International Journal on
Advanced Science
Engineering
Information Technology

Vol. 2 (2012) No. 6
ISSN: 2088-5334

A Comparison of Exhaustive, Heuristic and Genetic Algorithm for Travelling Salesman Problem in PROLOG

Nur Ariffin Mohd Zin[1] , Siti Norul Huda Sheikh Abdullah[2], Noor Faridatul Ainun Zainal[3]
Esmayuzi Ismail[4]

[1,2,3,4]*Center for Artificial Intelligence Technology, Faculty of Information Science and Technology*
Universiti Kebangsaan Malaysia 43600 Bangi, Malaysia.
E-mail: ariffin@uthm.edu.my$_1$*, mimi@ftsm.ukm.my*$_2$*, farida@ftsm.ukm.my*$_3$*, esmayuzi@ktb.edu.my*$_4$

[1]*Faculty of Computer Science and Information Technology Universiti Tun Hussein Onn Malaysia*
86400 Parit Raja, Malaysia.

***Abstract*— This paper discusses on a comparative study towards solution for solving Travelling Salesman Problem based on three techniques proposed namely exhaustive, heuristic and genetic algorithm. Each solution is to cater on finding an optimal path of available 25 contiguous cities in England whereby solution is written in Prolog. Comparisons were made with emphasis against time consumed and closeness to optimal solutions. Based on the experimental, we found that heuristic is very promising in terms of time taken, while on the other hand, Genetic Algorithm manages to be outstanding on big number of traversal by resulting the shortest path among the others.**

Keywords— **travelling salesman; exhaustive; heuristic; genetic algorithm; prolog; cities; cuckoo algorithm;**

I. INTRODUCTION

Travelling Salesman Problem (TSP) has been a prominent problem discussed widely among scholars in the community of mathematics and computer science. It was started by Irish and British mathematician, namely Hamilton and Thomas Kirkman in 1800s. The idea is to travel along a number of cities, *n* while maintaining to visit each city only once through a shorter path and returning to the starting city. It is a non-trivial problem, hence acquiring a sort of mathematical explanation to solve. A lot of effort has been made to conceive a new algorithm as to tackle the most optimized solution. Over years, the size of TSP solved has expanded dramatically, from solution of 49 cities [1] up to 85,900 cities recently.

TSP is feasibly solved by illustrating it as a graph as illustrated in Fig. 1. Cities are depicted as vertices, along with edges that resemble path. Thus, leading to acquiring shortest Hamiltonian cycle is the great concern towards solution. Given 2 cities, traversal could be done in both ways, as a symmetric TSP should be. On contrary, an asymmetric TSP may not allow traversal on both direction, which means the cost should not necessary the same.

Our aim here is to provide a concrete comparison in solving TSP via exhaustive, heuristic and genetic algorithm in regards of optimum cost of traversal. We opted to measure the cost in terms of time taken and utmost shortest path.

The rest of this paper is organized as follows. Section two elaborates on state of the art, reviewing the concept and several related studies from the past. Third section briefly discusses on proposed method to perform the solution. Section four reviews the result obtained by each method utilized. We conclude this paper in a brief and concise summary in section five. Last but not least will be a section of references that referred throughout the study.

Fig. 1 Example of 4-cities TSP as a graph based on [12]

II. State of the Art

Solutions are classified into two categories, either exhaustive or heuristic [2]. Exhaustive seems to be promising in a way it leads to ideal result. The search space for TSP is to search the all traversal combination; a symmetric determined by (n-1)!/2, an asymmetric defined by (n-1)!; where n is the number of cities. For instance, to solve a 5 cities problem consumes 24 possible solutions for asymmetric and half the number if it is symmetric. However, the efficiency lies within a polynomial factor of O($n.n$!), hence impractical in some way, especially for big number of cities. Exhaustive approach would search all (n-1)! possible paths and keep the one with the optimal cost which is shown in Algorithm 1. Inspired mathematically, there are such a quite numbers of its kind as explained in [3] namely, cutting plane method, branch-and-bound and branch-and-cut. Cutting plane method was presented by Dantzig et al. [1] which has solved an instance of 49 cities to optimality by constructing a tour and proving that no other tour could be shorter. Branch-and-bound by Land and Doig [4] can be determined as relaxation of upper and lower bound to eliminate unnecessary solution candidates. While branch-and-cut incorporates branch-and-bound and cutting plane method [5]. Apparently, of all those, the most breakthrough achieved stated to be a solution for about 85,900 cities [6]. As the number of cities exceeds, alternatively, we may have to rely on heuristic approach.

