01 Sep 2007

A Memetic Algorithm Configured Via a Problem Solving Environment for the Hamiltonian Cycle Problems

X. S. Chen

Meng-Hiot Lim

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Recommended Citation

X. S. Chen et al., "A Memetic Algorithm Configured Via a Problem Solving Environment for the Hamiltonian Cycle Problems," *Proceedings of the IEEE Congress on Evolutionary Computation, 2007*, Institute of Electrical and Electronics Engineers (IEEE), Sep 2007.
The definitive version is available at https://doi.org/10.1109/CEC.2007.4424821

A Memetic Algorithm Configured via a Problem Solving Environment for the Hamiltonian Cycle Problems

X.S. Chen, M. H. Lim, D. C. Wunsch II, *Member, IEEE*

Abstract—**Algorithm Development Environment for Permutation-based problems (ADEP) is a software environment for configuring meta-heuristics for solving combinatorial optimization problems. This paper describes the key features of ADEP and how the environment was used to generate a Memetic Algorithm (MA) solution for Hamiltonian Cycle Problems (HCP). The effectiveness of the MA algorithm is demonstrated through computer simulations and its performance is compared with backtracking and other heuristic techniques such as Simulated Annealing, Tabu Search, and Ant Colony Optimization.**

I. Introduction

Many real-world problems are known to be non-deterministic polynomial-time (NP) hard. Such problems are computationally intractable by conventional deterministic search algorithms. It is therefore not surprising that in recent years, researchers have begun to consider meta-heuristic algorithms such as Ant Colony Optimization, Genetic Algorithm, Tabu Search, Simulated Annealing and Memetic Algorithm. Still, in line with the potential complexity posed by the intractability of the problems, research have in recent years focused on hybridizing different search techniques to derive more powerful search algorithms. In general, such hybridization attempts to cultivate desirable search characteristics by drawing on a synergistic combination of global and local searches. One common direction is to combine genetic algorithms with at least one local search. Such an approach is also commonly referred to as memetic algorithms where the local search is akin to a meme, or in the context of a society, a unit of cultural information that is being passed on [1, 2, 3, 4 and 5].

One of the main issues faced by system developers of memetic algorithms is that there seem to be no "one size fits all" type of search algorithms [6, 7]. In other words, for any given memetic algorithm with a particular parametric setting, it is unlikely that it will perform equally well for all the problems of the same class. Hence, it is common to mix and match between meta-heuristics, or at least one meta-heuristic and one local search technique. The process of configuring such hybrid algorithms can be made easier if there is a computational platform to facilitate the algorithm design process. Our work here considers how such an environment can facilitate the design of algorithm for the Hamiltonian Cycle Problem (HCP).

X.S.Chen is with School of EEE, Nanyang Technological University, 637553 Singapore.(e-mail: chen0040@ntu.edu.sg)
M.H. Lim is with School of EEE, Nanyang Technological University, 637553 Singapore.(e-mail: emhlim@ntu.edu.sg)
D. C. Wunsch II is Dept. of Electrical & Computer Engineering, University of Missouri – Rolla, Rolla MO 65409 (email: dwunsch@ece.umr.edu)

The state of the art research on meta-heuristic algorithms has made significant advancement in the last two decades. This coupled with the technological advancement in computational hardware have opened up avenues to explore problems of complexity level that were previously considered insurmountable. In this regards, rapid design and development of meta-heuristic algorithms has become an important and challenging issue. The current meta-heuristic algorithm development methodology usually requires significant effort in codes generation and modifications. As a result, the quality of the meta-heuristic algorithm that solves a NP hard problem may be far from optimal in terms of performance. This is likely due to the fact that the designer may not have the resources or resolve to fine tune the algorithm. It is also likely that the programmer may not possess the necessary experience and knowledge on meta-heuristic algorithms to effectively make the necessary improvement.

ADEP is short for algorithm development environment for permutation-based problems, and was developed to address the need for rapid generation of efficient algorithms that target the real-life problems [8]. Currently ADEP is capable of generating and auto-configuring MA solutions for combinatorial optimization problems with permutation string solution representation. With ADEP, the user only needs to modify the objective/fitness function to convert the codes for solving the target problem. The time to implement the bulk of codes for the MA operators such as crossover, mutation, selection, fitness scaling, population update, etc. can be devoted to other more productive solution integration tasks. ADEP allows users to focus on the high-level problem abstraction, significantly reducing the algorithm development time and effort in designing and implementing the meta-heuristic algorithm to solve the problem.

The rest of this paper is organized as follow. In Section 2, we describe some of the more important features of ADEP from the viewpoint of algorithms design and development. Section 3 gives an overall introduction on the Hamiltonian cycle problem. In Section 4, we present some simulation results to give an indication of the performance level that is achievable by an algorithm configured in the ADEP

1-4244-1340-0/07/$25.00 ©2007 IEEE

environment before concluding this paper in Section 5.

