

Bioinspired Algorithms for Pricing Options

A thesis presented

by

Sameer Kumar

to

The Department of Computer Science
in partial fulfillment of the requirements
for the degree of
Master of Science
in the subject of

Computer Science

The University of Manitoba

Winnipeg, Manitoba

March 2008

© Copyright by Sameer Kumar, 2008

THE UNIVERSITY OF MANITOBA
FACULTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

Bioinspired Algorithms for Pricing Options

BY

Sameer Kumar

**A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of
Manitoba in partial fulfillment of the requirement of the degree**

MASTER OF SCIENCE

Sameer Kumar © 2008

Permission has been granted to the University of Manitoba Libraries to lend a copy of this thesis/practicum, to Library and Archives Canada (LAC) to lend a copy of this thesis/practicum, and to LAC's agent (UMI/ProQuest) to microfilm, sell copies and to publish an abstract of this thesis/practicum.

This reproduction or copy of this thesis has been made available by authority of the copyright owner solely for the purpose of private study and research, and may only be reproduced and copied as permitted by copyright laws or with express written authorization from the copyright owner.

Abstract

Option pricing is one of the fundamental problems in finance. Computing option prices is a challenging problem and finding the best time to exercise an option is an even more challenging problem. One has to be watchful for the asset price changes of financial assets in the market place and act at the right time. This work proposes a novel idea for pricing options using a nature inspired meta-heuristic algorithm called Ant Colony Optimization (ACO) that captures the asset price movements. ACO has been used extensively in combinatorial optimization problems and recently in dynamic applications such as mobile ad hoc networks. There has been no study reported in the literature on the use of ACO for pricing financial derivatives. We first study the suitability of ACO in finance and confirm that ACO could be applied to financial derivatives. We have then proposed two new ACO based algorithms to apply to derivative pricing problems in computational finance. In this study, asset prices are policed by ants to decide on the best time to exercise so that the holder will get the maximum benefit out of the option contract. The first algorithm, named Sub-optimal Path Generation generates various paths and identifies the best node in the solution space for exercising the option. In this algorithm, ants follow the paths generated by few leading ants to find better solutions as they search the solution space leading to what we call exploitation technique. In the second algorithm named Dynamic Iterative Algorithm, ants explore the solution space incrementally dragging more ants on the better path and eventually reaching the best node to exercise the option. This algorithm captures the market place by using an exploration and exploitation technique. We analyze advantages and disadvantages of both algorithms. Our goals for

the study, confirming the suitability of ACO for financial derivatives and identifying the best time for exercising an option under given constraints are achieved by both algorithms.

Contents

Table of Contents	iv
List of Figures	vi
List of Tables	vii
Acknowledgments	viii
Dedication	ix
1 Introduction	1
1.1 Basic Terminology	1
1.2 Nature Inspired Algorithms in Finance	3
1.3 Contributions of the Current Work	5
2 Some Option Pricing Techniques	7
2.1 Binomial Lattice Option Pricing Model	7
2.2 Some Numerical Option Pricing Techniques	12
3 Ant Colony Optimization (ACO)	15
3.1 Ant Colony Optimization (ACO)	15
3.1.1 Foraging behavior of ants	17
3.1.2 Different ACO algorithms	18
3.2 Application of ACO in Finance	22
4 ACO for Option Pricing	24
4.1 Problem Statement	24
4.2 A First Simple Algorithm	25
4.3 Design of ACO Algorithms for Option Pricing	27
4.3.1 Sub-optimal Path Generation Algorithm	28
4.3.2 Dynamic Iterative Algorithm	28
4.3.3 Some Observations	32
5 Implementation Details	33
5.1 Sub-optimal Path Generation Implementation	35
5.2 Dynamic Iterative Implementation	37

6	Results and Discussion	40
6.1	Sub-optimal Path Generation Algorithm	40
6.2	Dynamic Iterative Algorithm	44
6.3	Various Features of Sub-optimal and Dynamic Iterative Algorithm . .	49
6.4	Comparison between Sub-optimal, Dynamic Iterative Algorithm with Binomial Lattice	50
7	Conclusions and Future Work	52
A	Appendix	54
	Bibliography	57

List of Figures

2.1	Binomial Price Tree	10
2.2	Backward Computation of Option Prices	11
3.1	Overview of Search Techniques	16
3.2	Ants facing obstacle in their path	17
4.1	Initially ants wander randomly	26
4.2	Ants gradually start to find better nodes	26
4.3	Ants start following paths which lead to better nodes (bottom of the figure)	27
5.1	Flowchart for Sub-optimal path generation algorithm	36
5.2	Graph 1: Graph with 15 nodes	38
5.3	Graph 2: Graph with 30 nodes	38
5.4	Flowchart for Dynamic iterative algorithm	39
6.1	Chart for Desired Profit vs Execution Time with Time Steps 2000 . .	41
6.2	Chart for Time Steps vs. Execution Time with \$15 Profit	42
6.3	Initial Graph 1	44
6.4	Final Graph 1	44
6.5	Initial Graph 2	46
6.6	Final Graph 2	46

List of Tables

1.1	Similarities between ACO and the real market	4
3.1	A non-exhaustive list of successful ACO Algorithms	19
3.2	A non-exhaustive list of applications of ACO algorithms grouped by problem type	22
6.1	Desired Profit vs Execution Time with Time Steps 2000	41
6.2	Time Steps vs. Execution Time with \$15 Profit	42
6.3	Comparison between Binomial method and Sub-optimal path genera- tion algorithm	43
6.4	Number of Nodes vs Execution Time	48
6.5	Number of Maximum Links vs. Execution Time	49

Acknowledgments

First and foremost, I want to thank my advisors, Dr. Ruppa Thulasiram and Dr. Parimala Thulasiraman, for their guidance during my research and study at University of Manitoba. My very sincere thanks to them for the support, encouragement and great effort they put into training me in the scientific field during the course of this thesis.

I take this opportunity to express my profound gratitude from my heart to my beloved parents, “Dr. Keshari Kumar Singh” and “Mrs. Pushpa Singh”, for their never-ending support & faith and the sense of security they have always given me. I would especially like to thank my father for all the advises he gave me, when I needed them the most.

My deepest gratitude goes to my brother “Pawan Kumar Singh” and sister-in-law “Deepti Singh” for their love and support throughout my student life. I also would like to thank my niece “Vani” for the smile gifted to me, with her adorable and tender voice on the phone calling me Uncle.

All the graduate students at the Department of Computer Science made it a friendly place to work. I want to thank them for all their help, support, interest and valuable hints. In particular, I would like to thank Monirul Hasan for all his assistance and insight comments on the implementation issues.

This thesis is dedicated to loving memory of my late grandfather, "Dr. Shobh Nath Singh". I remember him for all his kindness and wisdom.

Chapter 1

Introduction

Computational Finance is a cross-disciplinary area that relies on mathematics, statistics, finance and computational algorithms to make critical decisions. One of the core tasks in this area is to analyze and measure the risk component that a financial portfolio would create. A portfolio would generally comprise stocks, bonds and other instruments such as derivatives. Derivatives are financial instruments that depend on some other assets such as stocks. They are also referred to commonly as options.

1.1 Basic Terminology

Option: An *option* is a contract in which the buyer (*holder* of an option) has the right but no obligation to buy (with *call* option) or sell (with *put* option) an underlying asset (for example, a stock) at a predetermined price (*strike price*) on or before a specified date (*expiration date*). The seller (also known as *writer*) has the obligation

to honor the terms specified in the option contract (*option*). The holder pays an premium to the writer (see for example [2] [3]).

There are two styles of options (*call* and *put*) and options can be divided into two categories, vanilla options and exotic options:

- An European option can be exercised only at the expiration date whereas an American option may be exercised on any date before the expiration date. Since the American options provides added flexibility, their premium is generally higher than European options. These two generally comprise vanilla options due to their simpler nature.
- Exotic options are complex. These include Asian (based on some average), barrier (looking for first stopping time), Bermudan (having few exercise points during the contract period), and Russian (expiration time itself is floating). These and other comprise exotic options.

Option pricing is one of the fundamental problems in finance which has lead to the award of two Nobel prizes. In 1997, Myron S. Scholes and Robert C. Merton shared the Nobel prize for the Black-Scholes-Merton model [4]. Recently, Engle received a Nobel prize in 2003 for his Auto-Regressive Conditional Heteroskedasticity (ARCH) model [5]. The generalized ARCH (GARCH) model has been a subject of intense research and use in option pricing (see for example [6] [7]). Based on the fundamental concept in these models, there are many numerical techniques proposed in the literature for option pricing. The next chapter reviews these techniques with detailed explanation on one of them, the binomial lattice approach [8].

1.2 Nature Inspired Algorithms in Finance

One of the current trends in science and engineering research is the introduction of nature inspired algorithms. Nature inspired algorithms (also known as Swarm Intelligence or evolutionary algorithms) [9] have been used in many applications to solve many combinatorial optimization problems, including problems in telecommunications [10] [11] and dynamic networks such as mobile ad hoc networks [12]. Swarm intelligence is an artificial intelligence technique inspired by animal societies such as bees [13] [14], termites [15], and ants [16]. These small animals live in a hostile, decentralized environment and communicate with one another through a stigmergic method to accomplish their tasks such as finding the food source. Ant Colony Optimization (ACO) is one such swarm intelligence technique inspired by real ants. The ants communicate with one another by depositing *pheromone* (scent) on the ground to attract their fellow ants to follow their trail, one of the stigmergic approaches in the animal world.

The feasibility of evolutionary algorithms in the field of finance has gained importance and is being explored [17]. These algorithms have prospects in many areas of finance such as to evolve trading rules, diagnosis of company's future etc. A key advantage of using evolutionary algorithms to design a trading process is that these algorithms can simultaneously evolve good rules and good parameters for such trading process [17].

We found many similarities between the ACO and the real market (Table 1.1). These similarities acted as motivation for the current study i.e. to apply ACO to the option pricing problem. For option pricing, the solution space consists of a large

ACO	Market
Meta-heuristic search technique	Investors look for best time to buy or sell
Based on the collective behaviour of decentralized ants; self-organized systems	Based on the collective behaviour of decentralized traders, investors etc.; self-organized systems
No centralized control structure	No central control among investors
Local interactions between agents lead to the emergence of global behaviour	Interaction between investors lead to emergence of market behaviour
The agents follow very simple rules which leads to very complex rules/algorithms at the global level	Investors follow simple rules that lead complex nature for the market

Table 1.1: Similarities between ACO and the real market

number of price nodes, each representing a time and price of the underlying asset during the option contract period. The ants are basically agents of an investor. They have a large bounded space of price nodes to search through in deciding the best time (node) to exercise the option. The nodes within the search space are dispersed in many locations and are connected in some random manner. The collective goal of the ants is to find the best node to exercise the option to help the investor in making an informed decision. This can be achieved by directing a path to the node, thereby allowing other agents to quickly arrive at the node. This path can be created using a variety of techniques as is discussed in the next chapter. ACO is a probabilistic approach that allows the path to be created in a random way. That is, the graph is ad hoc and random. There is no real structure to the graph. Therefore, ACO allows flexibility by distributing the nodes randomly and thereby capturing the real market place.

