Contemporary Engineering Sciences, Vol. 10, 2017, no. 3, 129 - 144
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ces.2017.68146

The Enhanced Best Performance Algorithm for the Travelling Salesman Problem*

Mervin Chetty and Aderemi O. Adewumi

School of Mathematics, Statistics and Computer Science
University of Kwa-Zulu Natal, University Road, Westville, Private Bag X 54001
Durban, 4000, South Africa

Abstract

The Travelling Salesman Problem (TSP) is a largely studied discrete optimization problem, which is NP-Hard. This study compares the performances of three local search metaheuristic algorithms, in determining solutions to seven benchmark instances of symmetric TSP's. The algorithms investigated include the enhanced Best Performance Algorithm (eBPA), Tabu Search (TS) and Simulated Annealing (SA). This study is the first on the eBPA for the TSP. eBPA is a newly introduced metaheuristic in the literature. The results demonstrate the effectiveness of eBPA as an alternative optimization technique for difficult optimization problems.

Keywords: enhanced Best Performance Algorithm (eBPA), Simulated Annealing (SA), Tabu Search (TS), Symmetric Travelling Salesman Problem (sTSP)

1. Introduction

The Travelling Salesman Problem (TSP) is defined as the problem of finding a minimal tour which traverses a list of n cities in a way in which every city is visited exactly once, except for the original city of departure where the salesman would start and finish. The problem is accounted to Euler in 1759, who presented a problem of trying to move a knight to every block on a chest board exactly once. The problem gained fame in a handbook written by B. F. Voigt in 1832 [14].

* The financial assistance of the National Research Foundation (DAAD-NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the DAAD-NRF.

Several classifications of TSP's exist; yet in this study, we investigate Symmetric Travelling Salesman Problem's (sTSP's). For sTSP's, the distance traveled from city i to city j is the same as the distance travelled from city j to city i. The TSP is formally defined as follows [10];

Given a set of n cities, a maximum travelling distance D, and for every pair of adjacent cities v_i and v_j a travelling distance $d(v_i, v_j)$ ($\forall\, i, j = 1,2, \dots, n$). The objective is to find a permutation $P = v_{p_1}, v_{p_2}, \dots, v_{p_n}$ that minimizes the travelling distance in satisfying the constraint,

$$\sum_{1 \le i < n} d(v_{p_i}, v_{p_{i+1}}) + d(v_{p_n}, v_{p_1}) \le D \tag{1}$$

The TSP is one of the most studied problems in discrete optimization. The complexity of the problem was proved by Richard. M. Karp [7].

The rest of this paper is structured as follows. Section 2 give descriptions of previous research work performed. Section 3 describes the local search metaheuristic algorithms to be investigated. Section 4 lists the seven benchmark test instances to be implemented. Section 5 presents and discusses the experimental results. Finally, section 6 draws conclusions and outlines possible future work.

2. Previous Research

Several local search approximation algorithms have been studied for the TSP. The most successful include: 2-opt, 3-opt, λ-ops and Lin-Kernighan (LK) [2]. λ-opt algorithms involve iteratively removing λ edges, and replacing these with different edges in reconnecting the tour. The objective is to find shorter tours without cycles. LK is a λ-opt heuristic which dynamically determines suitable values for λ, per iteration. Most λ-opt moves can be performed as sequential moves. The simplest non-sequential move is the 4-opt move; it is called the double-bridge move [11].

Several metaheuristic algorithms have also been investigated for TSP's. Common metaheuristics, with examples, include: the Genetic Algorithm (GA) [12], Evolutionary Algorithms (EA's) [19], Tabu Search (TS) [20], Simulated Annealing (SA) [15, 22], Ant Colony Optimization (ACO) [3], Particle Swarm Optimization (PSO) [18, 21] and the Firefly Algorithm (FA) [9].

3. Local Search Algorithms

This study investigates three local search (LS) metaheuristic algorithms in determining solutions to benchmark instances of sTSP's. The metaheuristics include the enhanced Best Performance Algorithm (eBPA), Tabu Search (TS) and Simulated Annealing (SA). TS and SA are well-known LS metaheuristic algorithms in the literature. On the other hand, the eBPA is newly introduced [1] (however, in this study a thorough explanation of the eBPA is given). The eBPA embeds characteris-

tics of both stochastic search, similar to that of SA, and deterministic search, similar to that of TS, in its technical and strategic design. For these reasons, the eBPA will be compared against TS and SA to measure performance. Descriptions of these algorithms are given below.