Algorithm 1: Exhaustive search algorithm:
Finds every possible route and returns the shortest [4].

```
Input : An undirected graph G = (V,E) with edge
weights We.
Output: A Hamilton cycle defined by the edges S
weight(T)= Σe∈E,We
Step 1: w = w(1,2) + w(2,3) + w(3,4) + ... + w(n-
1,n) + w(n,1)
Step 2: Best_S_so_far = ( n, [ 1, 2, 3, ... , n-1,
n ], w )
Step 2.1: S = ( 1, [ 1 ], 0 )
Step 2.2: Search( S, Best_S_so_far )
Step 2.3: print Best_S_so_far
Step 3: procedure Search( S, Best_S_so_far )
Step 3.1: let ( k, [ i1, i2, ... , ik ], w ) = S
Step 3.2: let ( n, [ i1B, i2B, ... , inB ], wB ) =
Best_S_so_far
Step 3.3: if k = n then new_w = w + w(ik,i1)
Step 3.3.1: if new_w < wB then Best_S_so_far = ( k,
[ i1, i2, ... , ik ], new_w )end if
Step 3.4: else for all j not in [ i1, i2, ... , ik ]
Step 3.4.1: new_w = w + w(ik,j)
Step 3.4.2: New_S = ( k+1, [ i1, i2, ... , ik, j ],
new_w )
Step 3.4.3: Search( New_S, Best_S_so_far )end for
end if
Step 4: return
```

Heuristic approach anticipates the solution guided by a given knowledge, which to some extent, leads to the exact result or remains merely approximation. Unlike exact algorithm, heuristic does not have to exhaustively permute all available paths, in which the time taken may be less nor reasonable. However, it may not guarantee of yielding an optimal result or even worse, not returning to the first city. It is marginal, but the possibilities are there. Instances of TSP employing heuristic are various; Lin-Kernighan by Lin and Kernighan [7], Tabu-Search by Glover [8], Simulated Annealing by Kirkpatrick et al. [9] and Cerny [10], Ant Colony by Dorigo and Gambardella [11] and quite a few. In this study, we take example of Greedy algorithm. This algorithm will always choose the next nearest city from the current city. According to Nilsson [12], the selection must not violate two conditions; 1) It doesn't create any cycle with less than n edges; 2) It doesn't increase the degree of any node to more than 2. Algorithm 2 shows the pseudo-code of Greedy algorithm.

As an addition of previously stated approaches, we opted for another called genetic algorithm (GA). Conceived by Holland [13], GA is a breed of a new form of nature-like heuristic. It is based on biological evaluation of a gene in human body. It starts by representing the problem as a population of solution candidates. The idea is to produce new offspring through a crossover and mutation process, with the intention to having the most fit candidates when the next evaluation takes place. Each candidate has its own fitness value to determine how good they are [12]. Applying GA to TSP may be varies on different studies. Some of them are spotted done by Braun [14] and Grefenstette [15]. However, a standard scheme proposed by Johnson and McGeoch [16] is used as our guide as presented in Algorithm 3.

Algorithm 2: Greedy search algorithm:
Takes each selected town in turn, and inserts it into the optimal location in the route as it grows.

```
Input: A connected undirected graph G = (V,E) with
edge weights We
Output: A minimum spanning tree defined by the edges
X
Step 1: for all u ∑ V :makeset(u)
Step 2: Set X = { }
Step 3: Sort the edges E by weight
Step 4: for all edges {u,v} є E, in increasing order
of weight:
Step 4.1: if find(u) ≠ find(v) then add edge {u,v}
to X
Step 5: union(u,v)
```