1.3 Contributions of the Current Work

In this thesis, we use ACO to compute option prices. The objective of ACO in most applications is to find the shortest path. However, in option pricing, the primary interest is not in finding the shortest path, it is rather finding the best node that brings large benefit to the investor by exercising the option at that node. In other applications, the destination (food source) is known, however in the option pricing problem we are looking to compute the destination (best time to exercise the option). Other applications involved ant cooperation based on distance/time between the nodes/activities but the option pricing problem involves cooperation based on asset values. Therefore, the general purpose ACO algorithm has to be redefined and redesigned for use in the option pricing problem. In the current work, we first study the suitability of ACO in finance and confirm that ACO could be applied to financial derivatives. We started with a simple algorithm, which did not have any cooperation among the ants in finding an optimal node. However, this understating acted as a good starting point. This experience and the analysis of the results gave us the knowledge to develop improved algorithms based on ACO. We have then designed two new ACO based algorithms to apply to derivative pricing problems in computational finance. In this study, prices are policed by ants to decide on the best time to exercise so that the holder will get the maximum benefit out of the option contract. The first algorithm is a *reactive* algorithm, named *Sub-optimal Path Generation Algorithm*. This algorithm generates various paths and identifies the best node in the solution space for exercising the option. In this algorithm, ants follow the paths generated by some leading ants to find better solutions as we search

the solution space leading to an exploitation technique. In the second algorithm, a *proactive* algorithm named *Dynamic Iterative Algorithm*, few ants explore the solution space incrementally dragging more ants on the better path and eventually reaching the best node to exercise the option. This algorithm captures the market place by using an exploration and exploitation technique. We analyze advantages and disadvantages of both algorithms. Our goals for the study, confirming the suitability of ACO for financial derivatives and identifying the best time for exercising an option under given constraints are achieved by both algorithms.

The current study is a proof of concept using ACO for derivative pricing. We use American options as they are computationally challenging. To the best of our knowledge the literature on pricing options using ACO is none.

We divide the rest of the document as follows. In the next Chapter, we provide some background information on option pricing with a detailed description of one of the common and classical techniques, the binomial lattice approach. We start Chapter 3 with a description of ACO algorithm and then we highlight some related work in general finance that uses other nature inspired algorithms. We describe the sub-optimal path generation and dynamic iterative algorithms in Chapter 4. In Chapter 5, we present the implementation details. We provide results and discussion in Chapter 6 and the last section presents conclusions and future work.

Chapter 2

Some Option Pricing Techniques

In this chapter we describe the binomial lattice model and other techniques available to price options.

2.1 Binomial Lattice Option Pricing Model

One of the early models for option pricing is the Nobel prize winning Black-Scholes-Merton (BSM) model [4] [18]. Black-Scholes developed a model to alleviate risk involved in financial investments. Merton [18] augmented it with stochastic calculus resulting in a stochastic partial differential equation (PDE) for the option. This model is valid for simple European options and a major assumption was that underlying stock would have constant volatility. A closed-form solution is available only for simplified BSM model with many assumptions, especially since numerical techniques for solving PDE were not popular in the finance community at that time. This scenario changed in 1979 when first discrete time approach was proposed by

Cox-Ross-Rubenstein (CRR) [8]. The CRR binomial model is a simple and intuitive numerical technique developed for pricing options. The binomial option pricing model has proved over time to be the most flexible, and popular approach to price options. If constructed assuming the same initial conditions, binomial model agrees asymptotically with the Black-Scholes model. Moreover, binomial model can be used for pricing American style options. The standard binomial option pricing model assumes that the binomial tree is recombining with constant volatility, constant riskless return and constant payout return. However, these could be relaxed unlike the BSM Model.

Binomial model uses a binomial tree structure to price options. We divide the time between valuation (current) date and expiration date into a certain number of time steps. Each node in the tree represents a possible price of the stock (underlying asset) at a particular time. In binomial method, knowing the asset price is the basis for computing the option value. The valuation of the binomial tree is iterative; After building the tree with asset price distribution it starts from the leaf nodes and works backwards towards the root node, which represents the valuation date. The option price at the valuation date is calculated by pricing option at all the intermediate nodes between expiration date and the valuation date. This process, called “discounting” involves moving backward in time from the expiration date to the valuation date.

Computing option using this iterative method involves three steps:

1. Price tree generation
2. Computing option price (local pay-off) at each leaf node
3. Iterative computation of option values at earlier nodes using a discounting fac-

tor. The value at the root node is the value of the option.

Some of the variables and parameters that are required to price an option are: Asset Price: S ; Strike Price: K ; Time to Maturity: T ; Interest Rate: r ; Number of Steps: N ; Interval time between 2 steps: Δt ; Volatility: σ ; Probability: p ; Upward Movement: u ; Downward Movement: d .

Binomial Tree

The prices at each node is computed by working forward through the tree from valuation date (current date) to expiration date. An example of the binomial price tree is shown in figure 2.1. At each step (or level) of the tree, the price of underlying asset can increase or decrease by a factor of u or d respectively, where u and d are generally 10% or 20% change. If S is the current price of the underlying asset, then the price at the next step will be either $S_u = S \times u$ or $S_d = S \times d$. Asset price computation is repeated until the expiration date is reached. The value of the increase (u) or decrease (d) factor is computed using the volatility σ of the underlying asset and the interval time between two steps [8] as follows:

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = e^{-\sigma\sqrt{\Delta t}} = \frac{1}{u}$$

It is easy to notice that the down movement is assumed to be inversely proportional to the up movement, to generate recombining tree.

Option value at the expiration date: The computation of option value starts at the leaf nodes (maturity date) of the binomial tree. The value of the option at the leaf nodes is simply its local pay off, which is defined as follows:

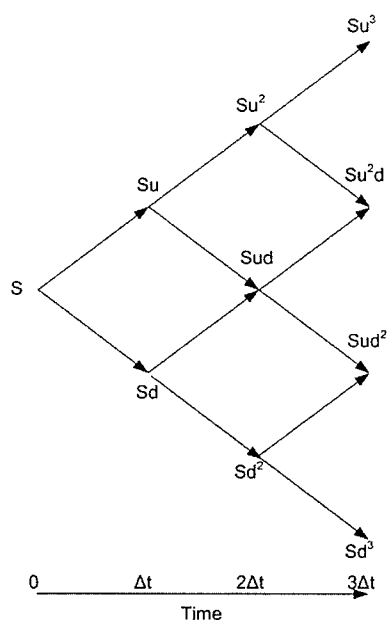


Figure 2.1: Binomial Price Tree

for a call option it is $= \text{Max}[(S - K), 0]$

and for a put option it is $= \text{Max}[(K - S), 0]$

Option value at intermediate nodes: Once the value of the option is computed at the leaf nodes, working backward (figure 2.2) towards the root node (valuation date) gives the value of the option. The option value at a node is computed using the option values of the two children nodes (*optionup* and *optiondown*) weighted by respective probabilities. *optionup* is multiplied by p , which is generally understood/interpreted as the probability of underlying asset to move up and *optiondown* is multiplied by $(1 - p)$, which is the probability of underlying asset to move down. The p is given by [8]

$$p = \frac{e^{(r-q)\Delta T} - d}{u - d}$$

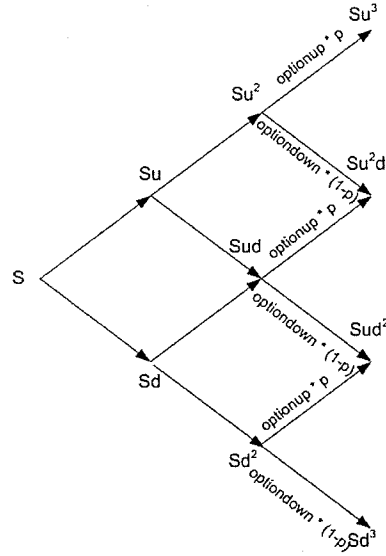


Figure 2.2: Backward Computation of Option Prices

and the option value is given by

$$OptionValue = e^{-r\Delta T}(p \times optionup + (1 - p) \times optiondown)$$

where the $e^{-r\Delta T}$ is the discounting factor. In other words, the option value at a node is the discounted value of the weighted sum of option values at a future time due to up or down movement of the underlying asset.

For the European option, the value at each node is simply the option value computed using the formula stated above. The value at the root node is the value of the option on the valuation day. For the American option, the value at each node is $Max[option\ value\ based\ on\ discounting, local\ pay-off]$ where local pay off is $\max(S - K, 0)$ for a call option or $\max(K - S, 0)$ for a put option. This step allows to identify the best possible time to exercise the American option. Thulasiram et al. [19] have designed and developed a parallel and multithreaded algorithm for this problem. Thu-

lasiram and Bondarenko [20] have developed parallel algorithm for multidimensional option pricing problem. Huang has extended these studies to Asian options [21] [22].

Now we will look at other popular techniques used for pricing options.

2.2 Some Numerical Option Pricing Techniques

There are many other techniques for pricing options such as Finite differencing to solve Black-Scholes model, Monte-Carlo simulations, Fast Fourier Transform (FFT), Neural Networks and Genetic Programming. I will briefly discuss finite differencing , Monte-Carlo simulations and FFT in this section.

The finite-difference methods (see for example [23]) solve the Black-Scholes partial differential equation (PDE) by approximating the partial derivatives by either an implicit or explicit discretization of individual terms in the PDE and then solving the resulting linear algebraic system of equations. Implicit method involves solving PDE by indirectly solving a system of simultaneous linear equations (where unknown variable being solved for, is itself used in the solution. Therefore, there is a higher cost of iteration for convergence) and convergence is always assured. Explicit method solves a PDE by using the appropriate boundary conditions and proceeding in time through small intervals. These methods are easy and effective when number of terms in the PDE is less. The choice of the discretization technique depends on the accuracy, convergence, stability, and the execution time required for the intended application.

There are many methods to discretize the partial differential equation for the finite-difference technique. For example, the explicit discretization technique is very simple but would compromise the accuracy of the result. Fully implicit finite-difference

method can be implemented with some ease but has some serious stability issues. On the other hand, the Crank-Nicolson method [24] has much better accuracy and stability and is popular for many general PDEs. While the error in the Crank-Nicolson method is bounded, there is no guarantee that this error will not propagate in the solution domain. One way to avoid this kind of error propagation is to use classical Páde approximation technique explored in [25] for financial application.

Carr and Madan [26] used the fast Fourier transform (FFT) approach for pricing options. The underlying idea of the method is to develop an analytic expression for the Fourier transform of the option price and to get the price by Fourier inversion. The approach assumes that the characteristic function of the log-price is given analytically. This algorithm offers advantages like speed, which allows the user to compute the prices for a whole range of strike prices. A mathematical improvement to [26] was provided by Barua et al. [27] and the improved model captured a wide variety of strike and spot (current) prices. Also, this new model was implemented in a parallel computing environment [27] [28] and the results were obtained in a shorter time for series of strike prices.

Monte Carlo simulation is widely used for pricing derivatives due to absence of straight forward closed form solutions for many financial models. Boyle [29] first introduced Monte Carlo simulation to option pricing. The Monte Carlo simulation allows simulation of several sources of uncertainties that affect the value of the underlying asset (and hence the option), given an optimal rule for exercising the option. The more the sampling size the better results can be expected by this technique. However, as the sampling size increases, computational resource requirement also in-

creases. For simpler options models, Monte Carlo is not a good choice because it is very time-consuming in terms of computation. The Monte Carlo simulation approach is of interest to solve complex real options models. Chen [30] [31] studied the quasi Monte Carlo method for option pricing. Quasi-Monte-Carlo technique introduces some determinism in the simulation that brings challenges in parallel implementation of the Monte-Carlo technique.

