A Dynamic Scatter Search Algorithm for Solving Traveling Salesman Problem

Aymen Jalil Abdulelah, Khalid Shaker, Ali Makki Sagheer and Hamid A. Jalab

Abstract Scatter Search (SS) is a population-based evolutionary metaheuristic algorithm that selects solutions from a specific memory called a reference set (*RefSet*) to produce other diverse solutions. In this work, a dynamic SS algorithm is proposed to solve the symmetric traveling salesman problem (TSP). To improve the performance of SS, a dynamic *RefSet* update and a dynamic population update are proposed. To test the performance of the proposed algorithm, computational experiments are carried out on the basis of the benchmark instances of the problem. The computational results show that the performance of the proposed algorithm is effective in solving the TSP.

Keywords Dynamic scatter search · Traveling salesman problem · Combinatorial optimization · Metaheuristic algorithm

1 Introduction

Scatter search (SS) is a population-based metaheuristic (P-metaheuristic) algorithm. SS was first introduced in 1977 by Glover [1] as a heuristic for integer programming. Originally, solutions are intentionally generated to assess characteristics in

A.J. Abdulelah · K. Shaker · A.M. Sagheer
College of Computer Science and Information Technology,
University of Anbar, Ramadi, Iraq
e-mail: ayman.ja90@gmail.com

K. Shaker
e-mail: khalidalhity@gmail.com

A.M. Sagheer
e-mail: ali_makki@computer-college.org

H.A. Jalab (✉)
Multimedia Unit, Faculty of Computer Science and Information Technology,
Universiti Malaya, 50603 Kuala Lumpur, Malaysia
e-mail: hamidjalab@um.edu.my; dr_hamidjalab@yahoo.com

H. Ibrahim et al. (eds.), *9th International Conference on Robotic, Vision, Signal Processing and Power Applications*, Lecture Notes in Electrical Engineering 398,
DOI 10.1007/978-981-10-1721-6_13

various areas of the solution space. SS orients its explorations systematically relative to a set of reference points that typically consist of good solutions previously obtained by problem-solving efforts. The criteria for "good" in this case are not restricted to objective function values, and may apply to sub collections of solutions rather than to a single solution, as in the case of solutions that differ from each other according to certain specifications. SS tries to avoid using many random stages; therefore, it does not use characteristic evolutionary algorithm operators, such as mutation and crossover. It is uses a small population, known as a reference set (*RefSet*), whose solutions are combined to construct new solutions through a systematic technique [2].

SS explicitly uses different approaches for both search intensification and search diversification. It incorporates search components from P-metaheuristics and single solution-based metaheuristics (S-metaheuristics) [3].

The traveling salesman problem (TSP) is a routing problem and it is classified as NP-hard combinatorial problem (NP-hard is a class of problems that are exponential time to be solved in optimality, "at least as hard as the hardest problems in NP") [3]. The TSP searches for the shortest route to visit a collection of cities and return to the starting point. Although the statement of the problem is very simple, it is a well-known NP-hard combinatorial optimization problem solved in polynomial time. Currently, the TSP has been comprehensively studied and solved using various meta-heuristic approaches, such as tabu search, neural networks, evolutionary algorithms, bees algorithm, and ant colony system.

Several previous studies have adopted SS algorithms in solving the TSP. Liu [4] introduced a hybrid SS for the probabilistic TSP (PTSP). In the hybrid SS, the nearest neighbor rule, edge recombination crossover operator, and threshold accepting are incorporated into the SS framework. The algorithm provides high-quality solutions for efficiently solving the PTSP. Pantrigo et al. [5] proposed the SS particle filter algorithm to solve the dynamic TSP. This work focused on reducing execution time without sacrificing the quality of the estimated solution. The authors effectively applied SS to a challenging dynamic TSP. Sagheer et al. [6] proposed an improvement of SS using a bees algorithm and achieved solution diversity. The results of the study suggested that the use of SS algorithms may provide solutions that are inferior to those of some other metaheuristic algorithms.

SS has also been used to solve other routing problems. Russell and Chiang [7] utilized an SS framework to solve a vehicle routing problem with time windows. They proved that an SS framework can be used to generate quality solutions and that SS is competitive compared with the best existing context-dependent metaheuristics. Tang et al. [8] developed an efficient SS procedure to solve a vehicle routing problem. Improved versions of the nearest neighbor heuristic and arc combination were respectively utilized in the improvement stage and in the combination stage.

2 Basic Scatter Search Design

SS methodology is very flexible because its elements can be implemented in various ways and degrees of refinement. In this section, a basic SS design is presented, and an improved design is explained. The implementation of SS is composed of five stages. So, the improvements of the algorithm depend on how these five stages are implemented.

