},$$

where $C(S)$ is the length of the best tour.

EXPERIMENT

The performances of the described algorithms were compared using four TSP instances with the known optimal solutions from TSPLIB, accessible at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. On the specific problem instance, each algorithm was run for the

same fixed amount of time. 50 trials were performed for each problem and algorithm.

Results are presented in Table 1.

Table 1

Experiment results

				TSP instance			
				eil51	eil101	kroB150	kroA200
	optimal tour length			426	629	26130	29368
	number of cities			51	101	150	200
	time limit, s			1	5	10	15
Algorithm	SA	tour length	avg	444,26 (+4,29%)	674,38 (+7,21%)	27976,58 (+7,07%)	32094,96 (+9,29%)
			min	430 (+0,94%)	650 (+3,34%)	26972 (+3,22%)	30381 (+3,45%)
			max	463 (+8,69%)	704 (+11,92%)	29409 (+12,55%)	33722 (+14,83%)
		average number of iterations		89475,26	268549,18	352318,3	408675,26
	TS	tour length	avg	457,02 (+7,28%)	694,64 (+10,44%)	29667,76 (+13,54%)	35990,9 (+22,55%)
			min	439 (+3,05%)	671 (+6,68%)	28201 (+7,93%)	33969 (+15,67%)
			max	492 (+15,49%)	730 (+16,06%)	31209 (+19,44%)	37618 (+28,09%)
		average number of iterations		267,06	707,94	949,28	1062,8
	ACS	tour length	avg	450,12 (+5,66%)	698,5 (+11,05%)	29486,66 (+12,85%)	34428,7 (+17,23%)
			min	430 (+0,94%)	675 (+7,31%)	27627 (+5,73%)	31807 (+8,3%)
			max	470 (+10,33%)	732 (+16,38%)	31136 (+19,16%)	37812 (+28,75%)
		average number of iterations		46,2	28,22	32,78	40,56

Parameters used by each algorithm for the certain problem are given in Table 2.

Table 2

Parameters of the algorithms

Algorithm	**Parameter**	**TSP instance**			
		eil51	**eil101**	**kroB150**	**kroA200**
SA	T_0	100000	1000000		
	α	0,99975	0,9999	0,99995	
TS	N_T	15			
	N_C	50			
ACS	N_α	10	20	15	10
	α	1,0			
	β	2,5			
	σ	0,1			
	ρ	0,1			
	q_0	0,9			

The parameters of the algorithms were chosen according to [4], [5], [7]–[11]. The number of ants in ACS were adjusted in such a way that enough iterations (>20) were performed on each trial to show the effect of exploiting search history. In SA, initial temperature T_0 and α coefficient were tuned so that on each trial, the system completely cooled down during ~75% of iterations.

The optimization process for *eil51* TSP instance was visualized. Figure 1 shows the relationship between Δ and the iteration number. Δ is the absolute difference between the tour length on the current iteration of SA algorithm and the optimal tour length for this problem.

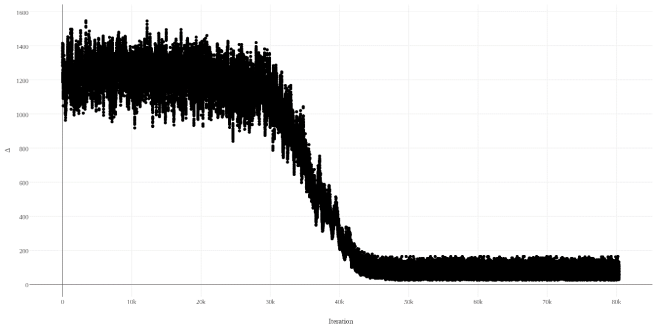

Fig. 1. Simulated Annealing optimization process (eil51)

For TS, the relationship between Δ and the iteration number is shown in Figure 2. Here, Δ is the difference between the best candidate tour length (there are N_C candidates on each iteration) on the current iteration and the optimal tour length for this problem.

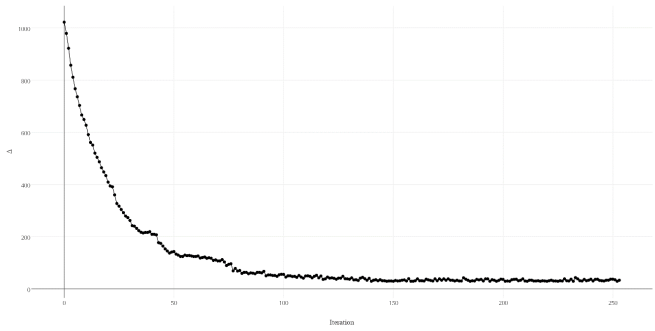

Fig. 2. Tabu Search optimization process (eil51)

Figure 3 shows the relationship between Δ and the iteration number for ACS. Δ is the difference between the best (of N_α ants) tour length on the current iteration and the optimal one.