3.1 The enhanced Best Performance Algorithm

The eBPA is modeled on the analogy of professional athletes desiring to improve upon their best registered performances within competitive environments. Numerous sporting disciplines exist, however the principles are the same in that professional athletes desire to improve upon their skill levels with the purpose of trying to supersede their previous best known performances. To start off with, all professional athletes develop an interest in the particular sport; they then realize the potential to succeed. Thereafter, with constant practice and strategizing, their skill levels increase. This happens as a result of learning from trial and error. In using trial and error, refined skills are developed by improving upon their strengths and weaknesses in the sport. The ultimate goal of the athlete is to develop a level of skill that would result in the athlete giving off a performance that would ultimately surpass any previous best performance.

Apart from other learning strategies, an effective strategy could be to maintain an archive of a limited number of the best performances delivered by the athlete; for example, video recordings could be archived. Video recordings contain the history of the way a previous best performance had been delivered. This also includes the technique (or techniques) that was employed, and the result determined. Knowledge of this information could be used to motivate the athlete to deliver higher quality performances. For example, the information of the worst performance on the list could motivate the athlete to at least improve upon this minimum benchmark standard. Hence, if a performance is delivered which improves upon that of the worst performance already registered on the list, then the list could be updated by replacing the performance of the worst with that of the improved performance. In this way, the archive size will be maintained, but it also so happens that the quality of the worst performance on the list is now of a higher benchmark standard. Since this improved performance is the latest delivered, the athlete could then continue to work with the technique that was used to deliver that improved performance in trying to improve upon strengths and weaknesses.

Given the increased benchmark of the worst performance registered in the archive, the athlete is now presented with the challenge of working harder in order to deliver further improved performances. In maintaining the archive, the athlete may want to be challenged further by decreasing the archive size. Decreasing the archive size will make it increasingly more difficult for the athlete to register further improved performances.

"Technique" or "skill" in this context refers to a solution determined by an optimization technique. Also, the "result" of a performance refers to the quality of a solution in it being used to evaluate the objective function f. Therefore, notable similarities can be seen between an athlete delivering performances and an optimization technique determining solutions. Based on the analogy of professional athletes desiring to improve upon their best archived performances, the eBPA was conceptualized. There are six foundational rules governing the design of the eBPA:

1. An athlete maintains an archive of a collection of a limited number of best performances.
2. From this collection, the record of the worst performance is identified. This becomes the minimum benchmark standard for the athlete to try and improve upon.
3. If a new performance is delivered which improves upon (or is at least equivalent to) that of the worst performance, then the archive is updated by replacing the performance of the worst with that of the new. However, upon performing the update, if it is realized that the result of the new performance is identical to that of any other performance in the archive, but different in terms of the technique that had been employed, then the new performance will replace the one with the identical result.
4. The athlete will continue to try and improve upon the performance that caused the most recent update of the archive.
5. All performances registered in the archive must be unique in terms of result and technique.
6. The archive size is strategically reduced until only one performance remains.

To artificially simulate this analogy, the eBPA maintains a limited number of the best solutions found by the algorithm in a list called the Performance List (PL). From all solutions, the design variables constituting the construction of each solution must be adjacently different; therefore, only unique solutions are allowed admittance into the PL. Dis-allowing duplicate solutions will prevent the algorithm from working with previously visited solutions. Also, the best, the worst and (what would be called) the working solutions in the PL must be indexed. The best and worst solutions are identified according to their solution qualities. Henceforth, the best, the worst and the working solutions will be referred to by the variables $best$, $worst$ and $working$.

To try and improve upon the $worst$ solution registered in the PL, local search moves will be applied to a copy of the $working$ solution; hence, a new solution $working'$ will be realized. $working'$ is chosen from a subset of candidate solutions within the neighborhood region of $working$. If $working'$ at least improves upon $worst$, or is at least equivalent in solution quality, yet unique in terms of its design variables, then the PL will be updated. If the solution quality of $working'$ is different from all solutions in the PL, then the $worst$ solution will be replaced by $working'$. However, if the solution quality of $working'$ is identical to that of another solution in the PL, then $working'$ will replace that particular solution in the PL; this will ensure that there are no two solutions in the PL with the same solution quality.

Being newly inserted into the PL, $working'$ will then become the next $working$ solution. Also, if $working'$ has improved upon $best$, it will also be indexed as the new $best$ solution. Upon this update, the $worst$ solution would then need to be re-determined and re-indexed. The solution quality of the latest $worst$ solution will now become the new benchmark standard to try and improve upon. If an update of the PL has not been made, then local search moves will continue to be applied to the copy of $working$.

However, given a certain probabilistic factor, the next $working$ solution could also be that of $working'$ even though an update of the PL had not been performed. The probabilistic factor represents the desire of the athlete to try out a new technique; this will continue indefinitely as determined by the probabilistic factor.

