Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 46 (2015) 388 – 395

Procedia Computer Science

International Conference on Information and Communication Technologies (ICICT 2014)

Metaheuristic Algorithms and Polynomial Turing Reductions: A Case Study Based on Ant Colony Optimization

Anandkumar Prakasam[a,*], Nickolas Savarimuthu[b]

[a-b]*Department of Computer Applications, National Institute of Technology, Trichy.*

Nowadays, there is an increasing dependence on metaheuristic algorithms for solving combinatorial optimization problems. This paper discusses various metaheuristic algorithms, their similarities and differences and how Ant Colony Optimization algorithm is found to be much more suitable for providing a generic implementation. We start with the solution for Travelling Salesman Problem using Ant Colony Optimization (ACO) and show how Polynomial Turing Reduction helps us solve Job Shop Scheduling and Knapsack Problems without making considerable changes in the implementation. The probabilistic nature of metaheuristic algorithms, especially ACO helps us to a greater extent in avoiding parameter fine-tuning. Through Sensitivity analysis we find that ACO exhibits better resilience to changes in parameter values in comparison to other metaheuristic algorithms.

Keywords: Ant Colony Optimization; Turing Reductions; Metaheuristic Algorithms; Travelling Salesman Problem; Job Shop Scheduling; Knapsack Problem;

1. Introduction

The necessary requirement for most of the applications is to provide appropriate results within stipulated time. Due to the increase in the amount of data for processing, guaranteeing the time requirement becomes infeasible

* Anandkumar Prakasam. Tel.: +91-9790636324
E-mail address: root.anand@gmail.com

doi:10.1016/j.procs.2015.02.035

when the application is processed with statistical methods. Hence metaheuristic algorithms came into view. Metaheuristics are a class of algorithms that uses lower level heuristics or procedures to solve a problem by providing a sufficiently good solution within acceptable time with imperfect or limited information[8]. These algorithms usually employ iterative methods and heuristics to generate a solution using stochastic optimization techniques, which will often be a satisfactory solution for the problem[9]. These algorithms are mostly nature inspired and can be solved within the specified time. The trade-off occurs by compromising the result's accuracy with a minimal error value. Another advantage of metaheuristic algorithms is that it is not problem dependent[10]. It is a generic implementation, which can be used for solving almost all varieties of problems. But the current applications using metaheuristic algorithms do not follow this standard. Many implementations have been proposed, that claims to solve problems using modified metaheuristic algorithms. Even though these methods assert on providing better results, they cannot be considered as a contribution to the field of metaheuristic algorithms, since the basic definition of a metaheuristic algorithm states that the method is not problem specific, rather a generic solution that can be fitted to all the NP Complete/ NP Hard problems. Further, these so-called modified methods depends too much on fine tuning of the parameters and they tend to be more deterministic trying to reason every probabilistic behaviour. So in our view the very act of perfecting a metaheuristic algorithm is wrong. For the current study, we have shortlisted three metaheuristic techniques; Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and the Firefly Algorithm.

2. Literature Survey

The process of discovery of solutions (food source) by ants was initially proposed by J. L. Deneubourg et al. in 1983[2]. According to Deneubourg, the behaviour of ants in discovering food is highly probabilistic and is prone to errors. It also states that this error prone system serves as the basis for sustainability of the system. If all the ants tend to follow the same path, then the solution has a high probability of getting struck in local optima, and exploration is curtailed. According to the nature, an inbuilt error is introduced such that all the ants do not follow the same pre-determined path, instead some ants tend to diverge, which makes them explore other areas to obtain diverge results. An optimization approach that uses the ant analogy was proposed by M. Dorigo in 1992[3]. The method proposed by M. Dorigo is a combination of distributed computation, positive feedback and constructive greed heuristic[11]. It provides efficient solutions within the stipulated time, and also provides effective exploration constraints such that the entire search space is explored in an efficient manner. Dorigo's basic model proposes three algorithm variations, *Ant Density, Ant Quantity* and *Ant cycle.* Later in 1997[11] proposed ACO for Travelling Salesman Problem using a 3-opt local search, which is found to be effective in overcoming the local optima and convergence issues. Later in 2000[4] provided the first evidence of convergence for an algorithm of ant colonies.Some common extensions to the ACO are the Elitist Ant System[14] where the global best deposits pheromone on every iteration, the Max-Min Ant Systems[15] where pheromone deposits are modified, Rank based Ant System[16] where solution ranking is performed on the basis of path lengths, Continuous Orthogonal Ant Colony[17] which provides enhanced global search capability and Recursive Ant Colony Optimization[18] using nested ant systems.