The mechanisms of SS are not limited to a single uniform design; thus, different strategic possibilities may be explored to prove the effectiveness of SS in a specific implementation. These principles are the bases of our improvement of SS.

Diversification Generation Stage. In this stage, a population (*Pop*) of the diverse trial solutions is generated. The size of the population is related to the size of *RefSet*.

Improvement Stage. In this stage, a trial solution from the population is transformed into one or more improved solutions. If the input trial solution occurs is not improved, the "improved" solution is considered to be the same as the input solution. SS uses a local search algorithm for solution enhancement.

***RefSet* Update Stage**. In this stage, a *RefSet* is built and updated. The said *RefSet* consists of the best solutions found. These are so organized to ensure that other parts of the stage can efficiently access the solution. Solutions gain membership to *RefSet* according to their quality or their diversity. Thus, the objective of building the *RefSet* is to ensure both the diversity the high-quality of solutions and to overcome the local optimum.

Subset Generation Stage. In this stage, *RefSet* is controlled, and subset groups of solutions are produced. These solutions are used as bases for generating combined solutions.

A Solution Combination Stage. In this stage, the given subset of solutions generated by the previous stage is transformed into one or more combined solution routes.

The size of *Pop* (*PopSize*) is typically at least 10 times the size of the *RefSet* (b), that is,

$$PopSize = \mathrm{b} \times 10. \tag{1}$$

The initial *RefSet* is built in the *RefSet* update stage. *RefSet* is a collection of both high-quality and diverse solutions that are used to generate new solutions via the combination stage. In a basic design, a simple mechanism is used to build the initial *RefSet*, which is then updated during the search. The size of the *RefSet* (b) is given by

$$\mathrm{b} = \mathrm{b}_1 + \mathrm{b}_2. \tag{2}$$

Hence,

$$|RefSet| = |RefSet1| + |RefSet2|. \tag{3}$$

The construction of the initial quality *RefSet* ($RefSet_1$) starts with the selection of the best b_1 solutions from the *Pop*. These solutions are added to the *RefSet* and removed from the *Pop*.

For each solution in the *Pop*, the minimum of the distances to the solutions in $RefSet_1$ is computed. Then, the solution with the maximum of these minimum distances is selected. This solution is added to the initial diverse *RefSet* ($RefSet_2$) and removed from *Pop* and the minimum distances are updated. The process is repeated b_2 times. The resulting $RefSet_1$ has b_1 high quality solutions and $RefSet_2$ has b_2 diverse solutions [2].

In current work, we use b1 = 10, and b2 = 5 [2]. After generating the new solutions in the solution combination stage, these solutions are enhanced by improvement stage, and a solution becomes a member of the *RefSet* if one of the following requirements is satisfied: 'The objective function value of the new solution is better than that of the solution with the worst objective value in $RefSet_1$' and 'The diversity value of the new solution is better than that of the solution with the worst diversity value in $RefSet_2$' [5].

The algorithm continues to operate while the *RefSet* is being changed. If no change occurs in the *RefSet*, then the algorithm will check if the iterations reached the maximum number of iterations (MaxIter) detected by the user, and subsequently displays the good solution(s) reached; else, a new population is generated, and *RefSet* is added to the start of this population [6].

3 Dynamic Scatter Search

RefSet is the core of any SS framework. If all the reference solutions are alike at any given time during the search, as measured by an appropriate metric, then the SS procedure is most likely incapable of improving the best solution found even if a sophisticated procedure is employed to produce combinations solutions. The proposed algorithm starts by generating the initial diverse population (*Pop*). Each solution in the population is then evaluated to find its quality. After this, an iteration of SS will be performed on the candidate pool to generate a new *RefSet*.

The pseudo-code for Dynamic Scatter Search algorithm implemented in this paper is given in Fig. 1.

The first stage performed of SS iteration is *RefSet* creation as seen in above pseudo code. After that, the process will enter to the local scatter search loop. Before everything, inside this loop to choose the candidate solutions, which will be used in generating RefSet, is done be using subset generation stage. The second stage inside local loop is combination stage which used to combine the candidate solutions produced by subset generation stage. Each solution produced from combination stage will be improved to better one with improvement stage. The final step is to replace the some worst solutions with the best solutions from the RefSet, in order to make the search are more close more possible from the optimal

```
Pop = Diversification Generation function ()
Preprocessing function ()
RefSet= ∅
While (stopping criteria not satisfied //General Loop
  Dynamic Pop Update function ()
  RefSet Update function ()            //Creation RefSet
  While (RefSet not been updated)      //Inner Loop
   Subset Generation function ()
   Combination and Dynamic RefSet Update Function ()
   Improvement function ()
   RefSet Update function ()           //Updating RefSet
  EndWhile
EndWhile
```

Fig. 1 Dynamic scatter search pseudo code

solution. This technique is called RefSet update and it will make sure that the best solutions always carried over RefSet.