Algorithm 3: Genetic algorithm [5]

```
Input : An undirected graph G = (V,E); edge weights
We.
Output: A Hamilton cycle defined by the edges S
weight(T)= Σe∈E,We
Step 1: Generate a population of k starting
solutions S = {S1 , . . . , Sk}.
Step 2: Apply a given local optimization algorithm A
to each solution S in S, letting the resulting
locally optimal solution replace S in S.
Step 3: While not yet converged do the following:
Step 3.1: Select k¢ distinct subsets of S of size 1
or 2 as parents (the mating strategy).
Step 3.2: For each 1-element subset, perform a
randomized mutation operation to obtain a new
solution.
Step 3.3: For each 2-element subset, perform a
(possibly randomized) crossover operation to obtain
a new solution that reflects aspects of both
parents.
Step 3.4: Apply local optimization algorithm A to
each of the k' solutions produced in Step 3.3, and
let S' be the set of resulting solutions.
Step 3.5: Using a selection strategy, choose k
survivors from S ∪ S', and replace the contents of S
by these survivors.
Step 4: Return the best solution in S.
```

Another current searching algorithm for optimization approach to engineering design called Cukcoo algorithm [17]. They claimed that it has been demonstrated successfully and outperformed compared to other approaches including the advanced particle swarm optimization approach. This approach was inspired by

obligating brood parasitism of some cuckoo species as laying eggs in the nests of other host birds (other species). Some cuckoo female parasitic cuckoos are often very specialized in the mimicry in colour and pattern of the eggs of a few chosen host species [18]. Its pseudo-code can be summarized as Algorithm 4. However, this approach is still remained untested on travelling salesman problem. In conclusion, out of four searching algorithms, we only test on three of them namely, Exhaustive, Heuristic and Genetic algorithm.

Algorithm 4: Cukcoo Search Algorithm [6]

```
Input: Initial population of n host nests
Output: Best solution / new nest
Objective function: f(x), x = (x1, x2, . . ., xd);
Step 1: Generate an initial population of n host
nests;
Step 2: While (t<MaxGeneration) or (stop criterion)
Step 3: Get a cuckoo randomly (say, i) and replace
its solution by performing Lévy flights;
Step 4: Evaluate its quality/fitness Fi [For
maximization, Fi α f(xi)];
Step 5: Choose a nest among n (say, j) randomly;
Step 6: if (Fi > Fj),Replace j by the new solution;
end if
Step 7: A fraction (pa) of the worse nests are
abandoned and new ones are built;
Step 8: Keep the best solutions/nests;
Step 9: Rank the solutions/nests and find the
current best;
Step 10: Pass the current best to the next
generation;
Step 11: end while
```

III. Proposed Method

While other's work used common programming language, Java by Sengoku and Yoshihara [19], C by Zhi et al. [20], C++ by Chapel and Palma [21], we tried to break with precedent by using Prolog due to its simplicity and large library of Graphical User Interface (GUI) supports. Besides, TSP in Prolog has never been published, which has turned it into a motivation that spurred us. Prolog is a logic programming language which provides semantic presentation of statements in a form of facts, rules and queries [22]. A fact constitutes of relation among entities; queries are used in order to retrieve logic information from a fact; a rule is conjunction of facts to express new queries.

There are two factors to take into consideration upon designing the program, which are algorithm and GUI. Essentially, they interact each others, sending data back and forth and eventually produce the result. Based on Fig. 2, algorithm is a part of three techniques proposed, exhaustive, and heuristic and GA. While GUI consists of 3 vital elements; buttons, map and event-handler, which apparently served to map the location and traversal of city based on the input of user and output of the algorithm selected. The process starts as the selection of cities begin. The input data (cities) will be passed to the selected algorithm for processing. Once it has been processed, the algorithm will send the output (traversal and total path) back to GUI and process it in a graphical manner (map). The same process should apply on every algorithm selected.

This work proposed a program written by Steel et al. [23] with some endeavour on altering the program to fill in the GA part as well as meeting the GUI needs. The 3 algorithms proposed, exhaustive, heuristic and GA are resided in algorithm part and the visual part lies in GUI. Modification did take into account of three things. First of all, the GA code itself need to be included. During the inclusion, precaution is necessary to avoid any changes that will affect the existing algorithm. Meanwhile, the development of the code should match the call function as the other algorithm does, to which the input variables need to be passed. Otherwise, problem will occur during the GUI calls. Lastly, GUI representation needs to be catered, covering from a new button to the addition of event-handler during selection of cities. The outcome program is depicted in Fig. 3, which explains the sample selection of 10-cities with addition of GA function.