These strategies represent the eBPA's ability to accept both improved and dis-improved $working'$ solutions. $working'$ is considered improved if it at least improves on $working$ in the PL. $working'$ is considered to be a dis-improved solution in two ways: if $working'$ is accepted into the PL yet is not an improvement over $working$; if $working'$ is not accepted into the PL and the probabilistic factor has been satisfied (i.e. this is a wayward solution that falls out of the scope of the solutions registered in the PL). Accepting dis-improved solutions is the eBPA's strategy of escaping premature convergence. Additionally, the admittance criterion shields against cycling.

An additional strategy is to dynamically reduce the PL size, until a PL size of one is achieved. Strategically decreasing the PL size allows for the admittance criterion to constrain further. Further constraining the PL size will intensify the search in sifting out higher quality solutions. This strategy also eliminates the possibility of cycling for PL sizes greater than one.

Also, to strategically try to break out of premature convergence (other than the strategies already encapsulated within the eBPA), an option exists to temporarily increase the size of the PL. However, temporarily increasing the PL size could open up the possibility of cycling in redirecting the search trajectory. The option of temporarily increasing the size of the PL is out of the scope of this initial research. After the termination criterion is satisfied, the $best$ solution will be returned. This solution is representative of the best performance delivered by the athlete. The eBPA is presented as Algorithm 1. In Algorithm 1: resize() checks to strategically resize the PL; is_PL_Populated() checks to see if the memory structure has been fully populated, and if not then it will populate it with $working'$ by calling method populate(…); perform_Update(…) inserts $working'$ into the memory structure and re-indexes the $best$, $worst$ and $working$ solutions where applicable.

Algorithm 1: The enhanced Best Performance Algorithm

1. Initialize variables: $bestIndex = 0,\ workingIndex = 0,\ worstIndex = 0$
2. Set the size of the Performance List, i.e. PL_size
3. Set probability p_a
4. Set the first solution in the Performance List, i.e. $PL_{workingIndex}$
5. Calculate the fitness value of $PL_{workingIndex}$, i.e. $PL\_Fitness_{workingIndex}$
6. Set Boolean variable $toggle\ =\ true$
7. **while** not Stopping_Criterion_Met() **do**
   - 7.1. **if** resize() **then**
     - 7.1.1. resize_PL()
   - 7.2. **end if**
   - 7.3. **if** $toggle$ **then**
     - 7.3.1. $working$ = Determine_Solution($PL_{workingIndex}$)
   - 7.4. **else**
     - 7.4.1. $working$ = Determine_Solution($working$)
     - 7.4.2. $toggle$ = true
   - 7.5. **end if**
   - 7.6. $f\_working$ = Determine_Fitness ($working$)
   - 7.7. **if** ($f\_working$ better than or equal to $PL\_Fitness_{worstIndex}$) and is_populated() **then**
     - 7.7.1. perform_Update($working, f\_working$)
   - 7.8. **else**
     - 7.8.1. **if** not is_populated() **then**
       - 7.8.1.1. populate($working, f\_working$)
     - 7.8.1. **end if**
   - 7.9. **end if**
   - 7.9. **if** random[0,1] $\leq p_a$ **then**
     - 7.9.1. $toggle = false$
   - 7.10. **end if**
8. **end while**
9. return $PL_{bestIndex}$

3.2 Tabu Search

TS is a neighborhood search algorithm based on the analogy of something that should not be touched or interfered with [5, 6]. This is implemented by maintaining a limited number of elite solutions (or specific solution attributes) in a list called the Tabu List (TL). The TL is commonly implemented in a first-in-first-out (FIFO) way, hence recording the most recent $|TL|$ best solutions found. In searching neighborhood regions of solution x, i.e. $N(x)$, the maximum number of neighbors considered is $N(x)\ -\ |TL|$, as any solution recorded in the TL has a tabu status and will not be interfered. This technique reduces the risk of cycling around local optima, as dis-improved moves are accepted in implementing its metaheuristic technique to escape premature convergence. The algorithm for TS is as follows.