In the basic design, the new solutions that become members of the *RefSet* are not combined until all qualified pairs are subjected to the combination stage. The new *RefSet* is constructed with the best solutions in the union of a pool and the solutions currently in the *RefSet*. This process is called static *RefSet* update. The alternative to this static update is the *RefSet* dynamic update technique, which applies the combination stage to new solutions. Instead of waiting until all the combinations have been generated to update *RefSet*, a new trial solution warrants admission in *RefSet*. Therefore, the set is immediately updated before the next combination is performed.

Diversification aims to produce solutions that differ from each other, such that they represent a different solution space for the problem. By contrast, the randomization is used to construct solutions that are simply different from each other. Moreover, measuring the diversity value between solutions allows the algorithm to keep these solutions uniformly distributed among different regions [9]. As seen previously, $RefSet_2$ has the most diverse solutions. The diversity value among the solutions in *Pop* and $RefSet_1$ are measured; therefor, the solutions with the highest value of diversity are included in $RefSet_2$.

In the basic SS design, a new trial solution is assessed for entry into *RefSet*. If the new trail solution (resulting from the combination stage) is better than the *RefSet* solution with the worst quality, then it will immediately be transferred to the *RefSet*, whereas the worst will be removed from the *RefSet*. If the new trial solution is not better than the worst *RefSet* solution in terms of quality, then it will be discarded.

Furthermore, in the basic SS design, when the solution that generated from the combination process is compared with those in the *RefSet* and has no permitted to be in the *RefSet*, the solution will immediately be rejected and eliminated. In the improved design, we use *Pop* dynamic update, which allows rejected solutions to be allocated to the general *Pop*. In other words, the solutions produced in the combination stage are given a new chance to be re-improved in the improvement stage.

4 Computational Experience

The improved SS has been programmed and realized to solve the TSP. In this work, the proposed algorithm is examined on 20 different instances. The improved SS is coded using Microsoft Visual C# 2010 Ultimate versions and performed on a personal computer with Intel® Core™ i3 2.53 GHz CPU and 3 GB RAM running on 32-bit Windows 8 operating system.

The results obtained for the usual TSPLIB problems with less than 532 cities, used for example in [10, 11], are presented in Table 1. In this table, the first column presents the name of the instance to which the size of the instance is joined. Instances are arranged in descending order by size. The second column indicates the optimum tour length. The other columns present the results for the different approaches considered. All the results are obtained after 32 runs of the algorithm. The percentage deviation from the optimum of the mean solution value after 32 runs is described in the column "%PDM". The percentage deviation from the optimum of the best solution value found is presented in column "%PDB".

Table 1 Comparison of the improved SS with the Memetic and Co-Adaptive algorithms among 20 TSPLIB instances with less than 600 cities

Problem	Optimal	Scatter search			Memetic SOM			Co-AdaptiveNet		
		% PDM	% PDB	Sec	% PDM	% PDB	Sec	% PDM	% PDB	Sec
eil51	426	0.88	0.00	1.10	2.14	1.64	0.25	2.89	0.94	0.04
berlin52	7542	0.54	0.00	1.22	2.01	0.00	0.30	7.01	0.00	0.05
st70	675	2.32	0.00	1.24	0.99	0.59	0.36	1.72	0.89	0.09
eil76	538	1.38	0.00	1.73	2.88	2.04	0.39	4.35	2.04	0.08
kroA100	21282	2.91	0.63	8.03	1.14	0.24	0.53	1.31	0.57	0.18
kroB100	22141	3.28	0.96	6.69	1.75	0.92	0.52	2.20	1.53	0.18
eil101	629	2.07	0.48	7.22	3.15	2.07	0.51	3.78	1.11	0.19
lin105	14379	3.64	0.88	5.98	0.34	0.00	0.55	1.08	0.00	0.20
pr107	44303	1.25	0.15	4.01	0.67	0.14	0.62	4.41	0.18	0.19
pr124	59030	3.78	1.75	2.22	1.52	0.26	0.63	2.93	2.36	0.29
bier127	118282	5.94	1.43	4.12	2.78	1.25	0.80	3.00	0.69	0.29
ch130	6110	3.93	1.87	5.50	2.83	0.80	0.66	2.82	1.13	0.30
ch150	6528	5.75	2.33	6.40	2.95	1.67	0.78	3.23	1.78	0.40
kroA150	26524	5.05	0.91	8.71	2.73	1.64	0.76	3.06	1.55	0.43
kroB150	26130	3.36	1.23	7.67	1.61	0.74	0.77	2.60	1.06	0.40
pr152	73682	4.77	2.02	1.49	2.60	1.57	0.89	2.06	0.74	0.43
kroA200	29368	2.69	0.95	10.01	2.20	1.08	1.06	3.27	0.92	0.58
kroB200	29437	3.31	1.53	10.76	3.92	1.82	1.07	2.31	0.88	0.58
lin318	42029	6.04	2.89	14.04	5.51	3.63	1.86	4.31	2.65	1.53
att532	27686	8.08	5.84	24.25	4.21	3.29	3.80	5.31	4.24	3.22
Av		3.69	1.51	7.34	2.61	1.41	0.89	3.44	1.6	0.52