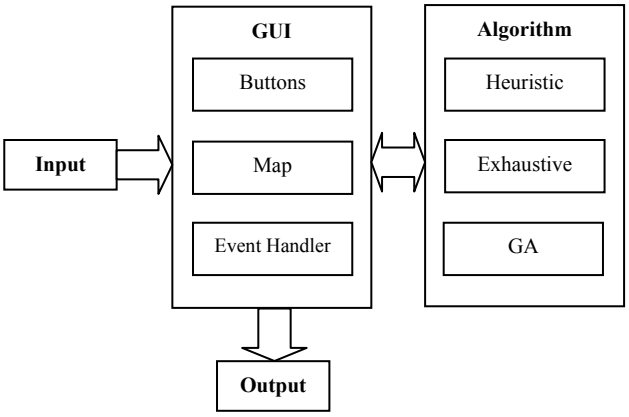

Fig. 2 Modified program design based on [12]

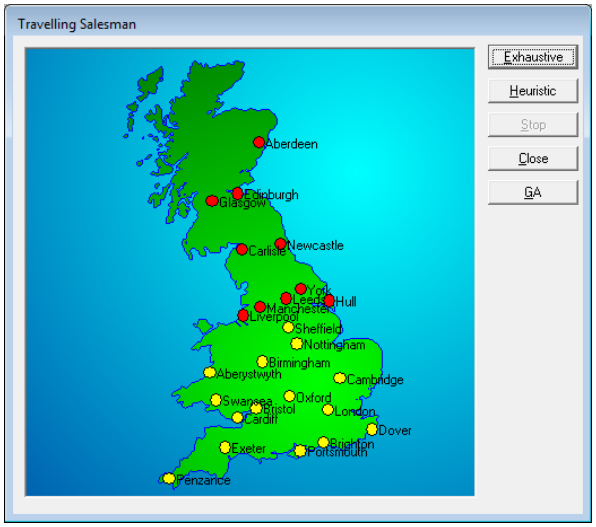

Fig. 3 An example of modified program with 10-cities selection in red with addition of GA function

IV. Experimental Results

Three methods were proposed to solve symmetric TSP by practising exhaustive, heuristic and genetic algorithm (GA). Solutions are written in Prolog, which then compiled and run with Win-Prolog 4320 application under a machine equipped with Intel Core 2 Duo 2.0GHz and 2GB of RAM. We used an existing program by Steel et al. [23] named SALESMAN.PL, whereby the solution for exhaustive and heuristic was already available. The code was written with respect of the graphical interface for ease. With some modifications took place, the resulting code was valid to

evaluate the problem in GA without loosing any graphical elements. 25 cities in England were opted which then broken into cases of different path. We started the evaluation for 4 cases, 4, 8, 12 and 25 cities consecutively. Measurement of time consumed and total path were recorded as in Table 1. It also shows the traversal from the starting city until it reaches back.

GA approach differs, as population size and maximum iteration need to be defined. During the experimental, the maximum iteration need to be defined. During the experimental, the maximum iteration for each case was heterogeneous. This is explained as the number of cities increrased, sticking to the same number of iteration throughout the process would results in inconsistency of path taken/total path. We experienced that insufficient amount of iteration may cause of training error escalates, thus requires training. We have determined the suitable iteration for each case; 100 – 4 cities, 170 – 8 cities, 400 – 12 cities, 900 – 25 cities. The best population size being used is referred from Whitley et al. [24], whereby determined by (1):

$$\text{Number of Cities} * 3 < \text{Population Size} < \text{Number of Cities} * 5 \quad (1)$$

We opted for 100 constantly. These figures were carried out throughout the experiment. Fig. 4 shows the results for every algorithm selected for a case of 8-cities. Based on the result shown in Table 1, in terms of total path, each approach seem to yield similar result until it reached 12 cities. During 25 cities, while exhaustive seems to fail, a big leap spotted between heuristic and GA. Apparently, GA outperformed heuristic by 2.5%, resulting lesser total path than heuristic. Exhaustive and GA have shown a salient time increases, however, until to some extent, exhaustive halted. While in the meantime, heuristic managed to stay consistent with a marginal increase. By referring to this result, we observed that heuristic took less time than exhaustive and GA. While GA outperformed the others on giving optimized path regardless of how big the traversal is. We are optimist that a better result can be obtained if other programming language is used. This is due to the nature of Prolog that always backtracks, causing a simple process taking longer than usual.