Algorithm 2: Tabu Search

1. Initialize $best$ to be the initial tour
2. Set $current = best$
3. Evaluate the fitness of $best = f\_best$
4. Set $f\_current$ (the fitness of $current$) $= f\_best$
5. Set the size of the Tabu List, i.e. $tabuListSize$
6. Set the size of the Candidate List, i.e. $candidateListSize$
7. Initiate the Tabu List (TL) and the Candidate List (i.e. $CandidateList$)
8. **for** i to $noOfIterations$ **do**
   8.1. $CandidateList$ = Generate_New_Candidate_List($current$)
   8.2. $current$ = Find_Best_Candidate($CandidateList$)
   8.3. $f\_current$ = Determine_Fitness ($current$)
   8.4. **if** $f\_current$ better then f_best **then**
      8.4.1. $f\_best = f\_current$
      8.4.2. $best = current$
      8.4.3. Update TL with $current$
   8.5. **else**
      8.5.1. **if** Intensification_Criterion_Met() **then**
         8.5.1.1. $current$ = Reset_Current()
      8.5.2. **end if**
   8.6. **end if**
9. **end for**
10. **return** $best$

3.3 Simulated Annealing

SA [8, 17] is a Markov chain optimization technique modeled on the analogy of heated metal annealing to an equilibrium state. At higher temperatures the atomic composition of metal is more volatile, making the metallic structure unstable. However, when the metal starts to cool, the atomic structure becomes less volatile allowing it to stabilize. When completely cooled, an equilibrium state of stability is reached. For the annealing process to be successful, the decrease in the rate of temperature must be slow. The algorithm for SA is as follows.

Algorithm 3: Simulated Annealing

1. Initialize $best$ to be the initial tour
2. Set $current = best$
3. Evaluate the fitness of $best = f\_best$
4. Set $f\_current$ (the fitness of $current$) $= f\_best$
5. Initiate starting temperature T and final temperature F
6. **while** $T \geq F$ **do**
   6.1. **for** i to $stepsPerChange$ **do**
      6.1.1. $working$ = Determine_Solution ($current$)
      6.1.2. $f\_working$ = Determine_Fitness($working$)
      6.1.3. **if** $f\_working$ better then $f\_current$ **then**
         6.1.3.1. $use\_solution$ = true
      6.1.4. **else**
         6.1.4.1. Calculate acceptance probability P

6.1.4.2. **if** $P >$ random[0,1] **then**
6.1.4.2.1. $use\_solution$ = true
6.1.4.3. **end if**
6.1.5. **end else**
6.1.6. **if** $use\_solution$ **then**
6.1.6.1. $use\_solution$ = false
6.1.6.2. $f\_current = f\_working$
6.1.6.3. $current = working$
6.1.6.4. **if** $f\_current$ better than f_best **then**
6.1.6.4.1. $best = current$
6.1.6.4.2. $f\_best = f\_current$
6.1.6.5. **end if**
6.1.7. **end if**
6.2. **end for**
6.3. Update T according to cooling schedule α
7. **end while**
8. **return** $best$

4. Benchmark Test Instances

This study investigates seven sTSP benchmark test instances from the TSPLIB collection. This collection has been made available online by Gerhard Reinelt. The problem instances, along with their characteristics, are given in Table 1. For each problem instance, the name, the number of vertices, the distance calculation type (i.e. Euclidean 2D-plane) and the optimal tour-lengths are given.

Table 1: Symmetric Travelling Salesman Problem test instances, and their characteristics

No.	sTSP	No. of Vertices	Type	Optimal Tour-Length
1	ch150	150	EUC_2D	6,528
2	tsp225	225	EUC_2D	3,916
3	a280	280	EUC_2D	2,579
4	lin318	318	EUC_2D	42,029
5	pcb442	442	EUC_2D	50,778
6	rat575	575	EUC_2D	6,773
7	d657	657	EUC_2D	48,912

5. Results and Discussion

The objective of this investigation is to test using the same parameter settings, for each metaheuristic algorithm, in solving multiple instances of the sTSP's. The problem instances differ in complexity, and range from 150 to 657 cities/vertices. This will provide sufficient challenges for the algorithms for testing purposes. The strength of the solutions determined by the eBPA will shed light on its abilities, in being compared to TS and SA for the multiple test instances investigated.

To solve the problem instances, we first employ the Nearest Neighbor (NN) tour construction heuristic. This is used to provide the initial solution to each metaheuristic algorithm, per problem instance. The NN heuristic is straightforward: it is implemented by starting off at the first city, and thereafter it moves to the nearest adjacent unvisited city. The NN tour-length (or fitness) solutions is given in Table 2.