II. FEATURES OF ADEP

Users interact with ADEP through a GUI (developed using MFC in VC++ 6) on which the various operators of a meta-heuristic algorithm are represented as components in a flow chart. The flow for data creates a simplified view of the process flow in the meta-heuristic algorithm, and the user interaction with the ADEP is through the modification or insertion of components in the work flow. The work flow of ADEP is illustrated in Figure 1.

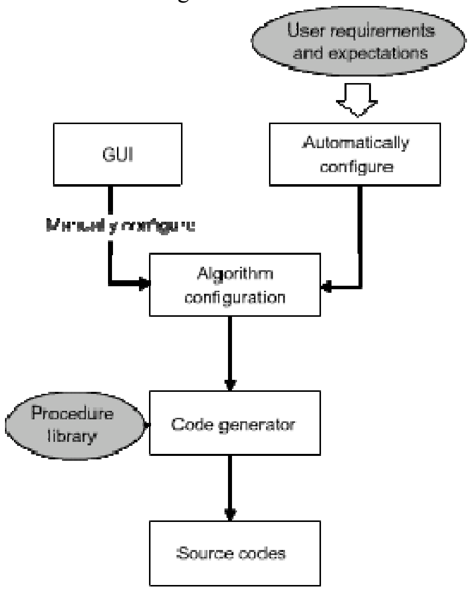


Figure 1: Work flow of ADEP

The work flow in ADEP is designed so as to minimize users' time in designing the algorithm and allow users to focus on the application of the algorithm to solve the intended problem at hand.

II. ADEP PROCEDURE LIBRARY

The procedure library of ADEP encapsulates a wide range of meta-heuristic routines. In the procedure library, the procedures, routines, parameters, etc. associated with the algorithm are organized and managed systematically though a novel tree structure named Left Variation – Right Property (LVRP) tree structure.

The LVRP tree structure works as follows. At the top level, an algorithm is divided into functional components (such as "Crossover", "Mutation" components as shown in Figure 2 of Section IV for the Memetic Algorithm). All the properties, routines and attributes within a functional component are grouped to form a tree structure in such way that each functional component has a "Variation" and a "Property" tree branch. The "Variation" tree branch enlists a collection of operators while the "Property" tree branch represents the common features of the particular operator as its children. Take the functional component "Offspring Producer" for the Memetic Algorithm as an example, the "Variation" branch of the "Offspring Producer" may contain "Crossover" and "Cloning" as its children while the "Property" branch may contain "Parent(s) Selection" as it child. [8]

The "Variation" and "Property" tree branch can have other branches. For instance, the "Crossover" can have "Uniform", "Order-1", "1-point" as its "Variation" and "Crossover Rate" as its "Property" while "Parent(s) Selection" can have "Random", "Roulette Wheel", "Tournament" as its "Variation". This rule is recursively applied as the system guides a user to configure a feasible algorithm [8]. In such a way, a complex algorithm configuration is broken down into a number of configurable "Variation" and "Property" objects.

During the user configuration stage, the user-specified features are added as the leaf nodes in the corresponding sub-trees of the LVRP. When the configuration is done, the code engine traverses the generated LVRP tree structures by depth-first traversal strategy to achieve a complete genetic algorithm. To improve the performance a generated MA for a particular type of problem, the code engine includes automatic algorithm optimizer. The automatic algorithms optimizer adopts meta-Lamarckian Learning (adaptive strategies for MA control that converge during training stage, to a set of operators chosen to locally improve the next chromosome [6, 7]). One of these strategies is the biased roulette wheel strategy MA-S2. The biased roulette strategy is a stochastic approach that makes use of knowledge gained online to form biases on the local methods (in terms of LRVP, the optimal "Variation" and the "Property" of each sub-tree that should be used for the particular problem). During the training stage, initially each local method is first given an equal opportunity to hybridize with the GA in optimizing the unseen parent chromosomes. After the initial stage, the probability that a local method will be chosen to work on any subsequent chromosome is biased according to its previous performance, which now changes dynamically as the search progresses. In this way, good configuration "Variation" and "Property" are rewarded with a higher chance of being selected in the next iteration and this preference is accumulated in a probabilistic and statistical manner until the evolutionary process converges with a few root-leaf paths dominating the trees by large relative preference values along them.

III. HAMILTONIAN CYCLE PROBLEM

The Hamiltonian Cycle Problem is NP-Complete (NP-C), which involves finding in a graph a cycle that contains all the vertices of the graph without visiting any of the vertices twice [9]. Hamiltonian cycles are named after William Rowan Hamilton who invented the Icosian Game, now also known as Hamilton's puzzle [10].