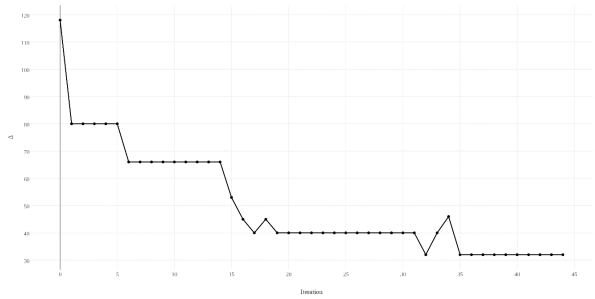

Fig. 3. Ant Colony System optimization process (eil51)

Analysis of results. Table 1 shows that all the described approaches were able to obtain tours that are close to the optimal one. For instances with 51 and 101 cities, found tours are 5–10% longer than the optimal tour (Fig. 4, 5, 6). Tours found for problems with 150 and 200 cities are 10–25% longer than the optimal tour. Experiment results also show that Simulated Annealing produced the best results on all the TSP instances. Ant Colony System algorithm found better tours than Tabu Search, in 3 out of 4 problem instances. The quality of Tabu Search solutions significantly degraded with increase in the number of cities.

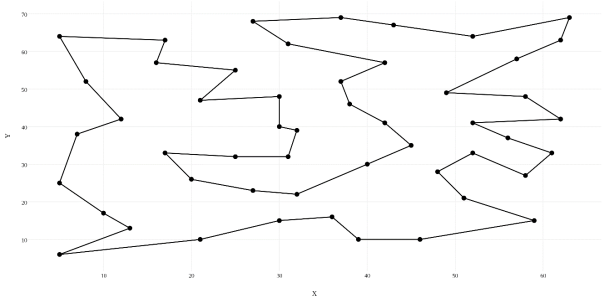

Fig. 4. Optimal tour in eil51

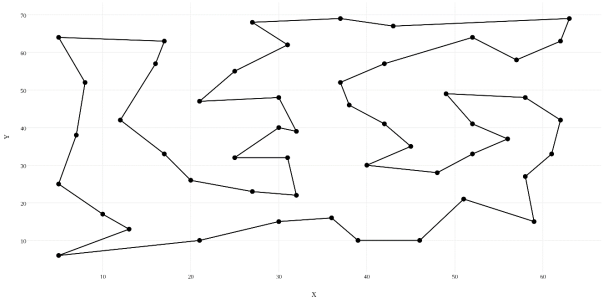

Fig. 5. Tour in eil51, which is 5.16% longer than the optimal tour

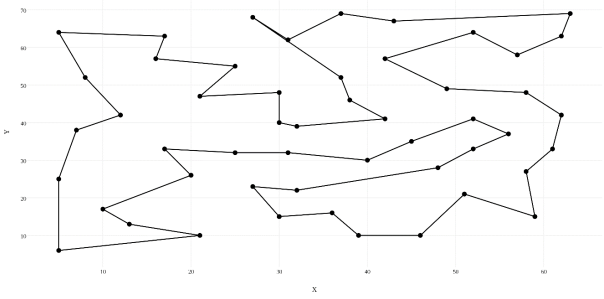

Fig. 6. Tour in eil51, which is 9.86% longer than the optimal tour

CONCLUSIONS

In this paper, Simulated Annealing, Tabu Search and Ant Colony System algorithms were described in the context of solving the Travelling Salesman Problem.

The performances of these approaches on four TSP instances with known optimal solutions were experimentally compared. Particularly, the length of a tour obtained by each algorithm in the same amount of time was compared.

This experimental study showed that the Simulated Annealing algorithm produced the best solutions to all the problem instances. The solutions obtained by the Ant Colony System were slightly worse. Tours found by Tabu Search were the longest. With the increase of number of cities, the absolute difference between the length of the found tour and the optimal one grew considerably slower for Simulated Annealing, slightly faster for Ant Colony System, and the fastest for Tabu Search.

These results give grounds to believe that Tabu Search will scale worse to larger problem instances than Simulated Annealing or Ant Colony System. It should also be noted that although Simulated Annealing performed better than the other algorithms, adjusting its parameters to the specific problem size and time limit is more sophisticated than for other approaches.

References

1. D. L. Applegate *et al*., *The Traveling Salesman Problem*. Princeton, NJ: Princeton University Press, 2007.

2. M. M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman & Co., 1979.

3. C. Rego *et al*., "Traveling salesman problem heuristics: leading methods, implementations and latest advances," *European Journal of Operational Research*, vol. 211, no. 3, pp. 427–441, June 2011.

4. S. Kirkpatrick *et al*., "Optimization by Simulated Annealing," *Science* vol. 220, no. 4598, pp. 671–680, May 1983.

5. V. Granville *et al*., "Simulated annealing: A proof of convergence," *IEEE Trans. Pattern Anal. Mach. Intell*., vol. 16, no. 6, p6. G. A. Croes. "A method for solving traveling salesman problems," *Operations Res*, vol. 6, no. 6, pp. 791–812, 1958.

7. F. Glover, "Tabu Search: A Tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

8. F. Glover and M. Laguna, *Tabu Search*. Norwell, MA: Kluwer Academic Publishers, 1997.

9. H. Osman and J. P. Kelly, *Meta-Heuristics: Theory and Applications*. Norwell, MA: Kluwer Academic Publishers, 1996.

10. M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

11. M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Trans. Evol. Comput*., vol. 1, no. 1, pp. 53–66, Apr. 1997.