The initial proposal regarding swarm intelligence[5] was considering the nearest neighbour velocity matching and craziness. Every particle in the population is assigned an *(x, y)* positions and their corresponding axis velocities. This velocity value determines the speed and direction of the particles. An enhancement to this approach[5] is the cornfield vector. Each agent now maintains the best solution explored by it called the *pbest[](which is an array)* and its positions. Additionally, the agents also have knowledge about the globally best position *gbest* that has been found by the flock. The proposal by James Kennedy[5] was improved by the usage of an inertia weight *w* that plays a role of balancing the global search and local search[6].

Inspired by the flashing behaviour of fireflies Xin-She Yang in 2008[7] formulated the firefly metaheuristic algorithm which is based on three fundamental assumptions regarding fireflies. It assumes that all fireflies are unisexual and hence get attracted towards each other irrespective of their sex, and the attractiveness is directly proportional to the brightness and in turn inversely proportional to distance. If there is no brighter fly then the movement is governed by randomness. Here the variation of light intensity and the formulation of attractiveness plays the key role. Even though ACO, Firefly and PSO are metaheuristic and can be used for performing similar functionalities, the operational domain of ACO is discrete, while that of PSO and Firefly is continuous. ACO starts its dispersion from a starting node and finds the best solution to that node, while PSO and Firefly disperses the particles onto points that are possibly the solutions and tries to move all the particles towards the best known solution. The movement of particles in PSO and Firefly is smooth, while that of ACO is discrete. The ant movement in ACO is basically a jump from one solution point to other. The ant movement in ACO is probabilistic. This behaviour takes the trail intensity into consideration in determining the destination point. Hence the direction of traversal is completely determined by the destination of traversal. Due to the continuous moving behaviour of the particles in PSO, it takes into account two solutions, the global best and the local best to determine its direction of travel. The velocity of travel is determined by these points along with some parameters generated in random. Every firefly moves towards another firefly with a better brightness. Fireflies do not maintain any data or share any information (other than their own brightness values) with other fireflies. In both the algorithms, the velocity of movement is directly proportional to the solution's fitness. The movement of particles in PSO is regular and towards the local and global best, while firefly's exhibit random movement. The random movement is due to the fact that comparison of the fitness function is performed on fly to fly basis. Due to the computation of solutions from the dispersed agents, the degree of getting struck in local optima is high. Node selection is made rather than point selection; hence there is no divergence from the course for an ant, while a particle in PSO can change its course after every iteration depending on the current local and global best, and in Firefly, it depends on the flies within the range of the current firefly. PSO works on an assumption that all best values lie around a particular region; hence it has a high probability of getting struck in the local optima if the actual best value is present in the opposite direction of traversal. ACO and Firefly defines their fitness function as the distance between the nodes, while the fitness function in PSO can be defined by the user. Hence PSO can be used to solve multi objective optimization problems, while some modifications are necessary to convert ACO and Firefly to solve multiple objectives based problems. All the algorithms are capable of working with noisy or irregular dynamic data. In order for PSO and Firefly to work on a discrete domain, corresponding mapping should be performed by the user, which can act as an additional overhead. Many similarities exist in PSO and Firefly Algorithm (FA), as well as the fact that by changing the absorption co-efficient (γ) of the Firefly Algorithm to zero, the Firefly Algorithm behaves exactly like the Standard PSO algorithm makes their functionality look very similar. PSO and Firefly do not exhibit probabilistic behaviour. Randomization is incorporated during the movement of particles. While Firefly Algorithm employs a Gaussian distribution function to determine its random parameters, PSO does not employ any. In ACO, the initial distribution of functions is random, while during traversal, uniform distribution function is used for determining the random parameters. PSO requires some parameters to be set by the user and it does not contain any fitness function, hence algorithms generated by employing PSO are considered to be problem specific.