Summary: Until now we briefly described the techniques that are used for pricing options. We propose to use ACO in this work for pricing options. We explain Ant Colony Optimization meta-heuristic in the next chapter.

Chapter 3

Ant Colony Optimization (ACO)

Swarm Intelligence is an approach to solve complex problems based on the social behaviors of some insects and animals. Ant Colony Optimization is an evolutionary algorithm and is a part of guided random search techniques (Figure 3.1). The ACO [16] [32] is a technique inspired by real ants in nature. ACO has been used to solve many combinatorial optimization problems and recently in a more dynamic environment, mobile ad hoc networks.

3.1 Ant Colony Optimization (ACO)

The ACO approach is based on the foraging behavior of some ants. Many studies have shown interest in finding how ants in nature are able to find shortest paths between their nest and food sources. Research in this area has discovered that ants use indirect communication to communicate with other ants. This indirect communication involves modifying their environment, and is called *stigmergic* communication.

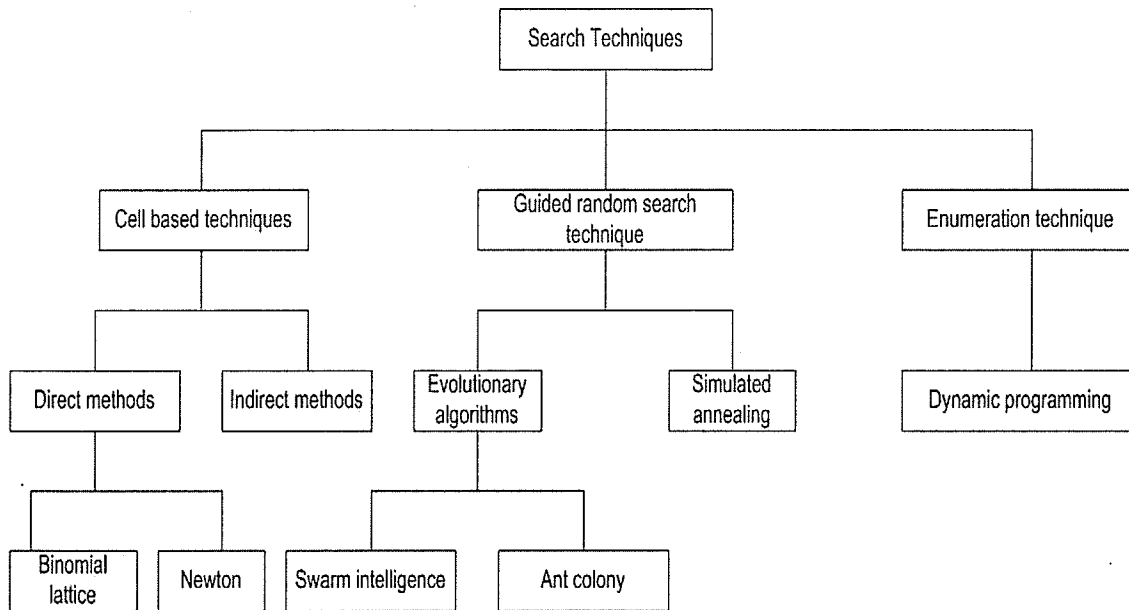


Figure 3.1: Overview of Search Techniques

In stigmergic communication, the ants move to food source from their nest, depositing on the path (ground) a chemical substance excreted from their body known as *pheromone*. Other ants can sense and perceive these pheromones and tend to follow the path with highest pheromone concentration. Using this communication, ants are able to forage and collect their food in an efficient way.

ACO involves a number of artificial ants (who behave like real ants) to build solutions to an optimization problem. These ants exchange information about their own solutions to other ants using a communication scheme similar to the one used by real ants [1].

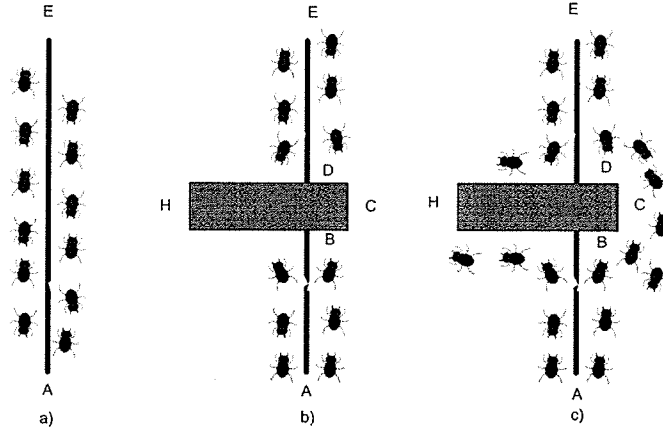


Figure 3.2: Ants facing obstacle in their path

3.1.1 Foraging behavior of ants

Consider figure 3.2a the ants move from Node A, the nest, to node E, the food source. We introduce an obstacle which obstructs the path between A and E. Ants have to make a decision whether to turn right or left at position B or D depending on whether they are coming from A to E or E to A respectively (Figure 3.2b). The decision or choice of an ant is based on the intensity of the pheromone on the trails. The first ant reaching point B or D has the same probability to turn right or left, as there is no pheromone on either of the two alternative paths. The ants following the shorter path will reach destination faster than ants following the longer path. In the example, the path BCD is shorter than the path BHD (Figure 3.2c). Therefore, the number of ants following the path BCD per unit time will be higher than the number of ants following BHD assuming all ants move at the same speed. This causes the pheromone levels on the path BCD to be higher than on path BHD. The ants approaching point B chooses path BCD because of higher pheromone levels,

thereby, following a shorter path to the food source E [33]. As time progresses, the pheromone along longer path evaporates leaving the ants to converge along shorter paths.

The objective of ACO in most applications is to find the shortest path. However, in option pricing, the primary interest is not in finding the shortest path, rather finding the best node that allows the investor to exercise the option. Therefore, the general purpose ACO algorithm has to be improved to handle this problem.

3.1.2 Different ACO algorithms

In the literature many types of ACO algorithms have been proposed. The first algorithm based on the behavior of ants was the Ant System (AS) [33]. This paper introduced an optimization technique based on ant behavior and proposed solution for solving Traveling Salesman Problem (TSP). In the TSP solution, an ant is placed on each city and it travels from current city, visiting other cities only once and returning to the original or initial city at the end of the tour. The ants communicate by depositing pheromones on edges connecting the cities. Eventually, the pheromone concentration on the shortest path increases due to more number of tours by ants. The pheromone on unused edges gradually evaporates completely in absence on any reinforcement of pheromone. Ant System work led to a development of number of ACO algorithms with good results in many applications. Table 3.1 outlines a non-exhaustive list of ACO algorithms in chronological order.

Applications of ACO

Dorigo [32] [33] proposed ACO which involves distributed computation, positive

Algorithm	Authors	Year
Ant System (AS)	Dorigo et al.	1991
Elitist AS	Dorigo et al.	1992
Ant-Q	Gambardella & Dorigo	1995
Ant Colony System	Dorigo & Gambardella	1996
MAX-MIN AS	Stützle & Hoos	1996
Rank-based AS	Bullnheimer et al.	1997
ANTS	Maniezzo	1999
BWAS	Cordón et al.	2000
Hyper-cube AS	Blum et al.	2001

Table 3.1: A non-exhaustive list of successful ACO Algorithms

feedback and a greedy heuristic to solve a given problem. Distributed computing helps in searching a wide area of the problem domain, positive feedback helps in finding good solutions and a greedy heuristic is needed to find solutions in early stages. ACO has found many applications and successful implementation has been applied to a number of different combinatorial optimization problems. Some examples of ACO being applied to static optimization problems are:

- **Travelling Salesman Problem:** The traveling salesman problem (TSP) requires the shortest route to visit a collection of cities and return to the starting point. Some TSP problems also require constraints like the salesman can only visit each city only once. Dorigo and Gambardella [34] applied ACO to the Travelling Salesman Problem (TSP). They concluded that using pheromones (positive feedback), ants were able to make shorter feasible tours. Their simulations demonstrated that ACO can give good paths for both symmetric and asymmetric instances of TSP. This algorithm outperformed simulated annealing and genetic algorithms [35].

- Quadratic Assignment Problem: In a standard problem, we are given a set of n locations and m facilities, and told to assign each facility to a location. The aim is to find the assignment that minimizes the cost. ACO have been applied to the quadratic assignment problem by Maniezzo and Colorni [36].
- The Job-shop Scheduling Problem: An instance of the job-shop scheduling problem consists of a set of n jobs and m machines. Each job consists of a sequence of n activities so there are total of $n \times m$ activities. Each activity has a duration to complete and requires a single machine for its entire duration. The activities within a single job require different machines. Two activities cannot be processed or scheduled at the same time if they both require the same machine. The objective is to find a schedule that minimizes the overall completion time of all the activities. ACO has been applied to job-shop scheduling problem [37] [38]. Yoshikawa and Terai [38] demonstrated that using ACO showed improvement compared to conventional scheduling techniques.
- Shortest Common Supersequence Problem: The shortest common supersequence is a common supersequence of minimal length. In this problem, the two sequences are given and the job is to find a shortest possible common supersequence of these two sequences. Michel and Middendorf [39] applied ACO to solve shortest common supersequence problem. They concluded that ACO is a promising alternative for strings which are not purely random.
- Graph Coloring Problem: Graph coloring problem involves assignment of colors to objects in a graph subject to certain constraints. It is a way of coloring the

vertices of a graph such that no two adjacent vertices share the same color also called vertex coloring. ACO have been used to solve this problem with the smallest possible number of colors [40].

- **Vehicle Routing Problem:** The Vehicle Routing Problem is a broad name given to a class of problems in which a set of routes for a fleet of vehicles based at one or more depots must be determined for a number of geographically spread cities or customers. The objective of the problem is to find minimum cost vehicle routes originating and terminating at a depot to deliver a set of customers with known demands. Bullnhiemer et al. [41] used ACO to solve the vehicle routing problem.
- **Routing in ad-hoc networks:** Most of the problems in combinatorial problem are static. Recently, ACO has been used for routing in ad-hoc networking. The use of the ACO technique for determining routing in ad hoc networks is an active area of research and exploration (see for example [10], [12], [42], [43], [44], [45], [46]).
- **Sequential Ordering Problem:** The Sequential Ordering Problem with precedence constraints consists of finding a minimum weight Hamiltonian path on a directed graph with weights on the edges and on the vertices, subject to precedence constraints among nodes. Gambardella and Dorigo used ant colony system to solve the sequential ordering problem [47]

Table 3.2 refers to an non-exhaustive list of applications of ACO algorithms grouped by problem type.

The following section provides a brief review of the related work on ACO in finance.