Table 2: Nearest Neighbor tour-length solutions for each problem instance

No.	sTSP	Nearest Neighbor Tour-Length
1	ch150	8,191
2	tsp225	5,030
3	a280	3,157
4	lin318	54,019
5	pcb442	61,979
6	rat575	8,605
7	d657	61,627

In the execution of the metaheuristic algorithms, the solution at each iteration is determined by selecting the best of six moves. The best move is the one that will result in having the lowest fitness value. The six moves are as follows:

1. 2-opt – The 2-opt move removes two edges from a complete tour. It then reconnects the tour by introducing two new edges, which join the opposite ends of the removed edges.
2. 3-opt – The 3-opt move is similar to the 2-opt, except that with 3-opt, three edges are removed instead of two. It is implemented as two sequential 2-opt moves; this results in two solutions.
3. Double-bridge – The double-bridge move is a non-sequentially move (unlike 3-opt), which is implemented by randomly dividing the complete tour into four segments, and then reconnecting it in the reverse order.
4. Random swap – This move is implemented by randomly selecting two vertices from a complete tour, and then swapping them.
5. Vertex reposition – This move is implemented by repositioning a randomly selected vertex at a randomly selected position in the tour.

The parameter settings for the algorithms will be as follows:

a) SA will be set according to recommendations from literature [16]: The initial temperate (T) will be set at 50% of the fitness of the initial solution, while the cooling rate alpha (α) will be set at 85%.
b) Likewise, the Tabu list size (TL_size) of TS will be set at 7 [4, 13]. To determine the Candiate List size (CL_size) for TS, we make use of the test instance pr439 from the TSPLIB collection; pr439 is also a EUC_2D sTSP.
c) Since this is the first research on the eBPA for the TSP problem, we also make use of the pr439 problem to determine the probability factor (p_a) and the Performance List size (PL_size) for eBPA.

Once the set of experiments are run to determine the remaining parameter settings, all parameter settings will remain constant for the second set of experiments, which will be to compare the performances of the algorithms for the multiple sTSP test instances.

For the first and second sets of experiments, the stopping criterion will be to terminate the execution at the point of convergence. Convergence is the point where further improvements in fitness will yield minimal benefits, compared to the large number of iterations required to yield those minimal benefits. In this study, convergence will be detected when no further improvements have been made to be $best$ solution for a large number of iterations. For the first set of experiments (i.e. to determine the parameter settings), convergence will be set at 3% of idle iterations. For the second set of experiments (i.e. in making algorithmic comparisons), convergence will be set at 5% of idle iterations. The termination criterion will apply provided that a minimum of 10^6 iterations have been executed. For example, if 10^6 iterations have executed, and the total number of consecutive idle iterations is 50,000 (assuming we are referring to the second set of experiments), then the algorithms will detect convergence and will terminate. For the first set of experiments, each algorithm will be run 50 times, per problem instance. For the second set of experiments, each algorithm will be run 30 times, per problem instance.

The program was written in Java, using Netbeans® 7.0 on a Windows® 7 Enterprise system, with an Intel® Celeron® Processor 430, 3GB of RAM and a 500GB hard-drive.

The experiment run to determine the CL size for TS is seen in Figure 1. The CL size's were randomly selected from within the range of $1 \leq CL\_size \leq 1{,}000$. For the 50 runs, the TL size remained constant at size 7. Figure 1 shows that CL size's below 200 determined weaker solutions, and that the most competitive solutions fell within the range of 400 to 1000. The best solution seen had a CL size value of 723. The CL size value of 723 will be the parameter value to be used for the second set of experiments.

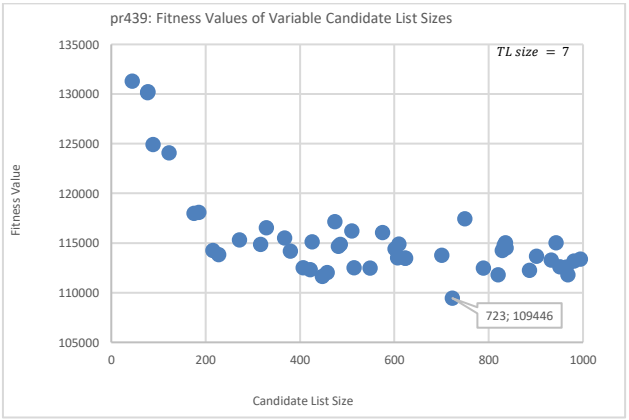

Figure 1: Fitness values determined by randomly selecting the $\boldsymbol{CL}$ size values.

The experiments run to determine p_a and the PL size for eBPA is seen in Figures 2 and 3. In Figure 2, the PL size remained fixed at 50, while p_a was randomly selected from within the range of $0 < p_a \leq 0.15$. From Figure 2, it can be seen that for the 50 runs, the solutions are scattered throughout the entire probability range, and roughly within the same height of the fitness range. There is no specific value for p_a that is best suited for this problem instance. The best solution seen had a p_a value of 0.045 (truncated to three decimal places). This will be the value used for the rest of the experiments.