The Hamiltonian Cycle Problem is one of the most studied NP-C problems due to its complexity and it has a number of variants such as the traveling salesperson problem, Satisfiability problem (SAT), the knight's tour and graph coloring.

In proving the Hamiltonicity [9, 10] of a graph, Posa [11] was one of the first to consider the Hamiltonicity of random graphs and proved that a random graph $G \in G_{n,m}$, with $m = cn\log n$ has a high probability of containing a Hamiltonian cycle if c is sufficiently large. Shamir [12] proved that his algorithm finds a Hamiltonian path on $G \in G_{n,m}$ in $O(n^2)$ time if

$$m = \frac{1}{n}(\log n + c\log\log n), c > 3 \tag{1}$$

where m is the number of nodes and n is the number of edges.

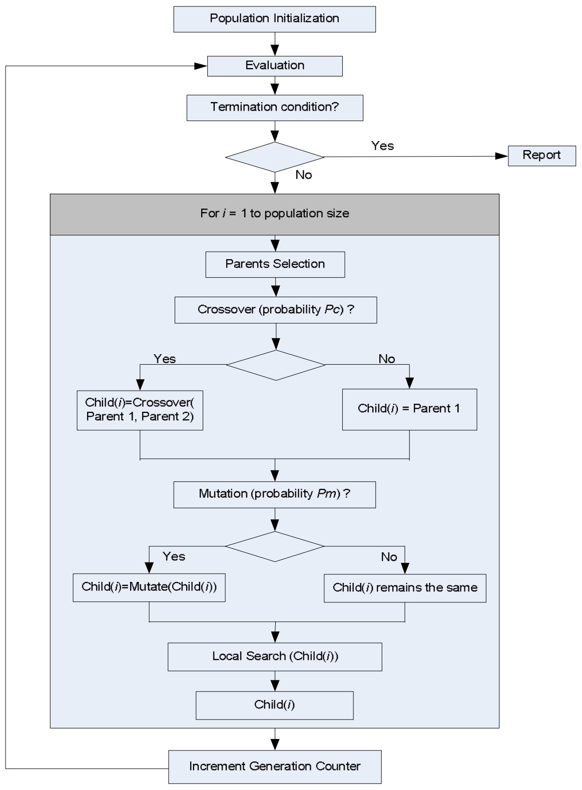
There are two major classes of Hamiltonian cycle algorithms: polynomial time heuristic algorithms and backtracking algorithm. The backtracking algorithm is a standard approach in solving Hamiltonian cycle problem. However, search by backtracking is computationally intractable. Heuristic algorithms on the other hand are fast, running in linear or low-order polynomial time by using general rule of thumbs and/or randomization. Heuristic HCP algorithms include Ant Colony Optimization [13,14], Neural Network [15], Tabu Search, Simulated Annealing, Undirected Hamiltonian Cycle algorithm (UHC, formulated by Angluin and Valiant) [16], HAM (Hamiltonian heuristic algorithm formulated by Bollobas, Fenner, and Frieze)[17, 18] and so on.

One practical use of HCP is the site visitation sequencing problem for an Unmanned Reconnaissance Aerial Vehicle (URAV). URAVs can cover a large set of spatially distributed sites that are present within its site envelope. Given *S* being the set of reconnaissance sites that the URAV should survey, it is a general requirement that a URAV visits the useful vicinity of each site exactly once, and lands back safely at the take-off zone or one of the other benign landing zones that are within its flight envelope. If the URAV returns to the take-off site, the problem becomes one of finding the Hamiltonian cycle. If no Hamiltonian cycle is uncovered, the solution becomes the longest path {LP} between take-off and landing sites. An acceptable solution to this problem is essential in choosing "minimal risk" waypoints for the URAV flight path [19-21].

Our focus is to develop a memetic algorithm (MA) for finding the longest path. The solution of the site visitation sequencing problem for a single URAV can be represented by an integer permutation. This is suitable for ADEP which was designed to handle permutation string structure. The ADEP environment automatically generates MA source codes using the specified parameters. The outcome is a set of platform-independent C++ source codes that can be compiled into a console application program. The source codes only need minor modification to be embedded into Windows application. The domain specific knowledge pertaining to the problem is incorporated into the MA application via the evaluation on the fitness function codes.

IV. Memetic Algorithm

To develop HCP algorithm using ADEP, the PSE was first used to generate the MA code framework, which was then configured to find the longest path by incorporating the appropriate objective function of the MA algorithm. The generic MA has a framework as shown in Figure 2. After the population initialization, the population is evaluated and the termination condition is checked. If the termination condition is not met, the population is allowed to evolve further. In each generation, offspring are produced from the parent population using several genetic operators (crossover, mutation, and local search) and the population is replaced with the offspring. When the termination condition is met, the chromosome with the highest fitness value is returned which contains the sequence of vertices on the graph. This sequence of vertices represents the longest path.