Problem Type	Problem Name	Authors	Year
Routing	Traveling salesman	Dorigo et al.	1991, 1996
		Dorigo & Gambardella	1997
	Vehicle routing	Stützle & Hoos	1997, 2000
		Gambardella et al.	1999
Assignment	Sequential ordering	Reimann et al.	2004
		Gambardella & Dorigo	2000
	Quadratic assignment	Stützle & Hoos	2000
		Maniezzo	1999
Scheduling	Course timetabling	Socha et al.	2002, 2003
	Graph coloring	Costa & Hertz	1997
	Project scheduling	Merkle et al.	2002
	Total weighted tardiness	den Besten et al.	2000
Subset	Open shop	Merkle & Middendorf	2000
		Blum	2005
	Set covering	Lessing et al.	2004
	l-cardinality trees	Blum & Blesa	2005
Other	Multiple knapsack	Leguizamón & Michalewich	1999
	Maximum clique	Fenet & Solnon	2003
	Constraint satisfaction	Solnon	2000, 2002
	Classification rules	Parpinelli et al.	2002
		Martens et al.	2006
	Bayesian networks	Campos, Fernandez-Luna	2002
	Protein folding	Shmygelska & Hoos	2005
	Docking	Korb et al.	2006

Table 3.2: A non-exhaustive list of applications of ACO algorithms grouped by problem type

3.2 Application of ACO in Finance

To the best of our knowledge, there is no work reported in the literature on the use of ACO for option pricing problem. In this section, we briefly outline the literature related to the use of ACO on other problems and applications, including finance though not option pricing.

In finance, knowing volatility of a financial instrument makes an investor/banker powerful and could make precise decisions. However, predicting volatility of an instru-

ment, say stock price, is a daunting task and is a research area by itself. Researchers and practitioners, using historical data have developed methods for predicting historical volatility (see [48] for a discussion on volatility). Historical volatility would only reflect past and accuracy of stock prices predicted based on historic volatility is questionable. There have been some efforts in measuring volatility accurately. Implied volatility measures the intrinsic dependence of past stock prices not only with time but with other factors affecting the price over a period of time. Keber and Schuster [49] used generalized ant programming to derive analytical approximations to determine the implied volatility for American put options. They used experimental data and validation data sets for computing the implied volatility. Their results outperformed any other approximations. Generalized ant programming is a new method inspired by genetic programming approach introduced by Koza [50] and Ant Colony System (ACS) [34] [51]. It is to be noticed that generalized ant programming is not ACO, which brings us to state the problem for our thesis.

Summary: In this chapter, we briefly described the related work concerning ACO and less explored financial application for nature inspired algorithms. We state the problem for the current study, namely, ACO for option pricing in the next Chapter followed by a solution methodology.

Chapter 4

ACO for Option Pricing

In this chapter, we discuss the problem and solution methodology.

4.1 Problem Statement

The objective of ACO in most applications is to find the shortest path. However, in option pricing, the primary interest is not in finding the shortest path, it is rather finding the best node that brings large benefit to the investor by exercising the option at that node. Our goals for this study are (i) to confirm the suitability of ACO for financial derivatives and (ii) identifying/computing the best time for exercising an option under given constraints. We started with a simple algorithm, which did not have any cooperation among the ants in finding an optimal node. We have then designed two new ACO based algorithms to apply to derivative pricing problems in computational finance.

4.2 A First Simple Algorithm

First we designed and implemented the following naive algorithm:

1. The solution space is divided into several subspaces which the ants explore to achieve the goal of finding the optimum node (time) to exercise. The algorithm divides the solution space based on the price of the underlying asset (vertical) and the time interval (horizontal). For the basic implementation of a small tree, we divided the horizontal and vertical axis in 3 steps each for a sub-space. However, we varied this over several runs of the algorithm.
2. Since there is an opportunity to exercise an American Option any time (before the expiry date) the holder finds it profitable, the pricing becomes an open boundary problem. That is, one has to check many possible nodes for maximizing the option value and find the best of these maximum values. In the current study as a simplification, we propose to let the holder prescribe either an option value or a particular time period before the expiration date to end the exploration. Also, instead of being greedy in finding the maximum in each subset of nodes, the holder could stop the search in a given subset once a predetermined value is reached. More ants are then injected from that node for exploring the next subspace.
3. Initially each ant randomly explores individual path emanating from the nest this is depicted in figure 4.1.
4. At each step, (in other words, for a given set of nodes) the best node is determined using a predetermined expression (for a put option $V_{def} \leq X - S \leq V_{opt}$,

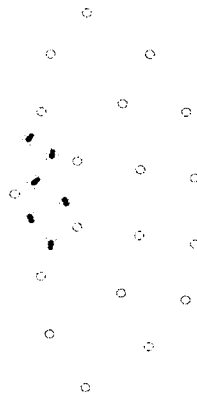


Figure 4.1: Initially ants wander randomly

where X is the strike price and S is the asset price; V_{def} is the predefined option value and V_{opt} is the optimal value of option. We use these as boundary conditions); this is repeated over several sub-solution spaces.

5. Once the best node is found, the pheromone density on the path leading to this node is increased so that more ants will likely follow this path. In figure 4.2, ants gradually find that better nodes are towards the bottom of the figure.

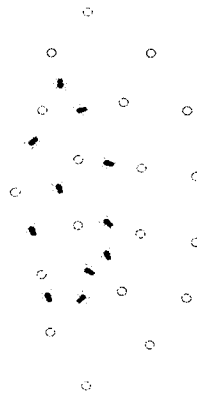


Figure 4.2: Ants gradually start to find better nodes

6. We keep finding better nodes until our conditions for an optimal node are met.

In figure 4.3, most of the ants start following paths which lead towards the best node.



Figure 4.3: Ants start following paths which lead to better nodes (bottom of the figure)

We were able to find the optimum node using the above algorithm. The design and implementation of the simple algorithm has been presented and published at MICS 2007 [52]. It is a naive algorithm because of lack of cooperation among the ants in finding the optimal node. However, this understanding acted as a good starting point. This experience and the results from the algorithm gave us the knowledge to come up with improved algorithms to achieve more cooperation among ants.

4.3 Design of ACO Algorithms for Option Pricing

With the above understanding , we developed and explored the following two improvements:

4.3.1 Sub-optimal Path Generation Algorithm

1. Algorithm starts by injecting ants from the valuation date (root). Ants can explore any path based on random behaviour.
2. Individual ants compute the payoff at each node based on an expression $V_{def} \leq X - S \leq V_{opt}$. As soon as an ant finds a value between the predefined values in the expression it updates (increasing) the pheromone density leading to the node.
3. Updating the pheromone on the path which has a good node (that satisfies the expression in the previous step) helps in making the path more attractive for other ants to explore more in the neighboring areas. However, ants still keep going on other paths to explore the whole search space.
4. If ants find a better node then pheromone values are updated to make the path to the newly found node more attractive.
5. Ants keep doing it until the most optimum node is found or until the constraint how far (how close to expiration date) the user wants to search in a graph is exceeded.

4.3.2 Dynamic Iterative Algorithm

1. In the solution space the source is the current date and the destination can be any node in the solution space. Destination is how far (future date) ants will search. The destination can be updated with a better node after some time to help us explore the whole search space. Here, "some time" refers to

the computational time needed to find the optimal node for a source and a destination.

2. The algorithm starts by injecting ants at the source, and the ants explore random paths to reach the destination.
3. Each ant while traveling to the destination identifies the best node throughout the journey which is computed using a predetermined expression (for example, in case of put option $X - S$, where X is the strike price and S is the asset price).
4. The ant updates the pheromone density (locally) on the path. Local pheromone update is done while looking for solutions. On the other hand, global pheromone update is done after all ants have found a solution. The purpose of the local pheromone update rule is to make the visited nodes less and less attractive as they are visited by ants, indirectly favoring the exploration of not yet visited nodes. The purpose of the global pheromone update rule is to promote ants to search for nodes in the neighborhood of the best node found so far.
5. Once all ants have updated their local pheromone values, the best nodes from all the paths explored by the ants are compared and the best node is identified. Now pheromone density is updated globally so that more ants follow the path which leads to the best node. The reason is to attract more ants so we can explore all possibilities from the identified node. However, to avoid a local minima, pheromone gets exhausted (by local update and evaporation) after some time which forces ants to explore other areas in the solution space. Here some time refers to the computational time taken by ants to slowly lower the

pheromone value to initial value.

6. We keep sending more ants from the source. Ants will follow the paths that have higher concentrations of pheromones compared to random behaviour, that was seen initially.
7. The algorithm keeps on finding better solutions until the best node is found.

We generate a random acyclic graph. A certain number of vertices (nodes) are connected to each other using edges (paths) randomly. The reason for not going for cyclic graph is that in real world it is not possible for investor to go back in time and have the same choices again. Ants can wander on these paths moving from node to node. Each node stores an asset price and each edge represents the transition from one stock value to another.

In the algorithm, ants deposit pheromone on the paths while walking and follow paths based on probability of pheromones deposited previously. Initially, all the paths have initial pheromone τ_0 on them so ants choose random paths. After a brief transitory time, the difference between the amounts of pheromones will differ. So the new ants coming from nest will prefer in probability to choose the path with higher pheromone. We have taken the probability expression from ACS [34] and modified to it suit the application. The following expression gives the probability an ant k at node r chooses to go to node s .

$$P_k(r, s) = \begin{cases} \frac{[\tau(r, s)][\eta(r, s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r, u)][\eta(r, u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases}$$

Where τ is the pheromone, η is the difference between the stock values of (r, s) , $J_k(r)$ is the set of nodes that ant k has still to visit and β is a measure to determine relative importance of pheromone versus difference between the stock values.

In this algorithm, the ant with the best value globally deposits the pheromone. This helps and probability equations are intended to make search more directed meaning ants mostly search in the neighbourhood of the best node found by the algorithm. Global updates are performed after all ants have reported their own best node. The best ant (globally) is identified based on the best values reported by all ants. The pheromone level is updated by this globally best ant by update rule

$$\tau(r, s) \leftarrow (1 - \alpha) \tau(r, s) + \alpha \Delta\tau(r, s)$$

$$where \Delta\tau(r, s) = \begin{cases} (V_{gb})^{-1} & \text{if } (r, s) \in \text{global best node} \\ 0 & \text{otherwise} \end{cases}$$

where α ($0 < \alpha < 1$) is the pheromone decay parameter and V_{gb} is the difference between the initial stock price and the best node globally. This is intended to provide greater amount of pheromone for more profitable nodes.

Ants while search for the solution, visits paths and alter their pheromone by applying the local pheromone update

$$\tau(r, s) \leftarrow (1 - \rho) \tau(r, s) + \rho \Delta\tau(r, s)$$

where $0 < \rho < 1$. Please refer to Section 5.2 for a set of parametric conditions for the current study.

4.3.3 Some Observations

The ants in the natural world try to find a shortest path from nest to a food source. The ad hoc network applications uses this idea from natural world directly, in finding shortest route for sending messages/packets. The idea from natural world put forth in ACO algorithm is improved for the finance application as follows:

- The option pricing application requires finding the best time for exercising the option. That means the ‘food’ consumption is going to happen only once (exercise of option). That single best node would become the destination where the option is exercised. In other words, the first major modification is that we relax the requirement of the shortest path for the ants to follow.
- The local optimum values are used to direct more ants to explore further in the solution space from the local optimum node.
- ACO has not been used to price options. Designing and implementing naive algorithm acted as a feasibility study for the current work. ACO only generates and evaluates a subset of the paths unlike binomial lattice model where all paths and nodes are exhaustively evaluated. This would address memory issues and explore the best time to exercise the option in an efficient way.