For the experiment run to determine the PL size, the value of $p_a = 0.045$ remained fixed, while the value of the PL size was randomly selected from within the range of $1 \leq PL\_size \leq 200$. From the 50 runs, it can be seen that values greater than 130 determined the poorest solutions. The solutions determined in having used the values between that ranges of 50 to 130 determined competitive solutions; however, these solutions also show evidence of having determined slightly weaker solutions. The most consistent and competitive cluster of solutions can be seen within the value range of 4 to 31, with the best solution having had a PL size of 10. Together with $p_a = 0.045$, these will be the parameter values for the eBPA for the second set of experiments.

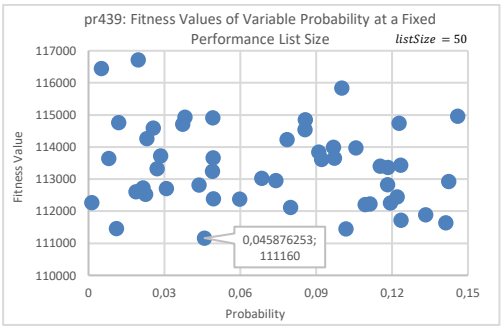

Figure 2: Fitness values determined using randomly selected probability factors, at a fixed ***PL*** size of 50

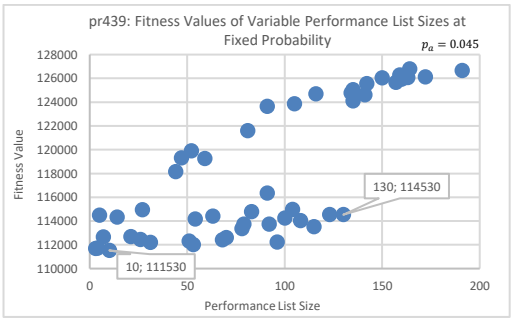

Figure 3: Fitness values determined using randomly selected ***PL*** sizes, at a fixed probability factor of 0.045

For the second set of experiments, the parameter values for all algorithms remained constant for all problem instances implemented. For each algorithm, per problem instance, their best (BFV) and average (AFV) fitness value solutions have been documented. For the average fitness values, their 95% Confidence Interval (95% CI) values are also given. The 95% CI values mean to be 95% certain that the 30 execution times of the algorithms, per problem instance, have fallen within those interval estimates. The statistical results are given in Table 3.

Table 3: Best, average and 95% CI fitness values, for each algorithm, per problem instance

sTSP	eBPA			TS			SA		
	BFV	**AFV**	**95% CI**	**BFV**	**AFV**	**95% CI**	**BFV**	**AFV**	**95% CI**
ch150	6,563	6,643	AVG $\pm$ 23	6,563	6,664	AVG $\pm$ 23	6,543	6,683	AVG $\pm$ 30
tsp225	3,971	4,011	AVG $\pm$ 8	3,988	4,034	AVG $\pm$ 8	3,993	4,064	AVG $\pm$ 14
a280	2,637	2,677	AVG $\pm$ 7	2,654	2,705	AVG $\pm$ 7	2,638	2,726	AVG $\pm$ 13
lin318	43,233	43,685	AVG $\pm$ 126	43,492	44,310	AVG $\pm$ 156	43,485	44,340	AVG $\pm$ 197
pcb442	51,519	52,400	AVG $\pm$ 146	52,257	53,071	AVG $\pm$ 161	52,584	54,148	AVG $\pm$ 263
rat575	6,955	7,062	AVG $\pm$ 18	7,024	7,093	AVG $\pm$ 14	7,206	7,290	AVG $\pm$ 20
d657	50,475	51,048	AVG $\pm$ 107	50,564	51,492	AVG $\pm$ 141	51,699	53,076	AVG $\pm$ 189

From Table 3, it is observed that eBPA determined the best BFV solutions for all problem instances, except for ch150. For ch150, SA determined the best BFV solution. However, eBPA did determine the best AFV solutions, and the lowest 95% CI values for all problem instances. Visual representations of the statistics given in Table 3 are seen in Figures 4 to 10. For the AFV solution towers, the 95% CI values are represented as the black interval estimates at the top.