Population Initialization
Evaluation
Termination condition?
Yes
Report
No
For i = 1 to population size
Parents Selection
Crossover (probability Pc) ?
Yes
No
Child(i)=Crossover(Parent 1, Parent 2)
Child(i) = Parent 1
Mutation (probability Pm) ?
Yes
No
Child(i)=Mutate(Child(i))
Child(i) remains the same
Local Search (Child(i))
Child(i)
Increment Generation Counter

Figure 2: Generic framework of the Memetic Algorithm

The ADEP-generated memetic algorithm represents the vertices of the graph by integer permutation string [22, 23]. In the chromosome, each gene represents a vertex on the graph. The position within the string dictates the visitation sequence of the graph's vertices. In the population evaluation, the objective/fitness function calculates the length of the longest substring/path in the chromosome by checking the connectivity between adjacent genes (i.e. the successive edges of the traversed vertices along the visitaton sequence) in linear time. Since the goal is to maximize the length of the longest path, the fitness value of the chromosome is set to the length of the longest path in the chromosome. The local searcher as configured by the automatic optimizer of the ADEP environment is a two-gene swap method which repeatedly swaps two genes in the chromosome to vary the topology of the search space. Other configured local methods include local methods such as single chromosome replacement over the parent population.

V. COMPUTER SIMULATION

The ADEP-generated MA was tested on a dataset comprising of 14 sparse graphs with edge densities (Ed) ranging from 0.0398 to 0.1947. All 14 graphs in their graphic and adjacency matrix representations are available online [24]. Out of the 14 graphs, 7 are Hamiltonian while the remaining 7 are non-Hamiltonian. There are 10 graphs derived through special modifications to the Tutte graph [25]. Table 1 shows the experimental results of MA simulation on these graphs with the last column of the table showing the maximum known path length. Each graph is labeled as *G(m,n)*. The * symbol indicates that the graph is Hamiltonian. The results were obtained in MS Windows XP that ran on an OPTIPLEX GX520 PC Desktop with 3.39 GHz Intel Pentium 4 CPU and 512 MB RAM. Simulations for each algorithm involve 10 independent runs.

TABLE 1
Simulation results of MA for solving the different HCPs

Graphs	Avg. CPU time (s)	Avg. longest path	Longest path obtained	Max. known path length
G(15,20)	0.08	14.0	14	14
G(20,30)*	0.11	20.0	20	20
G(20,37)*	0.10	20.0	20	20
G(46,69)	1.43	45.0	46	46
G(46,71)*	1.15	45.3	46	46
G(46,93)*	1.53	45.7	46	46
G(49,96)	0.98	48.0	48	48
G(49,142)	0.96	48.0	48	48
G(100,197)	4.29	95.5	98	99
G(100,197)*	5.29	97.8	100	100
G(100,203)*	4.17	96.6	99	100
G(100,207)	6.88	97.4	98	99
G(100,217)	9.75	97.9	99	99
G(100,217)*	6.23	98.2	99	100

In Table 1, it is shown that, for problems with size no more than 48, which is considered "very large" for many real applications, it is basically not a problem for the memetic algorithm to uncover the longest known path or Hamiltonian cycle within 1.6 seconds of computational time. For larger problems with 100 nodes, the resulting MA showed respectable performance in uncovering solutions that are very close to the longest path or Hamiltonian cycle, and the computational time is less than 10 seconds. For all the 14 graphs tested, the solutions with maximal length or {maximal_length-1} were found.

To evaluate the performance of the MA algorithm, it is essential to do a comparison on the performance to other HCP algorithms. Here the same graph dataset was tested using standard backtracking algorithm and other standard heuristic algorithms. Five other types of HCP algorithms were developed and tested on the same set of sparse graphs. The HCP algorithms include standard backtracking algorithm, Tabu Search, Ant System and Simulated Annealing algorithm.

Since the ADEP-generated MA algorithms was developed with minimal modification in the generated codes, for a fair comparison in performance, the HCP algorithms used should also have a form as "generic" as possible instead of being fine-tuned to solve the problem. The 4 testing algorithms were therefore developed in such a way that they all have similar structures and share a common framework that include common classes such as CProblem, CRandom, and so on. This was done to minimize the variance in performance of the algorithms.

The standard backtracking algorithm was derived by creating a number of Tracer objects which move from a starting vertex in the graph and proceed to the next vertex by checking the connectivity with the rest of the vertices. When no connectivity is found, the Tracer object backtracks to the previous move and look for a different connected vertex. The algorithm iterates until it finds a Hamiltonian cycle and if no Hamiltonian cycle is found in the graph, it returns the longest path as the solution.