Summary: In this chapter, we described our algorithms. We describe the implementation details of the algorithms in the next chapter.

Chapter 5

Implementation Details

In the literature on parallelizing combinatorial optimization problems such as TSP [53] and scheduling problems [54], where the graph is static, the implementation has been done on parallel computers using the Message Passing Interface (MPI) [55] or OpenMP [56]. Shared memory machines have shown [57] to produce better performance results for ACO algorithms. In applications such as mobile ad hoc networks, where the graph is dynamic and mobility needs to be incorporated, a simulator has been used for the implementation. Since, option pricing problem considers a static graph (solution space), we choose to implement our algorithms on a shared memory machine using OpenMP.

With *call* and *put* options we expect the underlying asset's price to go one direction, up for *call* and down for *put*. We can utilize this fact in channelling the ants in a particular direction towards the best node in the solution space. That is, when an initial set of ants move in one direction more ants follows in this direction. These ants are *reactive* ants, which we use in our sub-optimal path algorithm.

In another scenario, ants can explore the entire solution space for a best node both for *call* and *put* options. This is useful when there are both styles of options issued for the same underlying asset. The ants explore the solution space proactively on their own without any direction by the investor. While few of the *proactive* ants explore the solution space independently, others follow earlier ants just like in reactive case. Note that these definitions of proactive and reactive ants are different from usual networking literature. In fact, they have inverse meaning.

In our implementation, ants are agents for a single investor. In the current scenario all the agents are working for a single investor in finding a solution for the option prices that helps the investor in making an informed decision for entering the option contract. Node is a price point at a given time. Moving from a node to a future time (node in the future time), there could be various possible prices for the asset. The future time could be the next second, minute, hour, day or week. Each thread in the system represents an ant which can randomly move around the solution space. For ease of implementation, we have limited the number of nodes. We limit the number of nodes for implementation because of memory constraint. The primary goal of the algorithms is to find any good node (right time to exercise the option) rather than finding the shortest path leading to that node in minimum time. More nodes means more sampling, which is better, in general. However, sampling by minutes or every 10 minutes or even every 30 minutes may not yield much better results (in terms of accuracy) in pricing. Further, higher frequency sampling would result in heavier computational load and hence higher computational cost. Hence, we restrict the solution space to a reasonable number of steps.

5.1 Sub-optimal Path Generation Implementation

No study has been reported in the literature using ACO for the option pricing problem, let alone been parallelized. Therefore, we compare our results with binomial lattice results with same set of parameters (strike price, asset price, etc.) used in the sub-optimal path generation algorithm implementation. We will use synthetic test data to compare the proposed algorithms with the binomial model. The synthetic data used can easily be mapped to a real world stock.

The algorithm starts by sending ants randomly on each path (in the solution space) from an initial node. Once an ant reaches the next node it computes the local pay-off for a *put* option as $(X - S)$, where X is the strike price and S is the asset price at the new node. Once the solution satisfies either of the boundary conditions:

$$X - S \geq Y \quad (5.1)$$

or

$$(n * \Delta t) > a \text{ predefined time during contract period} \quad (5.2)$$

where Y and *predefined time* are user defined parameters, the option price computed is the optimum value. We have conducted experiments to price American *put* option using the proposed algorithms. We have experimented by varying the parametric conditions: initial stock price, strike price, volatility, time to maturity and number of time steps. The algorithm provides the optimal solution based on the user defined boundary conditions. The primary goal is to find the optimum node rather than finding it in shortest path or in minimum time. The measure of comparison is the pay-off an investor will get by exercising the option. The flow chart for the sub-

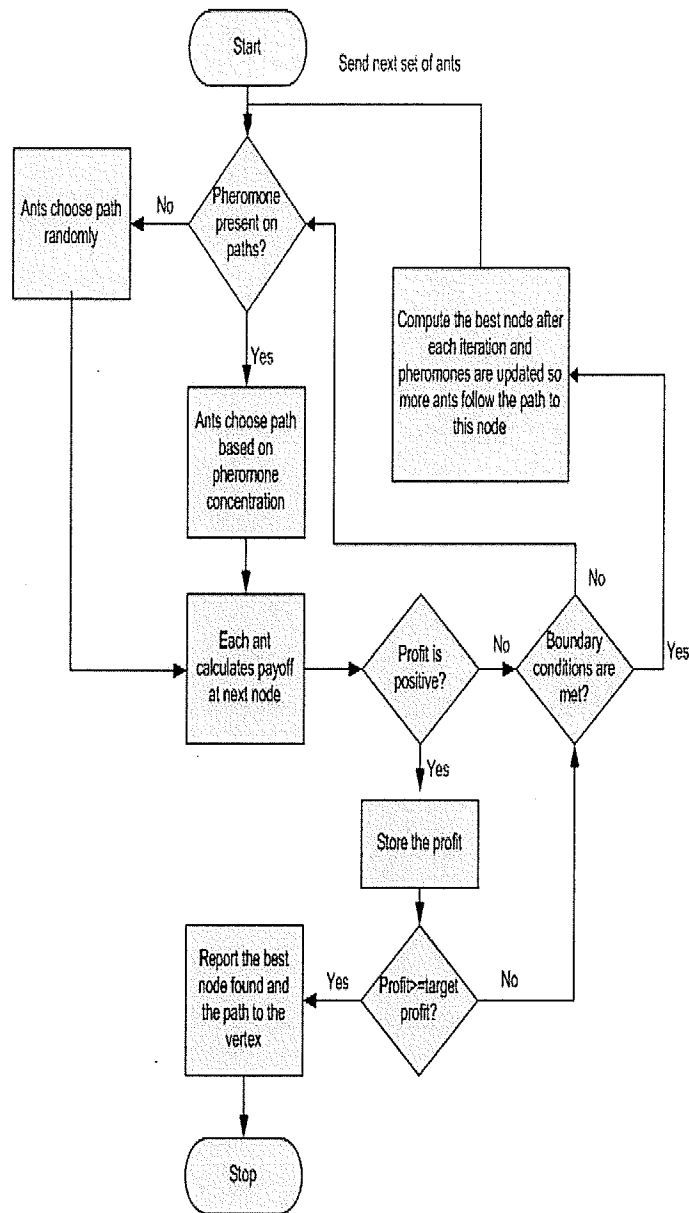


Figure 5.1: Flowchart for Sub-optimal path generation algorithm

optimal path generation algorithm is given (Figure 5.1).

5.2 Dynamic Iterative Implementation

Figures 5.2 (Graph 1) and 5.3 (Graph 2) show two random acyclic graphs used in the experiments with 15 and 30 nodes respectively. Here, $V_x : A$, refers to vertex V_x representing asset price A . For example in Figure 5.2, $V_7 : 15$ refers to vertex 7 with asset price 15. We applied the dynamic iterative algorithm to these random graphs.

The measure of comparison is the pay-off an investor will get by exercising the option. In other words, the time to exercise the option which gives us the highest profit. In all our experiments the numeric parameters are set to the following values: $\beta = 2, \alpha = 0.1, \rho = 0.1$ and $\tau_0 = 0.1$. The flow chart for the dynamic iterative algorithm is as follows (figure 5.4):

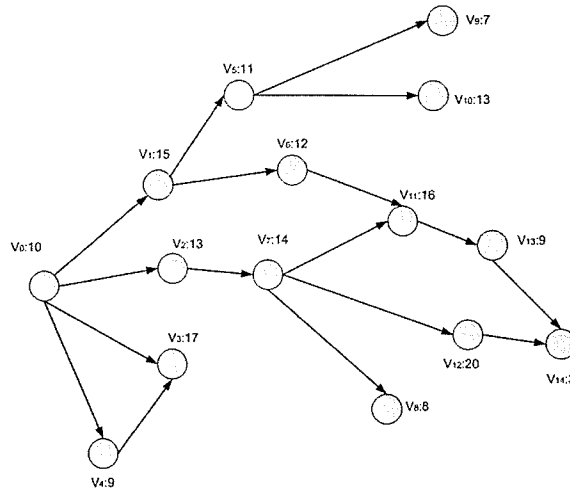


Figure 5.2: Graph 1: Graph with 15 nodes

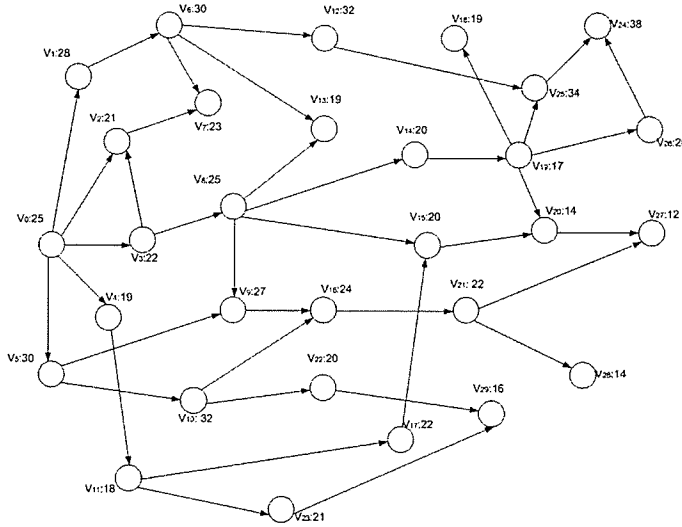


Figure 5.3: Graph 2: Graph with 30 nodes

Summary: In this chapter, we provided an outline of the implementation of our two algorithms. We analyze results in the next chapter.

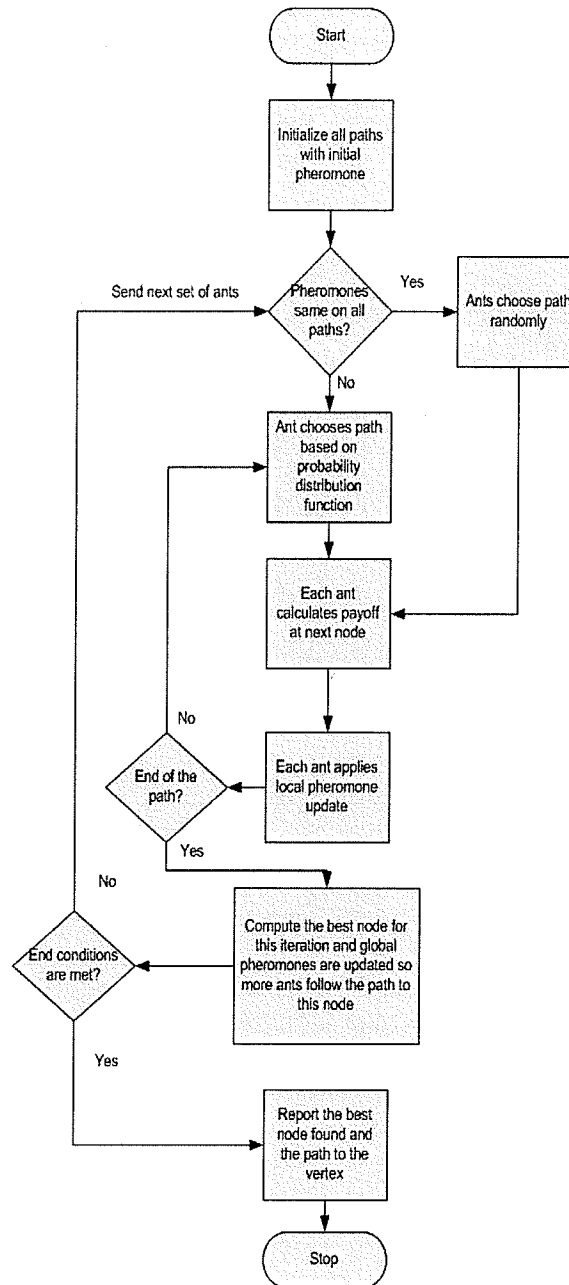


Figure 5.4: Flowchart for Dynamic iterative algorithm

Chapter 6

Results and Discussion

The experiments were done on an eight node shared memory machine with memory and hard disk of 7.5 GB and 36 GB respectively.