On profound analysis it has been found that ACO exhibits various traits that makes it generic and accurate. It performs exceptionally well by being flexible to Hybridization, and is also basically parallel in nature and they exhibit the ability to handle dynamic data sets efficiently. Its ability to define the stopping criterion it powerful and the complexity of the fitness function is moderate. Hence ACO has been chosen as the appropriate candidate for our study.

3. Stochastic Nature and Generic Implementations : The Need

The advantage of using a generic code is that it can be used for any type of problem, without much change in the basic logic. It also tends to provide reliable results irrespective of the problem being solved. The problem in creating a problem specific implementation of ACO is that the basic working model tends to be changed and hence the code reliability is lost and sometimes the change becomes intense and will lead to a code that cannot be called as ACO. Further deriving such algorithms is opposed to the basic notion of the Metaheuristics, which states that a metaheuristic algorithm should not be bound to a specific problem and it has to be designed in such a manner that it is usable in a variety of problem situations[1].

Hence, in order to use ACO for various varieties of problems, the input of the corresponding problems should be converted to a format that can be used by ACO, i.e. the input should be converted to a graph with nodes and weighted edges. TSP based graph for ACO is usually bidirectional, while other implementations like JSSP or Knapsack requires some modifications to be made in the graph. The drawback of using such an approach is that it adds up an additional pre-processing overhead for the system. But this overhead completely depends upon the nature of the problem in hand. If the problem is highly dynamic in nature, it will require frequent modification of the graph, but if it is of stable nature or if it has very low variations, ACO can be guaranteed to provide efficient results.

Since ACO solves TSP in its implementation, while PSO and Firefly algorithms are not problem specific. Hence we consider the basic problem structure for ACO, PSO and Firefly algorithms to be a graph and shortest path is obtained from it. The problems of Job Shop Scheduling and Knapsack are converted into graphs with edge weights and are solved using the generic algorithm rather than providing a problem specific algorithm to solve the problems.

4. Polynomial Turing Reductions

In computational complexity theory, NP-Hard or Non Deterministic Polynomial Time Hard, is a class of problems that is considered to be as hard as the hardest problems in NP (informally). Unlike Polynomial Many One Reduction which is used for reducing NP-Complete problems, we use Polynomial Turing Reduction for reduction between NP-Hard problems. "*A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using a polynomial number of calls to a subroutine for problem B, and polynomial time outside of those subroutine calls. Polynomial-time Turing reductions are also known as Cook reductions, named after Stephen Cook*".

4.1. Job Shop Scheduling Problem to Travelling Salesman Problem

Problem Definition

"Job shop scheduling is an NP-Hard problem, that contains a list of jobs and machines, and a constraint placed on the order of execution of these jobs. The requirement is to find the sequence of execution of all jobs while minimizing the make-span".

The Job Shop Problem can contain multiple operations in a single job, or can be atomic. The basic intention for selecting a job shop scheduling problem is that it can be mapped to most of the scheduling problems.
JSSP uses a disjunctive graph for representing data. This is a normal graph but with a combination of directed and undirected edges. A disjunctive graph is a directed graph$G = (V, C, D)$, where *V* represents the vertices, *C* represents conjunctive edges and *D* represents the disjunctive edges. The conjunctive edges are directed, while the disjunctive edges are undirected, signifying a two way connection.

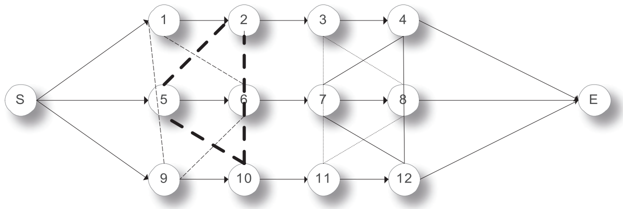

Fig. 1. Sample Disjunctive graph with 3 jobs and 4 machines.