The Tabu Search and Simulated Annealing algorithm were designed such that they share the same objective function as that of the ADEP genetic algorithm. Since these two algorithms were neighborhood search algorithms, they were designed to have similar class structure and methods. Both neighborhood search algorithms deploy two-gene swap technique and obtain the new neighborhood solution by selecting the best in the neighborhood of the current solution. The two algorithms were run for 10000 iterations before termination.

In the Tabu search algorithm, the recency Tabu table incorporates a Tabu tenure of randomly variable size for diversification and the aspiration function bypasses the Tabu for Tabu move steps that outperform the current solution (i.e. the solution has a higher objective value than the current solution).

The Simulated Annealing algorithm sets the initial annealing temperature to 10. The annealing temperature is then decreased by multiplying it with a delta value of 0.99 per iteration. The entropy function has the form of $\exp\left(-\frac{E}{T}\right)$, where E is the energy or objective value of the solution and T is the annealing temperature. The best solution obtained in the neighborhood of the current solution is selected to replace the current solution if the probability is greater than $\exp\left(-\frac{E}{T}\right)$ or if the annealing temperature is lower than 10E-4 and the best neighborhood solution has a higher objective value.

The Ant System algorithm was designed to have 200 ants uniformly distributed among the vertices of the graph. The ants will traverse the graph and find the longest non-repeated path it can reach before it becomes inactive. After all the ants become inactive, the pheromone table is updated by pheromone evaporation followed by having each ant drop pheromone on the path they visit. The amount of pheromone is weighted by the length of the path the ant traverses. The ant determines the next vertex to move based on the following pseudo-random state transition rule [26]:

$$p\left(c_i \mid s^p\right)=\frac{\tau_i^{\alpha} \bullet \eta_i^{\beta}}{\sum_{c_i \in N\left(s^p\right)} \tau_i^{\alpha} \bullet \eta_i^{\beta}}, \forall c_i \in N\left(s^p\right) \quad (2)$$

where τ_i and η_i are respectively the pheromone value and the heuristic value associated with the vertex c_i. In the ACO algorithm developed, α is set to 1 and β is set to 5. These values are the weights of the pheromone value and the heuristic value respectively. s^p represents all the vertices that have not yet been visited. The evaporation rate of the algorithm is set to 0.5.

Table 2 presents the experimental results of the ADEP-generated memetic algorithm and the 4 comparison algorithms in uncovering the longest path in terms of wall clock time. The values in the table are in the form *a (b, #)* where *a* is the longest path uncovered and *b* is the wall clock time. The # sign indicates the longest path uncovered by the algorithm in reaching the maximum known path length of the particular HCP.

Figures 3, 4 show the plot of the performance of the ADEP generated memetic algorithm and the 4 comparison algorithms in uncovering the longest path obtained and the time spent. Figure 3 contains the comparisons for HCPs of G(15, 20), G(20, 30)*, G(20, 37)*. Figure 4 shows the performances of the algorithms on the HCPs with 100 vertices. Due to the brute-force nature of the backtracking algorithm, only the simulation results for the first 6 sets of graphs were able to be obtained in reasonable amount of time. Since the rest of the simulation results for backtracking algorithm took too long a time to be generated, they are not included here for comparison.

Figure 5 shows the performances of the 5 HCP algorithms in terms of the wall clock time.

TABLE 2
Performances of the four algorithms in uncovering the longest path in terms of wall clock time

Graphs	Backtracking	Simulated Annealing	Tabu Search	Ant Colony Optimization
G(15,20)	14 (0.08s, #)	14 (0.062s, #)	14 (0.046s, #)	14 (0.015s, #)
G(20,30)*	20 (0.05s, #)	19 (0.156s)	20 (4.359s, #)	19 (0.015s)
G(20,37)*	20 (0.05s, #)	18 (0.187s)	20 (1.297s, #)	19 (0.015s)
G(46,69)	46 (9.32s, #)	36 (3.719s)	40 (4.703s)	44 (25.843s)
G(46,71)*	46 (0.08s, #)	29 (3.547s)	40 (16.5s)	45 (28s)
G(46,93)*	46 (110.0s, #)	27 (3.453s)	41 (3.61s)	45 (39.109s)
G(49,96)	NA	34 (5.078s)	45 (7.25s)	48 (40.593s, #)
G(49,142)	NA	44 (3.047s)	39 (7.219s)	48 (0.344s, #)
G(100,197)	NA	33 (38.36s)	78 (141.485s)	97 (75.343s)
G(100,197)*	NA	31 (51.25s)	23 (35.828s)	96 (1655.672s)
G(100,203)*	NA	57 (80.953s)	34 (51.958s)	93 (154.75s)
G(100,207)	NA	49 (76.25s)	71 (120.218s)	96 (1064.906s)
G(100,217)	NA	63 (86.844s)	49 (88.324s)	96 (1454.86s)
G(100,217)*	NA	56 (84.359s)	34 (75.61s)	89 (453.906s)