To compare and validate the results obtained from the ACO we independently implemented the binomial lattice algorithm for different data sets used in the ACO study and gathered both timing and pricing results. The pricing results from both the algorithms agree with results from binomial method.

6.1 Sub-optimal Path Generation Algorithm

The experiments carried out for this algorithm were driven by the general market conditions. Market conditions generally demand results instantaneously to beat the competition. The graph used in this algorithm is analogous to binomial or trinomial tree. The contract period for our experiments is six months and number of time steps vary between 2000 (amounts to about 30 minutes interval) to 5000 (amounts to about

Parameters	Value1	Value2	Value3	Value4
Desired Profit	10	15	20	25
Execution Time (secs)	0.049	0.106	0.515	1.269

Table 6.1: Desired Profit vs Execution Time with Time Steps 2000

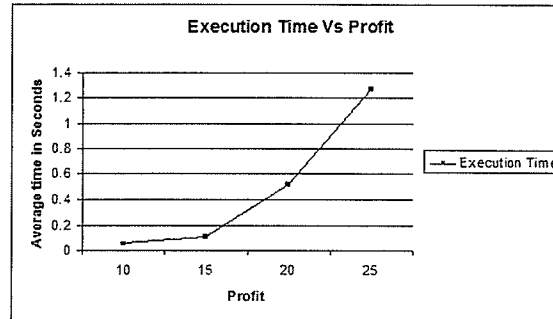


Figure 6.1: Chart for Desired Profit vs Execution Time with Time Steps 2000

11 minutes interval). That is, in some experiments ants compute option prices for changes in the asset price happening every 11 minutes and in other experiments ants compute option prices for changes in the asset price happening every 30 minutes. In table 6.1 and in Figure 6.1, we set the desired profit level between \$10 and \$25 at \$5 interval to determine how quickly one can achieve such profits. We set the number of time steps to be 2000. As the profit level increases, the execution time also increases. The reason for this is that for better profitability the ants would have to search more solution space and do more computation, hence increasing the execution time.

Table 6.2 and Figure 6.2 shows the execution time for various time steps at a desired profit of \$15. As the time steps increases with constant profit level, the execution time also increases. This is because increasing the number of time steps implies increasing the number of nodes in the solution space. Since the solution

Parameters	Value1	Value2	Value3	Value4
Time Steps	2000	3000	4000	5000
Execution Time	0.106	0.123	0.203	0.213

Table 6.2: Time Steps vs. Execution Time with \$15 Profit

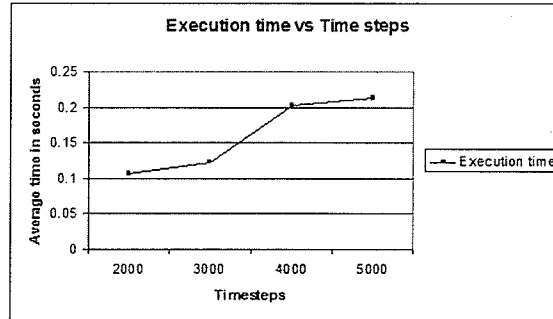


Figure 6.2: Chart for Time Steps vs. Execution Time with \$15 Profit

space is analogous to binomial tree, the number of nodes can be of the order of 2^L (for binomial tree), where L is the number of time steps. However, as can be seen from table 6.2, the increase in time is very negligible. We compared the sub-optimal algorithm to the binomial lattice method (Table 6.3). Our algorithm performed better than the binomial lattice method in terms of speed. The performance of sub-optimal algorithm is better because in the ACO algorithm we do not generate all the nodes at each time steps as in binomial lattice method. Our algorithm only generates and computes price on nodes which are needed to price the option.

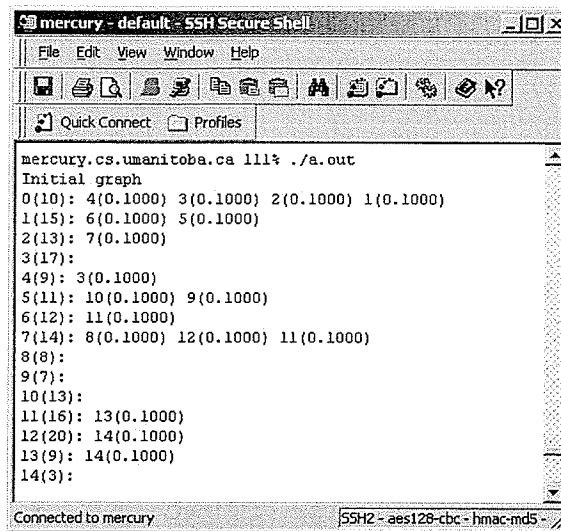
Since the ants search the solution space in various directions we observed a proportional increase of the execution time with higher desired profit. This happens due to structured search by the leading ants. This proportionality caught our attention to have a close look at the algorithm. Though the ants were allowed to search the

Timesteps	Binomial Method (Secs)	Sub-optimal Path Generation (Secs)
10000	3	0.37
20000	13	1.44
40000	67	2.38
70000	220	7.46
100000	451	11.71

Table 6.3: Comparison between Binomial method and Sub-optimal path generation algorithm

solution space, there was a controlled exploration. We allowed only a limited number of ants in a smaller region of the solution space to reach a sub-optimal solution at a node from which more ants were allowed to search in an orderly fashion. This is the nature of the algorithm. One set of ants searching and dragging more ants to the sub-optimal solution. That is, most of the ants are reacting to the first few ants. In other words, the *reactive ants* are exploiting the paths generated by few leading ants and hence they are more exploiting than exploring. Therefore, it is expected that with *proactive ants*, which explore the solution space, we would expect better results, which is studied in the dynamic iterative algorithm.

Also note that in the sub-optimal algorithm, we have a structured graph. The reason for this is that we initially wanted a fair comparison between the binomial lattice method and our ACO method. In the dynamic iterative algorithm, we do not restrict the structure of the graph. The data structure for the solution space is not necessary trees. They are random graphs used to capture real market movements.



```

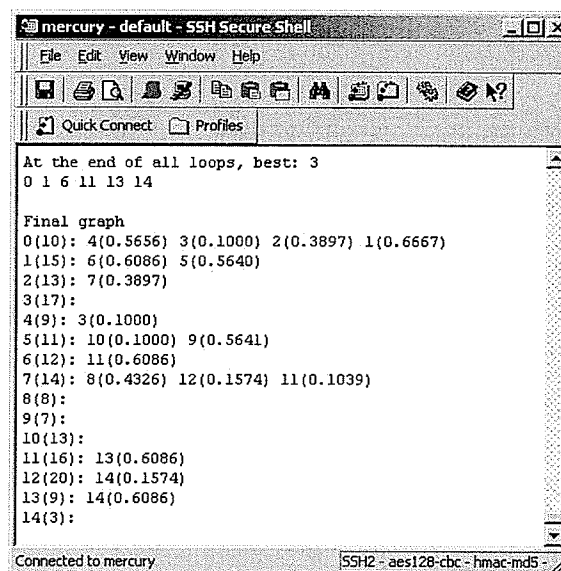
mercury - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

mercury.cs.umanitoba.ca 111% ./a.out
Initial graph
0(10): 4(0.1000) 3(0.1000) 2(0.1000) 1(0.1000)
1(15): 6(0.1000) 5(0.1000)
2(13): 7(0.1000)
3(17):
4(9): 3(0.1000)
5(11): 10(0.1000) 9(0.1000)
6(12): 11(0.1000)
7(14): 8(0.1000) 12(0.1000) 11(0.1000)
8(8):
9(7):
10(13):
11(16): 13(0.1000)
12(20): 14(0.1000)
13(9): 14(0.1000)
14(3):

Connected to mercury SSH2 - aes128-cbc - hmac-md5

```

Figure 6.3: Initial Graph 1



```

mercury - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

At the end of all loops, best: 3
0 1 6 11 13 14

Final graph
0(10): 4(0.5656) 3(0.1000) 2(0.3897) 1(0.6667)
1(15): 6(0.6086) 5(0.5640)
2(13): 7(0.3897)
3(17):
4(9): 3(0.1000)
5(11): 10(0.1000) 9(0.5641)
6(12): 11(0.6086)
7(14): 8(0.4326) 12(0.1574) 11(0.1039)
8(8):
9(7):
10(13):
11(16): 13(0.6086)
12(20): 14(0.1574)
13(9): 14(0.6086)
14(3):

Connected to mercury SSH2 - aes128-cbc - hmac-md5

```

Figure 6.4: Final Graph 1

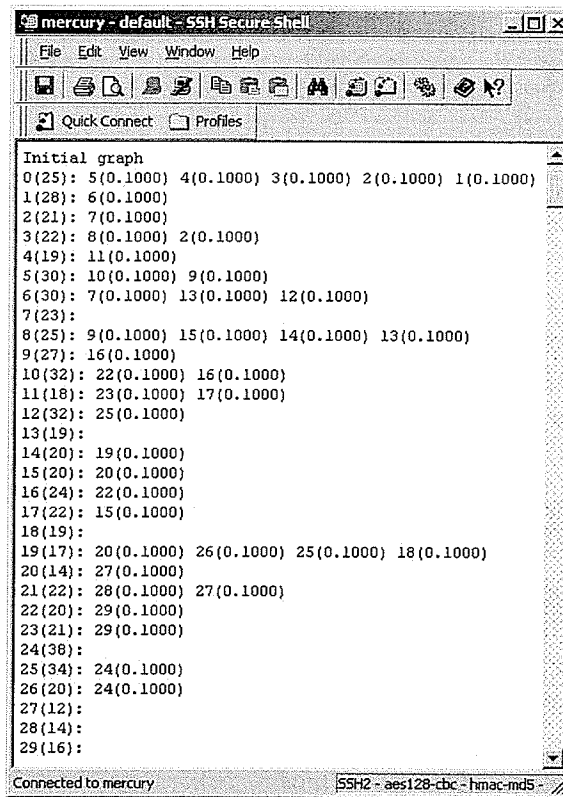
6.2 Dynamic Iterative Algorithm

We applied the dynamic iterative algorithm to both the graphs described in figures 5.2 and 5.3 . Figure 6.3 gives the initial graph for Figure 5.2 (Graph 1). The

graph is represented by an adjacency list. For example, $V_x(A) : V_y(p_1)V_z(p_2)$ means vertex (V_x) has an asset price of A and is connected to vertex V_y with pheromone p_1 and vertex V_z with pheromone p_2 . In Figure 6.3, 1(15): 6(0.1000) 5(0.1000) represents vertex V_1 with asset price 15 connected to vertex V_6 and vertex V_5 with pheromone level 0.1 at both nodes. Applying ACO to the initial graph shown in Figure 5.2 (Graph 1), we reach the final graph shown in Figure 6.4. The best vertex or node to exercise the option is V_{14} with an asset price of 3. The path to the best node is $V_0 \rightarrow V_1 \rightarrow V_6 \rightarrow V_{11} \rightarrow V_{13} \rightarrow V_{14}$. Also, note the changed pheromone levels by which ants are guided to the best solution.