Every job is represented in a single row and a directed arrow is added to each sub job in a row, to signify its sequential nature. E.g. The first row (Fig. 1) shows the sequence of Job I. 1, 2, 3 and 4 are the sub jobs that are to be completed in a sequence. I.e. sub job 2 cannot be performed without finishing sub job 1. So they are represented by directed arrows. Similarly 5, 6, 7 and 8 are the sub jobs of Job II, and 9, 10, 11 and 12 are the sub jobs of Job III.
Sub jobs running on the same machine are connected with disjunctive arcs (lines without arrow heads), i.e. 1, 6 and 9 are run on the same machine (Machine 1) and 2, 5 and 10 are run on the same machine (Machine 2) (Fig. 1). Similarly disjunctive arcs are added for other jobs (Fig. 1). Apart from the nodes defined from the dataset, start (S) and end (E) nodes are added to the graph. These nodes represent the beginning and the end points of the graph. Weight of an edge corresponds to the time taken by the originating node to complete the job. The job of ACO is to convert this graph into a directed graph by converting the disjunctive arcs to directed ones. This makes up the pre-processing stage of converting the input of the job shop scheduling to the input of the TSP, after which running the

TSP will ultimately lead to solving the Job Shop problem. The algorithmic mapping of TSP with Job Shop Scheduling is discussed below

	Job Shop		***TSP***
1.	*Initialize m ants and n cities(jobs)*	*1.*	*Initialize m ants and n cities*
2.	*S contains the list of jobs that are to be processed next (All initial jobs connected to S in the figure i.e., Elements not in TabuList)*	*2.*	*S contains the list of cities to be visited next (All unvisited cities i.e., Elements not in TabuList)*
3.	*No of machines is defined by the problem*	*3.*	*No of machines is set to 1*
4.	*Repeat until TabuList is full*	*4.*	*Repeat until TabuList is full*
4.1.	*Repeat for each ant*	*4.1.*	*Repeat for each ant*
4.1.1.	*Find job j to be performed next with probability P_{ij} (Based on CDF in probability)*	*4.1.1.*	*Find town j to move to with probability P_{ij} (Based on CDF in probability)*
4.1.2.	*If machine for job j is free,*	*4.1.2.*	*If the machine is free,*
4.1.2.1.	*Remove the job from S*	*4.1.2.1.*	*Remove the job from S*
4.1.2.2.	*Add the job to the TabuList*	*4.1.2.2.*	*Add the job to the TabuList*
4.1.2.3.	*Add the time to its corresponding machine.*	*4.1.2.3.*	*Add the distance to its corresponding machine.*
4.1.2.4.	*Add its Conjunctive neighbors to S*		
4.1.3.	*else goto 4.1.1*	*4.1.3.*	*else goto 4.1.1*
4.2.	*Increment time and decrement all machine values*	*4.2.*	*Increment time and decrement the machine value*
5.	*Repeat section 4 until the stopping criterion is reached*	*5.*	*Repeat section 4 until the stopping criterion is reached*

4.2. Knapsack Problem to Travelling Salesman Problem

Problem Definition

"Given a set of items, each with a mass and a value, we need to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible".

A Directed graph $G = (V, E)$ is used for depicting the Knapsack data, where vertices V correspond to the items that can be chosen from the Knapsack and the edges E corresponds to the connection between the vertices. Every node is connected to every other node in the graph. The directions are specifically provided because the weight of an edge from node A to node B is not the same as the weight from node B to A. Weights of edges are added by calculating the profit obtained for 1 unit of the item i.e. $weight\ /\ profit$ from which the edge originates. Hence even though a bidirectional edge exists between two vertices, their weights differ.

The job of ACO is to traverse this graph and determine the largest distance of traversal. In this problem, we also have the size of Knapsack as a constraint; hence the path taken is trimmed according to the maximum limit and the remaining nodes in the path are ignored.