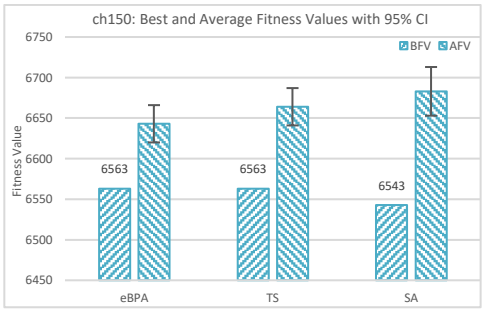

Figure 4: The best and average fitness values, along with their 95% CI estimates for ch150

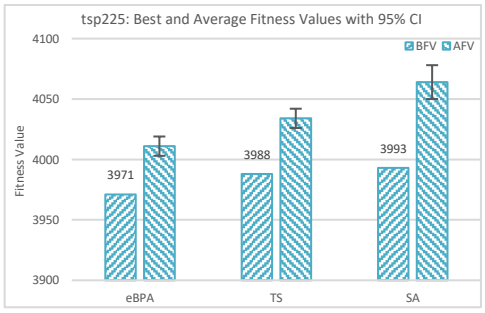

Figure 5: The best and average fitness values, along with their 95% CI estimates for tsp225

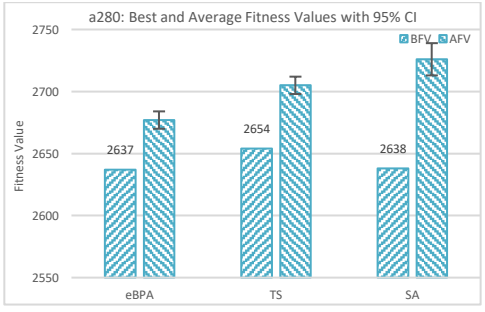

Figure 6: The best and average fitness values, along with their 95% CI estimates for a280

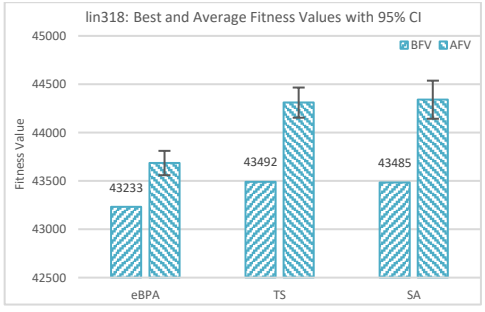

Figure 7: The best and average fitness values, along with their 95% CI estimates for lin318

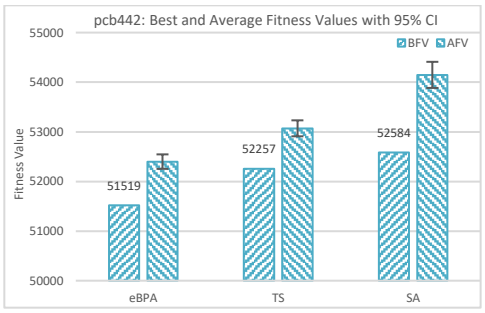

Figure 8: The best and average fitness values, along with their 95% CI estimates for pcb442

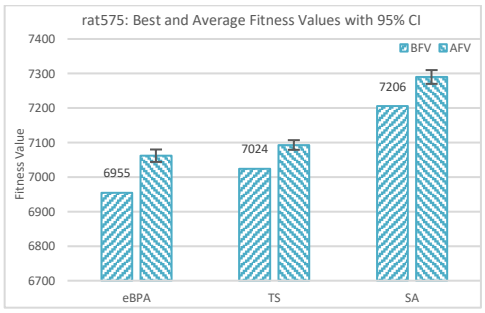

Figure 9: The best and average fitness values, along with their 95% CI estimates for rat575

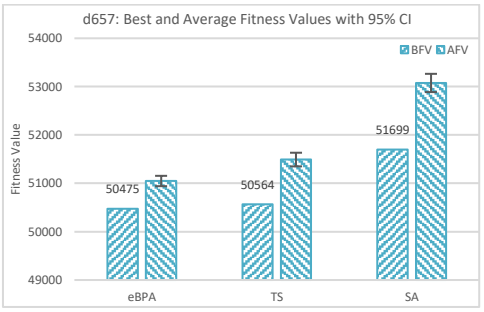

Figure 10: The best and average fitness values, along with their 95% CI estimates for d657

The experiment shows the eBPA's ability to determine competitive solutions across multiple problem instances, in using the same parameter settings. The strength of eBPA is attributed to its ability to influence the trajectory of the stochastic search by way of memory. The acceptance criterion of the eBPA memory structure allows for a $working$ solution to get inserted into the PL, provided it meets the minimum requirements (please refer to section 3.1). The newly inserted solution will then become the next solution to be used to direct the search. Implicitly, any solution inserted into the PL could possibly be the next $best$ solution. With every insert into the PL, the admissible criteria will constrain further as the quality of the $worst$ solution will improve. This makes acceptance into the memory structure more difficult, and controls the transition from exploration to exploitation. Exploitation is further enhanced by reducing the PL size, which also serves the purpose for eliminating cycling for PL sizes greater than one.