Similar results for Figure 5.3 (Graph 2) are also shown in Figures 6.5 and 6.6. Figure 6.5 shows the list of vertices and how they are connected to other vertices in the graph. It also captures the initial pheromone level (i.e. 0.1) on the edges between the vertices. We apply our algorithm to Graph 2 and determine that the best node is vertex V_{27} with asset price 12. The algorithm also computes the path of the ants followed to reach this best node. The path is $V_0 \rightarrow V_4 \rightarrow V_{11} \rightarrow V_{17} \rightarrow V_{15} \rightarrow V_{20} \rightarrow V_{27}$.

For verification of our pricing results, we have done the following during our implementation. We opened a counter to store the asset prices from nodes when we generate the graph. We compared the asset price from a node as we generate the node with the asset value in the counter. For a *put* option, the counter price value is replaced with the asset price at a new node if the price at the new node is smaller than the counter price value (for a *call* option, the counter price is replaced with a larger node price). This information tells us beforehand where to expect the best option



```

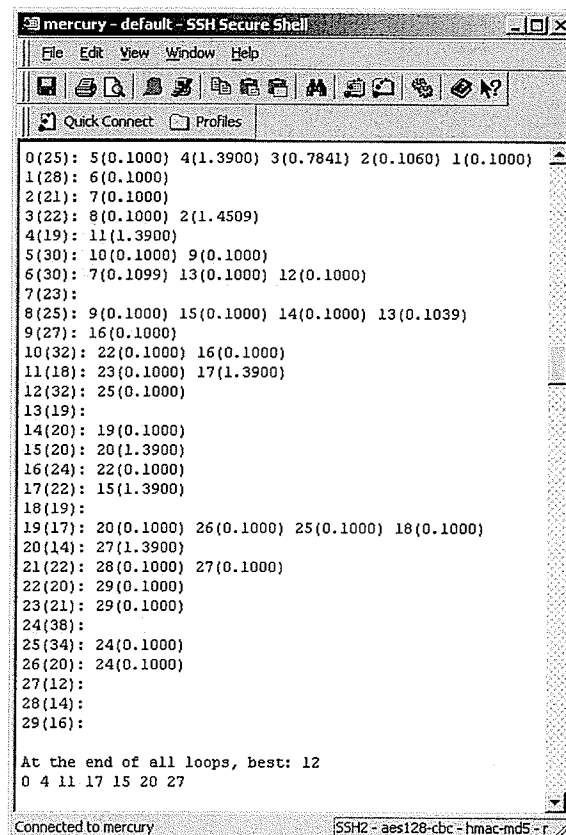
mercury - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

Initial graph
0(25): 5(0.1000) 4(0.1000) 3(0.1000) 2(0.1000) 1(0.1000)
1(28): 6(0.1000)
2(21): 7(0.1000)
3(22): 8(0.1000) 2(0.1000)
4(19): 11(0.1000)
5(30): 10(0.1000) 9(0.1000)
6(30): 7(0.1000) 13(0.1000) 12(0.1000)
7(23):
8(25): 9(0.1000) 15(0.1000) 14(0.1000) 13(0.1000)
9(27): 16(0.1000)
10(32): 22(0.1000) 16(0.1000)
11(18): 23(0.1000) 17(0.1000)
12(32): 25(0.1000)
13(19):
14(20): 19(0.1000)
15(20): 20(0.1000)
16(24): 22(0.1000)
17(22): 15(0.1000)
18(19):
19(17): 20(0.1000) 26(0.1000) 25(0.1000) 18(0.1000)
20(14): 27(0.1000)
21(22): 28(0.1000) 27(0.1000)
22(20): 29(0.1000)
23(21): 29(0.1000)
24(38):
25(34): 24(0.1000)
26(20): 24(0.1000)
27(12):
28(14):
29(16):

Connected to mercury
SSH2 - aes128-cbc - hmac-md5

```

Figure 6.5: Initial Graph 2



```

mercury - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

0(25): 5(0.1000) 4(1.3900) 3(0.7841) 2(0.1060) 1(0.1000)
1(28): 6(0.1000)
2(21): 7(0.1000)
3(22): 8(0.1000) 2(1.4509)
4(19): 11(1.3900)
5(30): 10(0.1000) 9(0.1000)
6(30): 7(0.1099) 13(0.1000) 12(0.1000)
7(23):
8(25): 9(0.1000) 15(0.1000) 14(0.1000) 13(0.1039)
9(27): 16(0.1000)
10(32): 22(0.1000) 16(0.1000)
11(18): 23(0.1000) 17(1.3900)
12(32): 25(0.1000)
13(19):
14(20): 19(0.1000)
15(20): 20(1.3900)
16(24): 22(0.1000)
17(22): 15(1.3900)
18(19):
19(17): 20(0.1000) 26(0.1000) 25(0.1000) 18(0.1000)
20(14): 27(1.3900)
21(22): 28(0.1000) 27(0.1000)
22(20): 29(0.1000)
23(21): 29(0.1000)
24(38):
25(34): 24(0.1000)
26(20): 24(0.1000)
27(12):
28(14):
29(16):

At the end of all loops, best: 12
0 4 11 17 15 20 27

Connected to mercury
SSH2 - aes128-cbc - hmac-md5 - r

```


price among various nodes generated. We use this to verify the best node computed by the dynamic iterative algorithm. This is only for verification purposes. In all our experiments, dynamic iterative algorithm worked well in computing the best node to exercise the option.

Parameters	Value1	Value2	Value3	Value4
Number of Nodes	10	100	1000	10000
Execution Time (in secs)	3.06	4.90	7.01	9.14

Table 6.4: Number of Nodes vs Execution Time

Table 6.4, shows that the execution time increases as the number of nodes is increased. Note that unlike the sub-optimal algorithm we do not fix a constant profit because the graph is random and we do not restrict the limit on the gain. In other words, we make the boundary open. From table 6.4, it can be seen for 10 nodes, the execution time is 3 seconds while for a larger graph (10000) it is 9.14 seconds. The overhead incurred in local and global pheromone updates, is predominantly attributed to the larger time needed for a smaller graph.

The market is full of uncertainties. We generally do not know what is going to happen tomorrow. In the binomial lattice algorithm the price change happens at a given node only in two different ways: up or down that too by a known factor. Similarly in trinomial lattice it happens three different ways. One advantage of ants for finding paths is that we can relax the restriction on price movements by letting the ants explore in many different possible paths naturally. This physically means ants can capture day-to-day changes of the volatility in the market place. Therefore, unlike the other numerical approaches we do not have to specify volatility of the underlying asset, one less parameter to handle in the implementation. For the simulation purposes, we restrict the number of the different possible links that an ant can have. In our implementations, we have case studies with number of links between 5 and 20 as shown in table 6.5. A single link between a node in a given time step

Parameters	Value1	Value2	Value3	Value4
Number of Maximum Links	5	10	15	20
Execution Time (in secs)	4.48	7.01	9.13	12.3

Table 6.5: Number of Maximum Links vs. Execution Time

to a node in the future (next time step) means the asset is stable. An experiment with 5 links means that an ant can have a maximum of five links going from one time step (node) to the nodes in the next time steps. That is, five different possible price changes are captured going from one node to the next. It need not have all five links. If there are all five links present from a node to the next time step, it implies that the underlying asset is highly volatile. In other words, we are able to capture the volatility of the underlying asset. Similarly, presence of 20 links at a node imply that the asset price is highly volatile. In table 6.5, it can be seen that as we increase the maximum number of links (or volatility), the execution time increases. This is because increase in volatility increases the computational intensity of the problem.

6.3 Various Features of Sub-optimal and Dynamic Iterative Algorithm

The performance of the sub-optimal algorithm is far better than the dynamic iterative algorithm in terms of speed. This is due to the structure of the graph, we use graphs that are analogous to binomial & trinomial trees. In one of our experiments, we compared the execution time for both the algorithms for 10000 nodes. Sub-optimal algorithm took less than one millisecond versus dynamic, which took 4.5 seconds (for

maximum of 5 links). However, it should be noted that the sub-optimal algorithm will not be able to find a good pricing solution in a dynamic environment in which dynamic iterative algorithm works. This is because we do not use the parameters such as evaporation criteria and local pheromone update that are used in the dynamic iterative algorithm. The evaporation criteria allows ACO to converge towards a better solution by providing a means of exploring many different good paths, while at the same time eliminating the paths leading to bad nodes. Sub-optimal algorithm produces faster results because we are exploiting the solution space generated by few initial ants. Sub-optimal algorithm mainly relies on exploitation of already discovered paths and nodes. This is good for a given style of option, *call* or *put* by channelling the ants and it is expected that both local and global optimal solution will be available in the same neighbourhood. Dynamic iterative algorithm is exploration as well as exploitation. Since it is advantageous to both styles of options *call* and *put*, it spends more time exploring. Also, by doing so it finds the global optimal solution. Dynamic iterative algorithm converges slower than the static algorithm because of the use of local pheromone update and evaporation.

6.4 Comparision between Sub-optimal, Dynamic Iterative Algorithm with Binomial Lattice

We compared our algorithms to the binomial lattice model. Sub-optimal algorithm and binomial lattice model gave us the same pricing results, that is the best time to exercise the option. Sub-optimal is faster than binomial model because binomial

model exhaustively prices option for all the nodes in the tree, whereas sub-optimal algorithm need not have to price options at all the nodes. Dynamic iterative algorithm is not compared with the binomial model as the dynamic iterative algorithm works in more complex and volatile environment than the environment in which binomial and sub-optimal algorithm works. For the dynamic iterative algorithm, structure of the solution space is random and the volatility parameter need not be specified unlike the binomial and sub-optimal algorithm. In other words, this algorithm could handle underlying assets that are represented with varying volatility models. In sub-optimal and binomial lattice algorithms these volatility models would pose large computational challenges.

Summary: We presented the results from both of our algorithms. We conclude our current research work and outline some of our future work in the next chapter.

Chapter 7

Conclusions and Future Work

Pricing of options is a challenging problem. This work proposed a novel idea of using nature inspired meta-heuristic algorithm called Ant Colony Optimization (ACO) for pricing options. We first studied the suitability of ACO in finance and confirmed that ACO could be applied to financial derivatives. Then, we designed and implemented two new ACO based algorithms to apply to a derivative pricing problem in computational finance. The first algorithm, named sub-optimal path generation generates various paths and identifies the best node in the solution space for exercising the option. In this algorithm, ants follow the paths generated by some leading ants to find better solutions as we search the solution space leading to an exploitation technique. Sub-optimal path generation algorithm outperformed the binomial lattice model. The second algorithm named dynamic iterative algorithm, where few ants explore the solution space incrementally dragging more ants on the better path and eventually reaching the best node to exercise the option. This algorithm captures the real market place and finds the best time to exercise the option using exploration and

exploitation techniques. Though the dynamic iterative algorithm converges slower than sub-optimal path generation algorithm, the dynamic iterative algorithm can be executed on any random graph. Dynamic iterative algorithm is better choice when dealing with dynamic and highly volatile market place.