	Knapsack		***TSP***
1.	*Initialize m ants and n cities(Items)*	*1.*	*Initialize m ants and n cities*
2.	*Calculate the edge weights using the formula weight of the originating vertex / profit of the originating vertex*	*2.*	*Calculate the edge weights using the distance formula*
3.	*Repeat until TabuList is full*	*3.*	*Repeat until TabuList is full Repeat for each ant*
3.1.	*Repeat for each ant*		
3.1.1.	*Find job j to be performed next with probability P_{ij} (Based on CDF in probability)*	*3.1.1.*	*Find town j to move to with probability P_{ij} (Based on CDF in probability)*
3.1.1.1.	*Add the job to the TabuList*	*3.1.1.1.*	*Add the job to the TabuList*
4.	*Calculate the profit of each ant by summing up the profit till the sum of weights reach the maximum knapsack capacity*	*4.*	*Calculate the total distance by summing up the distance of all the edges in the path*
5.	*Find the path with the highest profit*	*5.*	*Find the path with the lowest distance*
6.	*Repeat sections 3,4 and 5 till the stopping criterion is reached*	*6.*	*Repeat section 3,4 and 5 until the stopping criterion is reached*

5. Implementations, Experimental Results and Discussion

All the experimental simulations were performed on a Dell Precision T7600 Workstation, with the following configuration: Two Intel Xeon Processor E5-2680, 8 Cores each @ 2.7 GHz, 20 MB Cache, 32 GB ECC RAM in a Windows 7 - 64 bit environment. Implementation was done in C# language using the Visual Studio 2012 development environment. Travelling Salesman Problem was implemented using ACO (Ant Cycle) on the Oliver 30, elion 50, elion 75, kroa 100 datasets, and sensitivity analysis was performed using the obtained results. The results (Fig. 2, Fig. 3, and Fig. 4) depicts that parameters play a very minor role in determining the accuracy of results. Changes in the values of parameters results in the result deviation of 0.02%, which is very negligible and can be tolerated by any real time application for the benefit of the low convergence time offered by ACO.

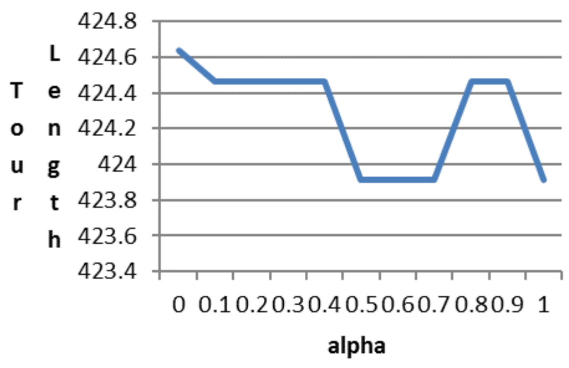

Fig. 2. Sensitivity Analysis TSP (Alpha)

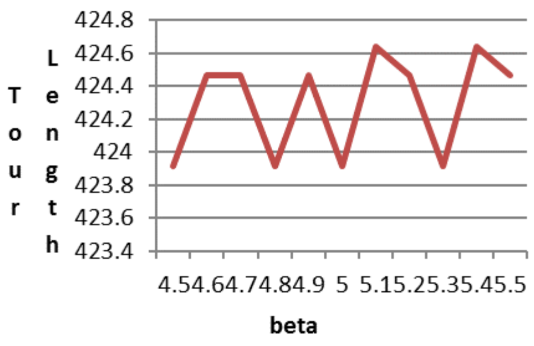

Fig. 3. Sensitivity Analysis TSP (Beta)

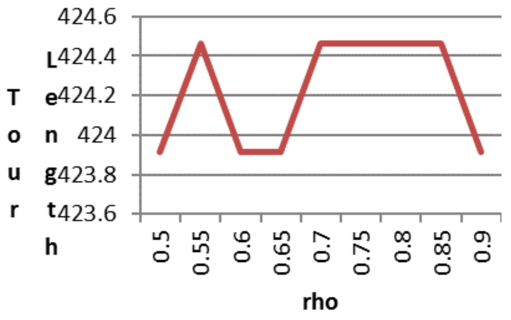

Fig. 4. Sensitivity Analysis TSP (Rho)

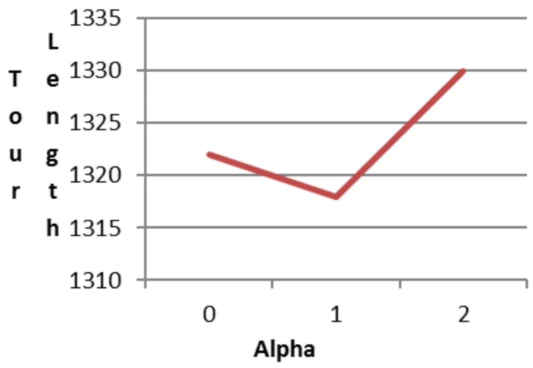

Fig. 5. Sensitivity Analysis JobShop (Alpha)