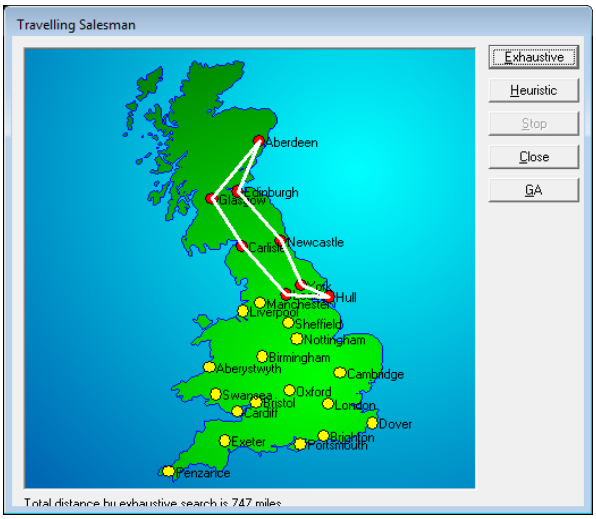

Fig. 4 Examples of 8-cities results based on Exhaustive, Heuristic and GA with 170 of iterations and 100 of population size

TABLE I
PATH TAKEN, TOTAL PATH AND TIME TAKEN FOR EACH CASE

Case (cities)	Path Taken		Total Path (miles) / Time Taken (seconds)
4	Exhaustive	Aberdeen-Edinburgh-Newcastle-Glasgow-Aberdeen	509 / 0.18
	Heuristic	Aberdeen-Edinburgh-Newcastle-Glasgow-Aberdeen	509 / 0.18
	GA	Aberdeen-Edinburgh-Newcastle-Glasgow-Aberdeen	509 / 0.58
8	Exhaustive	York-Newcastle-Edinburgh-Aberdeen-Glasgow-Carlisle-Leeds-Hull-York	747 / 0.18
	Heuristic	York-Hull-Leeds-Carlisle-Glasgow-Aberdeen-Edinburgh-Newcastle-York	747 / 0.18
	GA	York-Hull-Leeds-Carlisle-Glasgow-Aberdeen-Edinburgh-Newcastle-York	747 / 1.48
12	Exhaustive	Edinburgh-Aberdeen-Glasgow-Carlisle-Liverpool-Manchester-Sheffield-Nottingham-Hull-York-Leeds-Newcastle-Edinburgh	923 / 1440
	Heuristic	Edinburgh-Aberdeen-Glasgow-Carlisle-Liverpool-Manchester-Nottingham-Sheffield-Leeds-Hull-York-Newcastle-Edinburgh	926 / 0.18
	GA	Edinburgh-Aberdeen-Glasgow-Carlisle-Liverpool-Manchester-Sheffield-Nottingham-Hull-York-Leeds-Newcastle-Edinburgh	923 / 2.94
25	Exhaustive	Failed	Failed
	Heuristic	London-Dover-Brighton-Portsmouth-Exeter-Penzance-Bristol-Cardiff-Swansea-Aberystwyth-Liverpool-Manchester-Glasgow-Aberdeen-Edinburgh-Carlisle-Newcastle-Leeds-York-Hull-Sheffield-Nottingham-Birmingham-Oxford-Cambridge-London	2041 / 0.19
	GA	London-Cambridge-Oxford-Birmingham-Nottingham-Sheffield-Leeds-Hull-York-Newcastle-Edinburgh-Aberdeen-Glasgow-Carlisle-Manchester-Liverpool-Aberystwyth-Swansea-Cardiff-Bristol-Penzance-Exeter-Portsmouth-Brighton-Dover-London	1989 / 5.15

V. CONCLUSIONS

This work attempted on evaluating three methods for solving symmetric TSP for 25 contiguous cities in England. These methods were exhaustive, heuristic and genetic algorithm (GA) which then translated into Prolog as a solution. Exhaustive may be conventional but the result was impressive on yielding optimal total path and traversal. We have found that GA is a close rival to exhaustive, as their total path was similar, but at the end GA won. While, on the other hand, heuristic has outcame a close result as well as maintaining mild consume of time. During the cities proliferation, it is wise to conclude that exhaustive and GA suffers from time deterioration, which was also happen to heuristic but less. To produce a consistent result, GA may need bigger iteration with respect to number of cities, thus a lot of time taken. However, GA exceled compared to heuristic as we increased the number of cities to the highest. We may assume that with accurate iteration and population size, GA approach may be able to cater larger problems. Due to the restriction of Prolog that always backtrack upon finding solution, we observed that the result is still far behind. In future, we can enhance this work by using Cukcoo search algorithm.