6. Conclusion

The TSP is largely studied in discrete optimization. Its complexity is NP-Hard. In this study, the eBPA, TS and SA metaheuristic algorithms were compared in their abilities to determine their best and average tour-length solutions, for the seven benchmark test instances investigated. The complexities of the problem instances differed in ranging from 150 to 657 vertices. The results show the competitiveness

of eBPA in determining solutions across multiple problem instances, in using the same parameter settings. Further research is required to investigate eBPA's ability to determine solutions for other types of optimization problems.

References

[1] S. Chetty and A.O. Adewumi, Three New Stochastic Local Search Algorithms for Continuous Optimization Problems, *Computational Optimization and Applications*, **56** (2013), 675-721. https://doi.org/10.1007/s10589-013-9566-3

[2] D. Davendra, *Traveling Salesman Problem, Theory and Applications*, InTech, 2010. https://doi.org/10.5772/547

[3] M. Dorigo and L. Gambardella, Ant Colonies for the Travelling Salesman Problem, *Biosystems*, **43** (1997), 73-81. https://doi.org/10.1016/s0303-2647(97)01708-5

[4] F. Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research*, **13** (1986), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

[5] F. Glover, Tabu Search - Part 1, *ORSA Journal on Computing*, **1** (1989), 190–206. https://doi.org/10.1287/ijoc.1.3.190

[6] F. Glover, Tabu Search - Part 2, *ORSA Journal on Computing*, **2** (1990), 4–32. https://doi.org/10.1287/ijoc.2.1.4

[7] R.M. Karp, Reducibility among Combinatorial Problems, Chapter in *Complexity of Computer Computations*, Springer US, 1972. https://doi.org/10.1007/978-1-4684-2001-2_9

[8] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by Simulated Annealing, *Science*, **220** (1983), 671–680. https://doi.org/10.1126/science.220.4598.671

[9] S.N. Kumbharana and G.M. Pandey, Solving Travelling Salesman Problem using Firefly Algorithm, *International Journal for Research in Science & Advanced Technologies*, **2** (2013), 53-57.

[10] S. Lin, Computer Solutions of the Traveling Salesman Problem, *Bell System Technical Journal*, **44** (1965), 2245-2269. https://doi.org/10.1002/j.1538-7305.1965.tb04146.x

[11] S. Lin and B.W. Kernighan, An effective heuristic algorithm for the travelling-salesman problem, *Operations Research*, **21** (1973), 498-516. https://doi.org/10.1287/opre.21.2.498

[12] S.J. Louis and R. Tang, Interactive Genetic Algorithms for the Travelling Salesman Problem, *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, **1** (1999), 1043-1048.

[13] M. Malek, M. Guruswamy and M. Pandya, Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Traveling Salesman Problem, *Annals of Operations Research*, **21** (1989), 59-84. https://doi.org/10.1007/bf02022093

[14] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 2nd Edition, 1994. https://doi.org/10.1007/978-3-662-07418-3

[15] M. Miki, T. Hiroyasu and T. Jitta, Adaptive Simulated Annealing for Maximum Temperature, *Proceedings of the 2003 IEEE International Conference on Systems, Man & Cybernetics (SMC 2003)*, (2003), 20-25. https://doi.org/10.1109/icsmc.2003.1243786

[16] E. Soubeiga, *Development and Application to Hyperheuristics to Personal Scheduling*, Ph.D. Thesis, University of Nottingham, 2003.

[17] C.M. Tan, *Simulated Annealing*, InTech Publisher, 2008. https://doi.org/10.5772/67

[18] F.M. Tasgetiren, P.N. Suganthan and Q.K. Pan, A Discrete Particle Swarm Optimization Algorithm for the Generalized Traveling Salesman Problem, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation - GECCO '07* (2007), 158-167. https://doi.org/10.1145/1276958.1276980

[19] H.K. Tsai, J.M. Yang, Y.F. Tsai and C.Y. Kao, An Evolutionary Algorithm for Large Traveling Salesman Problems, *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, **34** (2004), 1718-1729. https://doi.org/10.1109/tsmcb.2004.828283

[20] S. Tsubakitani and J.R. Evans, Optimizing Tabu List Size for the Travelling Salesman Problem, *Computers Operational Research*, **25** (1998), 91-97. https://doi.org/10.1016/s0305-0548(97)00030-0

[21] X. Yan, C. Zhang, W. Luo, W. Li, W. Chen and H. Liu, Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm, *IJCSI International Journal of Computer Science Issues*, **9** (2012), 264-271.

[22] X. Yao, Dynamic Neighbourhood Size in Simulated Annealing, *International Joint Conference on Neural Networks (IJCNN 1992)*, **1** (1992), 411-416.

Received: September 7, 2016; Published: January 9, 2017