* indicates that the graph has known Hamiltonian cycle

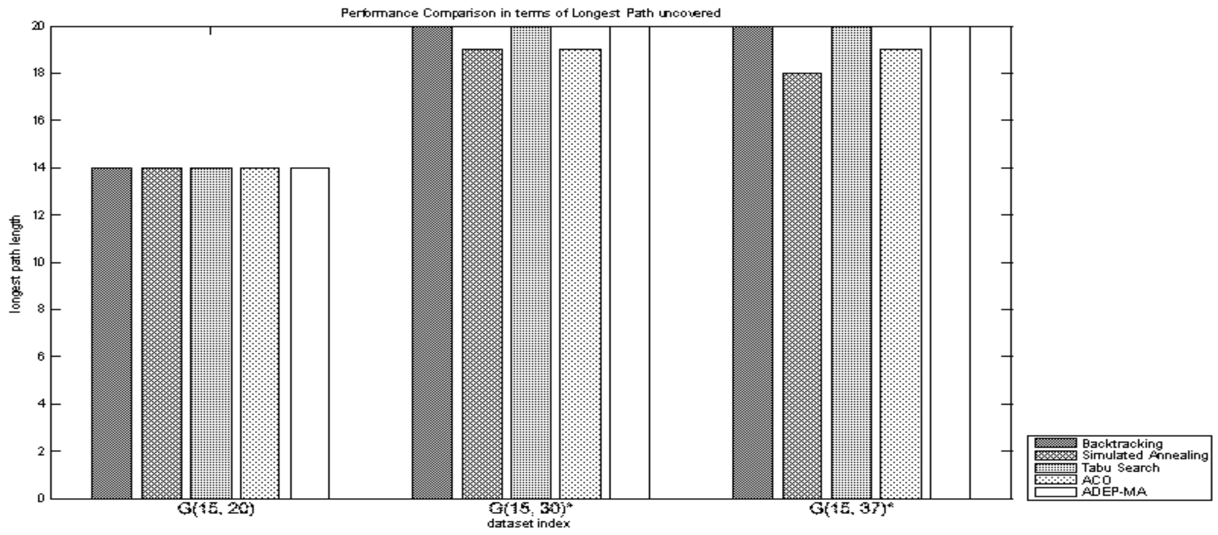


Figure 3: Longest Path uncovered by the five algorithms on HCP with 15 and 20 vertices

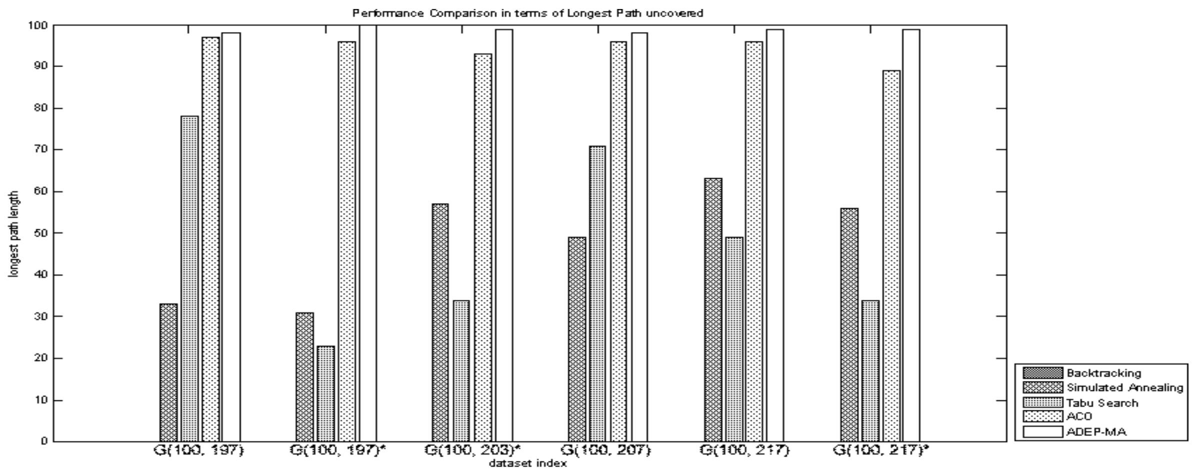


Figure 4: Longest Path uncovered by the five algorithms on graphs with 100 vertices