ACKNOWLEDGEMENT

The authors would like to thank Faculty of Information Science and Technology, University Kebangsaan Malaysia for providing facilities and financial support under Research University operation Project No. UKM-GGPM-ICT-119-2010 and Fundamental Research Grant Scheme No. UKM-TT-03-FRGS0129-2010. Besides, special thanks to Pattern Research Group for their greatest contribution in this research.

REFERENCES

[1] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale travelling salesman problem," *Journal of the Operations Research Society of America,* vol. 2, pp. 393-410, Nov. 1954.

[2] D. Lin, X. Wu, and D. Wang, "Exact heuristic algorithm for traveling salesman problem," in *The 9th International Conference for Young Computer Scientists (ICYCS '08),* 2008, pp. 9-13.

[3] M. Hahsler and K. Hornik, "TSP infrastructure for the travelling sales-person problem," *Journal of Statistical Software*, vol. 23(2), pp. 1-21, Dec. 2007.

[4] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, pp. 497-520, 1960.

[5] J. E. Mitchell, *Branch-and-cut Methods for Combinatorial Optimization Problems*, *Handbook of Applied Optimization*, pages 65-77, Oxford, London: Oxford University Press, 2002.

[6] D. L. Applegate, R. M. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study, Princeton*, New Jersey: Princeton University Press, 2006.

[7] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," Operations Research, vol. 21(2), pp. 498-516, 1973.

[8] F. Glover, "Future paths for integer programming and links to artificial intelligence," Computers & Operations Research, vol. 13, pp. 533-549, 1986.

[9] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol 220, pp. 671-680, May 1983.

[10] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Application*, vol. 45, pp. 41-51, 1985.

[11] M. Dorigo and L. M. Gambardella, "Ant colonies for the traveling salesman problem", Universit Libre de Bruxelles, Belgium, 1996.

[12] C. Nilsson, "Heuristic for travelling salesman problem," Linköping University, Sweden, Journal of Theoretical Computer Science Reports, 2003.

[13] J. H. Holland, Adaptation in Natural and Artificial Systems, Cambridge, MA: MIT Press, 1992.

[14] H. Braun, "On solving travelling salesman problems by genetic algorithms," in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, 1991, pp. 129-133.

[15] J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proc. Int. Conf. on Genetic Algorithms and their Applications*, 1985, pp. 160-168.

[16] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization", in Local Search in Combinatorial Optimization, 1997, pp. 215-310.

[17] X.-S.Yang and S. Deb, "Engineering optimisation by cuckoo search," in Int. *J. Mathematical Modelling and Numerical Optimisation, 2010*, 1, pp. 330-343.

[18] R. B. Payen, M. D. Sorenso, K. Klitz, and J. Megahan, The Cukcoos, New York: Oxford University Press, 2005.

[19] H. Sengoku and I. Yoshihara, "A fast TSP solver using GA on Java," in *Proc. of the 3rd AROB*, 1998, pp. 283-288.

[20] X. H. Zhi, X. L. Xing, Q. X. Wang, L. H. Zhang, X. W. Yang, C. G. Zhou, and Y. C. Laing, "A discrete PSO method for generalized TSP problem," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2004, pp. 2378-2383.

[21] M. I. Capel and A. Palma, "A Programming tool for Distributed Implementation of Branch-and-Bound Algorithms," Parallel Computing and Transputer Applications, IOS Press/CIMNE, pp. 138-147, Sep. 1992.

[22] L. Sterling, E. Shapiro, and M. Eytan, The Art of Prolog, Advanced Programming Techniques Second Edition, England: The MIT Press, 1999.

[23] B. D. Steel, R. Shalfield, N. Johns, F. McCabe, P. Vasey, A. Westwood, and D. Westwood, "The Travelling Salesman," Logic Programming Associates, London, 2002.

[24] D. Whitley, T. Starkweather, and D.Shaner, The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, in Handbook of Genetic Algorithms, New York: Van Nostrand Reinhold, 1991.