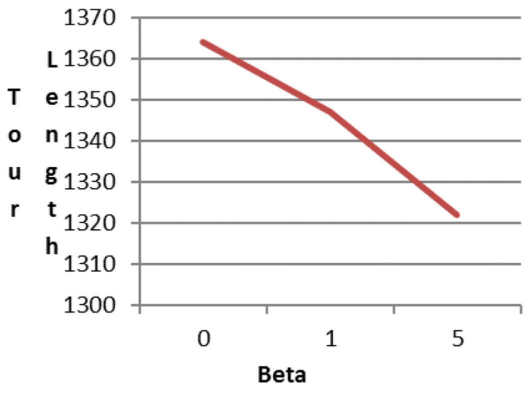

Fig. 6. Sensitivity Analysis Job Shop (Beta)

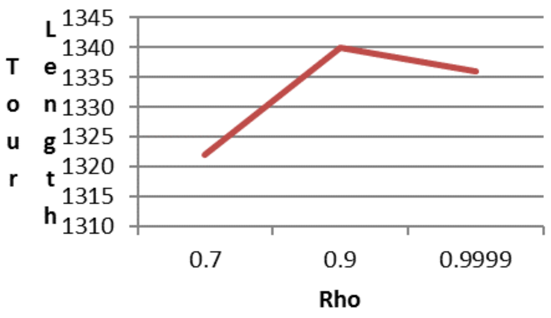

Fig. 7. Sensitivity Analysis Job Shop (Rho)

Fig. 5, Fig. 6 and Fig. 7 depict the results of sensitivity analysis carried out in the Job Shop Scheduling process. A variance of 0.009% has been observed, while Knapsack shows a variance rate of 0.06%, which is very negligible. It can be observed from the Fig. 8 and Fig. 9 that implementing the JobShop Scheduling and Knapsack based on a generic ACO implementation using Turing Reduction provides near optimal results within acceptable time limits. JobShop Algorithm was implemented using the abz5, abz6, abz7, abz8 and abz9 datasets[12] and Knapsack Algorithm was implemented using the Knapsack_01 dataset (P01, P02, P03 and P04 instances)[13].

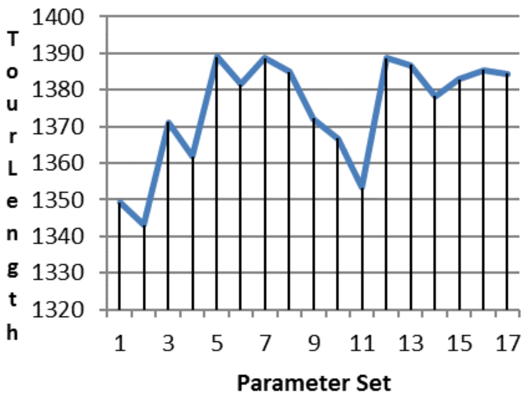

Fig. 8. Job Shop Scheduling (abz5)