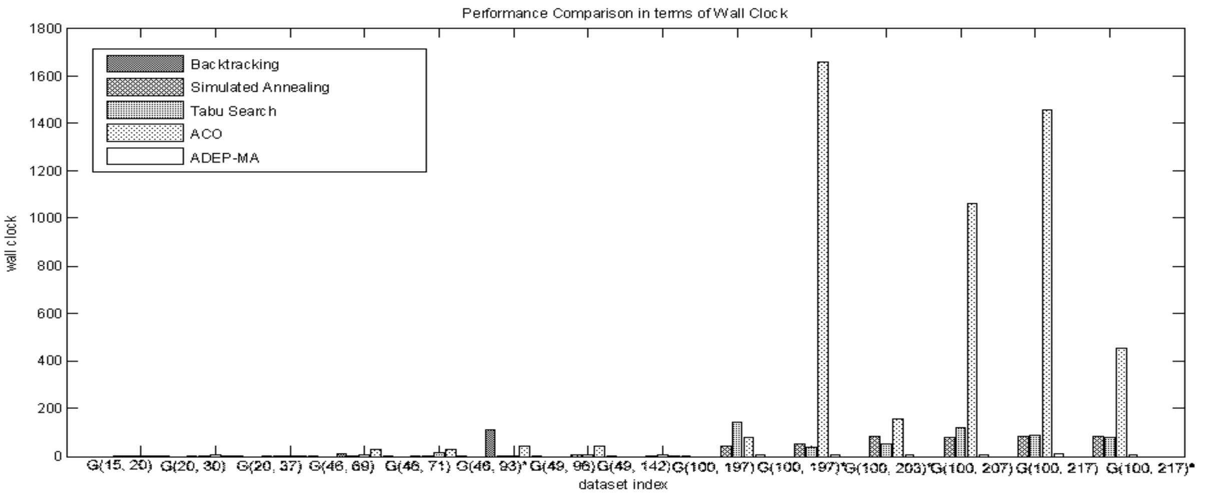


Figure 5: Wall clock time spent by the five different algorithms in uncovering the longest paths on the set of graphs

Figure 3 shows the performance of the 5 algorithms in terms of the longest path uncovered on the graphs with 15 and 20 vertices. In this figure, the number of vertices in the graphs is relatively low, the 5 algorithms have similar performance in that they can uncovered the max-known or near-max-known path length of these graphs

Figure 4 shows the simulation result of the algorithms on the graphs with 100 vertices. The results of the backtracking algorithms are not included in the figure since the problems are not solvable within reasonable time. It is seen in this figure that the performance of the Simulated Annealing and Tabu Search algorithms were lagging behind in terms of the longest path uncovered. ACO and MA are still close in performance in terms of uncovering the longest path.

Figure 5 shows that ACO algorithms took the longest time to uncover the longest path length whereas ADEP-generated MA algorithms was the fastest in uncovering the longest path among the 5 HCP algorithm in terms of wall clock time.

The simulation results of the 5 algorithms show that the ACO and ADEP-generated MA algorithms have the best performances among the 5 algorithms compared in terms of the longest paths uncovered. However, it is seen in Table 1 and Table 2 that ADEP-generated MA algorithm outperformed ACO algorithm in terms of wall clock time. Moreover, the ADEP-generated MA algorithm was better at uncovering the maximum path length than ACO on these 14 sparse graphs as can be seen in Table 3. In addition, ADEP-generated algorithms took a shorter time in uncovering the longest path than the ACO algorithm.

VI. CONCLUSION

In this paper, a MA for HCP was configured in the ADEP environment. The algorithm was tested against 14 sparse graphs with edge densities (Ed) ranging from 0.0398 to 0.1947 and number of vertices ranging from 15 to 100, the results were compared with that of the backtracking and 3 other heuristic algorithms. It was shown that the MA took less than 10 seconds of computation time and an average 1.6 seconds average time to come up with optimal or near-optimal solution. When compared with other HCP algorithms such as backtracking, Simulated Annealing, Tabu search and ACO, the ADEP generated algorithm showed superior performance in terms of both the longest path uncovered and the wall clock time to uncover the path.

The configuration and development of the HCP MA algorithm in ADEP and subsequent computer simulation demonstrate that the ADEP dramatically reduces the time required in developing and formulating the competitive and optimal MA algorithm for the real-world problems. The solution methodology was shown to deliver high level of performance, competitive and even superior in performance to the handcrafted algorithms. The demonstration on development and the subsequent computer simulation of MA algorithm configured in the PSE open up avenue for PSE to address the optimization needs of other NP-hard problems.

ACKNOWLEDGMENTS

The authors acknowledge the funding support of Singapore Technologies Engineering Pte Ltd in this work.

REFERENCES

[1] Pablo Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms", Caltech Concurrent Computation Program 158-79, California Institute of Technology, 1989.

[2] Y. S. Ong and A.J. Keane, "A domain knowledge based search advisor for design problem solving environments", Engineering Applications of Artificial Intelligence, 2002, Vol. 15, No. 1, pp. 105-116.