The results of our comparative computational tests are reported in Table 1. These tests use the 20 problems solved by [10, 11]. For the test, we adopted 15 runs, whereas the authors of the previous two studies provided the best solution found after 10 runs. Nevertheless, such difference does not affect the calculated execution time because time is based on the rate of the implementation of all the solutions. Table 1 shows the comparison between the best results obtained by the proposed algorithm in this work with the best known results from the literature i.e. Memetic and Co-adaptive. It can be clearly seen that the best results obtained by our approach are competitive to the previously best known results on average considering both the mean (PDM) and the best (PDB) solutions.

Generally, our approach with the small size instances DSS has an efficient record in quality of solutions, but with the large instances there is a clear gap especially in computation times. According to the results shown in above table, the proposed SS algorithm is not the best choice compared with the other tested algorithms. The proposed algorithm takes more computation time; however, it generates results that are close to those of the other algorithm. To be competitive, the solution quality and the computation time of the proposed SS, both must be improved both by a factor ten.

5 Conclusions

This paper presents a dynamic Scatter Search (SS) algorithm to solve the well-known TSP problem. The effectiveness of the dynamic Scatter Search algorithm was compared with that of the basic of SS design and two other algorithms based on the basis of the percentage deviation of the mean solution (%PDM) and the percentage deviation of the best solution (%PDB). Compared with a previous SS algorithm with basic design, the proposed SS algorithm improves the solution quality on standard test problem cities but with a possibly higher computation time. Compared with other approaches, the proposed improved SS is relatively simple to understand and to implement, and it offers good solutions. However, the proposed SS consumes a longer execution time, but with the dynamic updating technique, the time gap is decreased as much as possible. In our future study, the proposed SS algorithm will be improved by a combining it with advanced *RefSet* specifically dedicated to the TSP. As given that most of the computation time consumed is related to the main loop of SS, study a lower-cost inexpensive improvement loop coupled with some a number of standard construction strategies may be considered for the future approach.

Acknowledgments The study is supported by Project No.: RG312-14AFR from University of Malaya.

References

1. Glover Fred (1977) Heuristics for integer programming using surrogate constraints. Decis Sci J 8:156–166
2. Glover F, Laguna M, Mart´ı R (2000) Fundamentals of scatter search and path relinking. Control Cybern 39:653–684
3. El-Ghazali T (2009) Metaheuristics from design to implementation, 1st edn. Wiley
4. Liu Y-H (2007) A hybrid scatter search for the probabilistic traveling salesman problem. Computers and Operations Research, vol 34. Elsevier Ltd, pp 2949–2963
5. Pantrigo JJ, Duarte A, Sánchez Á, Cabido R (2005) Scatter search particle filter to solve the dynamic travelling salesman problem. In: Evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 3448. Springer, Berlin Heidelberg, pp 177–189
6. Sagheer AM, Sadiq AT, Ibrahim MS (2012) Improvement of scatter search using Bees Algorithm. In: 6th International Conference, Signal Processing and Communication Systems (ICSPCS), IEEE, Gold Coast. pp 1–7
7. Russell RR, Chiang WC (2006) Scatter search for the vehicle routing problem with time windows. European J Oper Res 169:606–622. Elsevier
8. Tang J, Zhang J, Fung RYK (2011) A scatter search for multi-depot vehicle routing problem with weight-related cost. Asia-pacific J Oper Res 28:323–348
9. Alkhazaleh HA, Ayob M, Othman Z, Ahmad Z (2013) Diversity measurement for a solution of team orienteering problem. Int J Adv Comput Technol (IJACT). 5:21–28
10. Créput J, Koukam A (2009) A memetic neural network for the euclidean traveling salesman problem. Neurocomputing 72 1250–1264. Elsevier Ltd
11. Cochrane EM, Beasley, JE (2003) The Co-adaptive neural network approach to the euclidean travelling salesman problem. Neural Netw 16:1499–1525. Elsevier Ltd