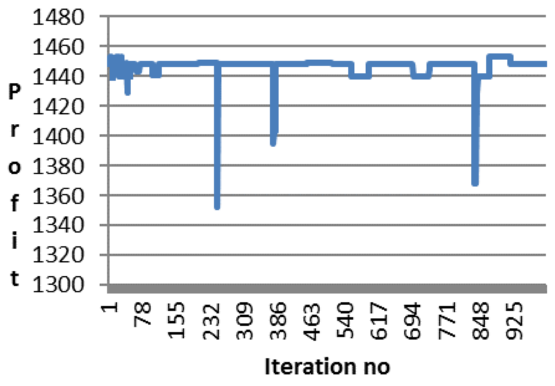

Fig. 9. Knapsack

6. Conclusion and Future Works

This paper provides a generic implementation of ACO to solve NP-Hard problems such as the Travelling Salesman Problem, Job Shop Scheduling Problem and the Knapsack Problem. Results show that the generic implementation provides near optimal results within acceptable time limits and also found to be equivalent to results obtained from problem specific implementations of the same. We have also found that it is possible to solve other related NP-Hard problems using this generic implementation based on Polynomial Turing Reduction. This study has been performed as a part of our process of creating a comprehensive, open source, parallel metaheuristic toolbox. Since our goal is to create libraries for almost all metaheuristic algorithms, providing problem specific implementations is found to be very tedious and is also not feasible. Hence we carried out the initial process of creating a generic implementation of ACO along with Turing reductions for solving other NP-Hard Problems. In future we are planning to extend this library to incorporate other metaheuristic algorithms, including Particle Swarm Optimization, Firefly Algorithm, Bee Colony Algorithm, etc. Parallelization of these algorithms in-order to harvest the potential of multi-core CPUs and many core Graphics Processing Units (GPUs) is the prime motivation of our work. We also consider porting these algorithms to the Hadoop Map Reduce based Distributed Processing Engine in order to exploit their truly distributed nature.

Acknowledgements

The authors would like to acknowledge the infrastructure support provided by the Massively Parallel Programming Laboratory (CUDA Teaching Centre), Department of Computer Applications, National Institute of Technology, Trichy and Dr. Hemalatha Thiagarajan, Professor, Department of Mathematics, NIT, Trichy for providing valuable insights on the simulation of various probability distribution functions.

References

1. Blum, Christian, and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)* 35.3; 2003. 268-308.
2. Deneubourg, Jean-Louis, Jacques M. Pasteels, and Jean-Claude Verhaeghe. Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology*; 1983. 105.2: 259-271.
3. Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. The ant system: An autocatalytic optimizing process. No. 91-016. *Technical report;* 1991.
4. W.J. Gutjahr. A graph-based Ant System and its convergence. *Future Generation Computer Systems*; 2000. Volume 16, pages 873-888.
5. Kennedy, James, and Russell Eberhart. Particle swarm optimization. *Proceedings of IEEE international conference on neural networks;* 1995. Vol. 4. No. 2.
6. Shi, Yuhui, and Russell Eberhart. A modified particle swarm optimizer.*Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. IEEE, 1998.
7. Yang, X. S. Nature-Inspired Metaheuristic Algorithms.*From: Luniver Press*; 2008. ISBN 1-905986-10-6.
8. Kenneth Sorensen. Metaheuristics – the metaphor exposed. *International Transactions of Operations Research*; 2012. Pub Online: Feb 08, DOI: 10.1111/itor.12001 (p).
9. I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*; 1996. 63:513.623.
10. The Metaheuristics Network: *http://www.metaheuristics.net/,* 2000. Visited in January 2003.
11. Dorigo, Marco, and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation*; 1997. IEEE Transactions on 1.1: 53-66.
12. OR Library: *https://files.nyu.edu/jeb21/public/jeb/orlib/files/.* Visited in August 2014.
13. Knapsack 01: *http://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html* Visited in August 2014.
14. Sorin C. Negulescu, Constantin Oprean,Claudiu V. Kifor, Ilie Carabulea..Elitist ant system for route allocation problem. *World Scientific and Engineering Academy and Society (WSEAS) Stevens Point*, Wisconsin, USA ; 2008.
15. T. Stützle et H.H. Hoos. Max Min Ant System. *Future Generation Computer Systems*; 2000. Volume 16, p.889-914.
16. Bernd Bullnheimer Richard F. Hartl, Christine Strau. A New Rank Based Version of the Ant System - A Computational Study. *Working*; 1997. Paper No. 1.
17. Xiao-Min Hu, Jun Zhang, Yun Li. Orthogonal Methods Based Ant Colony Search for Solving Continuous Optimization Problems. *Journal of Computer Science and Technology*; 2008. Volume 23, Issue 1, p 2-18.
18. Gupta.D.K, Arora, Singh.U.K, Gupta.J.P. Recursive Ant Colony Optimization for estimation of parameters of a function. *Recent Advances in Information Technology (RAIT), International Conference*; 2012. DOI: 10.1109/RAIT.2012.6194620, p. 448 – 454.