[3] Z. Z. Zhou, Y. S. Ong, P. B. Nair, A. J. Keane and K. Y. Lum, "Combining Global and Local Surrogate Models to Accelerate Evolutionary Optimization", IEEE Transactions On Systems, Man and Cybernetics - Part C, Vol. 37, No. 1, Jan. 2007, pp. 66-76.

[4] Z. Zhu, Y. S. Ong and M. Dash, "Wrapper-Filter Feature Selection Algorithm Using A Memetic Framework", IEEE Transactions On Systems, Man and Cybernetics - Part B, vol. 37, no. 1, Feb 2007.

[5] J. Tang, M. H. Lim and Y. S. Ong, "Diversity-Adaptive Parallel Memetic Algorithm for Solving Large Scale Combinatorial Optimization Problems", Soft Computing Journal, Vol. 11, No. 9, pp. 873-888, July 2007.

[6] Y. S. Ong and A. J. Keane, Meta-Lamarckian Learning n Memetic Algorithm, IEEE Transactions On Evolutionary Computation, Vol. 8, No. 2, pp. 99-110, April 2004.

[7] Y. S. Ong, M. H. Lim, N. Zhu and K. W. Wong, "Classification of Adaptive Memetic Algorithms: A Comparative Study, IEEE Transactions On Systems, Man and Cybernetics – Part B", Vol. 36, No. 1, pp. 141-152, February 2006.

[8] Xu Yilaing, PhD Thesis "Meta-Heuristic Algorithm Development for Combinatorial Optimization within an Integrated Problem Solving Environment", NTU, 2007

[9] W. T. Tutte, "On Hamiltonian Circuits.," *Journal of the London Mathematical Society*, vol. 21, pp. 98-101, 1946.

[10] Hamilton, William Rowan, "*Account of the Icosian Calculus*". Proceedings of the Royal Irish Academy, 6 1858

[11] L. Posa. "Hamiltonian Circuit in random graphs". Discrete Mathematics 14:359-364, 1976.

[12] Eli Shamir. "How many edges make a graph Hamiltonian? "Combinatorica, 3(1):123-131, 1983.

[13] Dorigo, V. Maniezzo, A. Colorni, "The Ant System: Optimization by a Colony of Cooperating Agents," IEEE Trans. on Systems, Man, and Cybemetics-Part B vol. 26 (1996), 29-41.

[14] Wagner, I.A.; Bruckstein, A.M, "Hamiltonian(t)-an ant-inspired heuristic for recognizing Hamiltonian graphs", Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, Vol.2, Iss., 1999 Pages:-1469 Vol. 2

[15] Mehta, S.; Fulop, L. "A neural algorithm to solve the Hamiltonian cycle problem" Neural Networks, 1990., 1990 IJCNN International Joint Conference on, Vol., Iss., 17-21 Jun 1990 Pages:843-849 vol.3

[16] D. Angluin and L. G. Valiant. "Fast probabilistic algorithms for Hamiltonian circuits". System Sci. 18(2):155-193, 1979.

[17] Basil Vendegriend, "Finding Hamiltonian cylces: Algorithms, graphs and performance", Department of Computing Science, Edmonton, Alberta, Spring 1998.

[18] B. Bollobas, T. I. Fenner, and A. M. Frieze. "An algorithm for finding Hamiltonian paths and cycles in random graphs". Combinatoria, 7(4):327-341, 1987.

[19] M. Jun and R. D. Andrea, "Path Planning for UAVs in Uncertain and Adversarial Environments", Cooperative Control: Models, Applications and Algorithms: Kluwer, 2003.

[20] A. Richards, J. Bellingham, M. Tillerson, and J. How, "Co-ordination and Control of Multiple UAVs," AIAA Guidance, Navigation and Control Conference, Monterey, CA, 2002.

[21] R. W. Beard, T. W. McLain, and M. Goodrich, "Coordinated Target Assignment and Intercept for Unmanned Aerial Vehicles," IEEE Intl. Conf. on Robotics and Automation, Washington DC, 2002.

[22] K. K. Lim, Y. S. Ong, M. H. Lim, A. Argarwal, "Hybrid Ant Colony Algorithm for Path Planning in Sparse Graphs", Soft Computing, In Review process.

[23] A. Agarwal, M. H. Lim, Y. L. Xu and Y. S. Ong. "Evolutionary Graph Mining for the Discovery of Site Visitation Sequences for a Single URAV". The second International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore, 2003

[24] "Graph Dataset": http://www.ntu.edu.sg/home/asysong/sparse/default.htm

[25] W. T. Tutte, "On Hamiltonian Circuits.," *Journal of the London Mathematical Society*, vol. 21, pp. 98-101, 1946.

[26] M. Tim Jones, "AI Application Programming", Charles River Media, 2003