Future Work

In this work, our study was limited to vanilla options. As a future work, we would like to apply ACO to exotic options such as to price Asian options. In our dynamic iterative algorithm, the ants keep track of the price values at each node along a path. We can easily use these price information to compute an average price so that we can apply this algorithm to price an Asian option. We intend to this in the near future. Similarly, the nature of our algorithms help us to price barrier option, where it is required to find the first stopping time, that is, earliest node to exercise the option. This can also be extended to Bermudan option.

We also intend to look into the possibility of using digital pheromones. Digital pheromones are data structures inspired by the insect model. In our application, we can allow ants to communicate more information among each other such as asset price, length of the path etc. using digital pheromones.

Evolutionary algorithms such as ACO algorithms are gaining importance in many areas of finance such as to evolve trading rules, diagnosis of company's future etc. This thesis work is the forerunner for more research to be done in future in financial applications using ACO algorithms.

Appendix A

Appendix

Data: Asset price tree

Result: Optimum node to exercise the option

Step 1: In this phase ants build their paths

for *Each ant inserted at the root node* **do**

if *Pheromone levels are zero* **then**

 ant chooses a path randomly

else

 If there are pheromones present on the paths then the ant chooses the paths accordingly

end

end

Each ant calculates the profit at each node ($X - S$)

if *profit is positive* **then**

 | it is stored

end

if *profit \geq target profit* **then**

 the program terminates and details about the corresponding node is printed

end

if *ant travels more than $\frac{3}{4}$ the total time* **then**

 ant dies

 break out of for loop

end

end

Step 2: Pheromones are updated on the paths

for *all ants* **do**

 | Compute the most profitable node

end

the pheromone on the path which leads to this profitable node is updated

go to step 1

Algorithm 1: Sub-optimal path generation algorithm

Data: Random graph

Result: Best vertex (Time) to exercise the option

Step 1: Initialization phase

for each pair of vertices (r,s) which are connected by an edge **do**

$\tau(r,s) = \tau_0$

end

Step 2: In this phase ants build their paths. The path of ant k is stored in

$Path_K$

for $K=1$ to m **do**

for $K=1$ to m **do**

 Choose the next vertex according to

$$P_k(r,s) = \begin{cases} \frac{[\tau(r,s)][\eta(r,s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r,u)][\eta(r,u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases}$$

$Path_K(i) = (r_k, s_k)$

end

 In this phase local updating occurs and pheromone is updated

for $K=1$ to m **do**

$$\tau(r,s) \leftarrow (1 - \rho) \tau(r,s) + \rho \Delta \tau(r,s)$$

$r_k = s_k$

end

end

Step 3: In this phase global updating occurs and pheromone are updated

for $K=1$ to m **do**

 Compute V

end

Compute V_{best}

for each edge (r,s) **do**

$$\tau(r,s) \leftarrow (1 - \alpha) \tau(r,s) + \alpha \Delta \tau(r,s)$$

end

if End condition = True **then**

 Print the vertex with highest profit and the path to this vertex

else

 go to step 2

end

end

Algorithm 2: Dynamic iterative algorithm

Bibliography

- [1] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization: artificial ants as a computational intelligence technique," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006. [Online]. Available: <http://dx.doi.org/10.1109/CI-M.2006.248054>
- [2] J. C. Hull, *Options, futures, and other derivative securities*. Englewood-Cliff, N.J.: Prentice Hall, May 2006.
- [3] E. Z. Prisman, *Pricing Derivative Securities: An Interactive Dynamic Environment with Maple V and MATLAB with Cdrom*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.
- [4] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, pp. 637–654, Jan 1973.
- [5] R. Engle, "Autoregressive Conditional Heteroskedasticity with estimates of the variance of U.K. inflation," *Econometrica*, vol. 50, no. 9, pp. 987–1008, 1982.
- [6] J. C. Duan, "Term structure and bond option pricing under GARCH,"

- unpublished manuscript, McGill University, 1996. [Online]. Available: citeseer.ist.psu.edu/duan96term.html
- [7] —, “The GARCH option pricing model,” *Mathematical Finance*, vol. 5, pp. 13–32, 1995.
- [8] J. C. Cox, S. A. Ross, and M. Rubinstein, “Options pricing: a simplified approach,” *Journal of Financial Economics*, vol. 7, pp. 229–263, 1979.
- [9] M. Dorigo, E. Bonabeau, and G. Theraulaz, *Swarm Intelligence: From natural to artificial systems*. Oxford University Press, New York, NY, 1999.
- [10] G. D. Caro, F. Ducatelle, and L. M. Gambardella, “AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks,” *European Transactions on Telecommunications (ETT), Special Issue on Self Organization in Mobile Networking*, vol. 16, no. 5, pp. 443–455, 2005.
- [11] G. D. Caro and M. Dorigo, “AntNet: Distributed stigmergetic control for communications networks,” *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.
- [12] M. Gunes, U. Sorges, and I. Bouazzi, “ARA – the ant-colony based routing algorithm for MANETs,” in *Proceedings of the international conference on parallel processing workshops (ICPPW’02)*, Vancouver, B.C., August 2002, pp. 79–85.
- [13] T. D. Seeley, *The Wisdom of the Hive : The Social Physiology of Honey Bee Colonies*. Harvard University Press, Cambridge, Massachusetts, USA, 1995.

- [14] H. F. Wedde, M. Farooq, T. Pannenbaecker, B. Vogel, C. Mueller, J. Meth, and R. Jeruschkat, "BeeAdHOC: An energy efficient routing algorithm for mobile ad hoc networks inspired by bee behavior," in *Proceedings of Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005, pp. 153–160.
- [15] M. Roth and S. Wicker, "Termite: ad-hoc networking with stigmergy," in *Proceedings of IEEE Global Telecommunications Conference (Globecom 2003)*, San Francisco, USA, 2003, pp. 2937–2941.
- [16] M. Dorigo and T. Stützle, *Ant Colony Optimization*. MIT Press, Cambridge, MA, May 2004.
- [17] A. Brabozan and M. O'Neill, *Biologically Inspired Algorithms for Financial Modelling*. New York: Springer, 2006.
- [18] R. C. Merton, "Theory of rational option pricing," *Bell Journal of Economics*, vol. 4, pp. 141–183, 1973.
- [19] R. K. Thulasiram, L. Litov, H. Nojumi, C. Downing, and G. Gao, "Multithreaded algorithms for pricing a class of complex options," in *Proceedings (CD-RoM) of the IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, CA, 2001.
- [20] R. K. Thulasiram and D. Bondarenko, "Performance evaluation of parallel algorithms for pricing multidimensional financial derivatives," in *IEEE Computer Society Proceedings of the Fourth International Workshop on High Perfor-*

- mance Scientific and Engineering Computing with Applications*, Vancouver, BC, Canada, 2002, pp. 306 – 313.
- [21] K. Huang and R. K. Thulasiram, “Parallel algorithm for pricing American Asian options with multi-dimensional assets,” in *Proc. (CD-RoM) 19th Intl. Symp. High Performance Computing Systems and Applications (HPCS)*, Guelph, ON, Canada, May 2005, pp. 177–185.
- [22] K. Huang, “A parallel algorithm to price Asian options with multi-dimensional assets,” Master’s thesis, Department of Computer Science, University of Manitoba, Winnipeg, MB, CA, 2005.
- [23] J. C. Tannehill, D. A. Anderson, and R. H. Pletcher, *Computational Fluid Dynamics and Heat Transfer*. Second edition, Taylor & Francis, Washington DC, 1997.
- [24] J. Crank and P. Nicolson, “A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type,” in *Proceedings of the Cambridge Philosophical Society*, vol. 43, 1947, pp. 50–64.
- [25] R. K. Thulasiram, C. Zhen, A. Chhabra, P. Thulasiraman, and A. Gumel, “A second order L0 stable algorithm for evaluating European options,” *Intl. J. of High Performance Computing and Networking (IJHPCN)*, vol. 4, no. 5/6, pp. 311–320, 2006.
- [26] P. Carr and D. Madan, “Option valuation using the fast Fourier transform,” *Journal of Computational Finance*, vol. 2, pp. 61–73, 1998.

- [27] S. Barua, R. K. Thulasiram, and P. Thulasiraman, "High performance computing for a financial application using fast Fourier transform," in *Springer LNCS Proceedings of the European Parallel Computing Conference, (EuroPar 2005)*, vol. 3648, Lisbon, Portugal, 2005, pp. 1246–1253.
- [28] S. Barua, "Fast Fourier transform for option pricing: Improved mathematical modeling and design of efficient parallel algorithm," Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, MB, CA, 2004.
- [29] P. Boyle, "Options: A Monte Carlo approach," *Journal of Financial Economics*, vol. 4, pp. 223–238, 1977.
- [30] G. Chen, "Distributed quasi Monte Carlo algorithms for option pricing on HNOWs using mpC," Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, MB, CA, 2006.
- [31] G. Chen, P. Thulasiraman, and R. K. Thulasiram, "Distributed quasi-monte carlo algorithm for option pricing on hnows using mpc," in *ANSS '06: Proceedings of the 39th Annual Symposium on Simulation*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 90–97.
- [32] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. dissertation, DEI, Politecnico di Milano, Italy [in Italian], 1992.
- [33] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

- [34] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, April 1997.
- [35] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant colony optimization: A new meta-heuristic," in *Proceedings of the Congress on Evolutionary Computation*, P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds., vol. 2. Mayflower Hotel, Washington D.C., USA: IEEE Press, 6-9 1999, pp. 1470–1477.
- [36] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 5, pp. 769–778, 1999.
- [37] S. van der Zwaan and C. Marques, "Ant colony optimisation for job shop scheduling," in *Proceedings of the Third Workshop on Genetic Algorithms and Artificial Life (GAAL 99)*, 1999. [Online]. Available: citeseer.ist.psu.edu/vanderzwaan99ant.html
- [38] M. Yoshikawa and H. Terai, "A hybrid ant colony optimization technique for job-shop scheduling problems," in *SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 95–100.
- [39] R. Michel and M. Middendorf, "An ACO algorithm for the shortest common supersequence problem," in *New Ideas in Optimization*, D. Corne, M. Dorigo,

- and F. Glover, Eds. London: McGraw-Hill, 1999, pp. 51–61. [Online]. Available: citeseer.ist.psu.edu/michel99aco.html
- [40] T. Ahmed, “Simulation of mobility and routing in ad hoc networks using ant colony algorithms,” *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, vol. 2, pp. 698–703 Vol. 2, 4-6 April 2005.
- [41] B. Bullnheimer, C. Strauss, and R. F. Hartl, “An improved ant system algorithm for the vehicle routing problem,” in *Annals of Operations Research*, vol. 89, 1999, pp. 319–328.
- [42] M. T. Islam, P. Thulasiraman, and R. K. Thulasiram, “Implementation of ant colony optimization algorithm for mobile ad hoc network applications: OpenMP experiences,” *Parallel and Distributed Computing Practices Journal*, vol. 2, pp. 61–73, 2004.
- [43] M. T. Islam, “Design, implementation and performance analysis of the ant colony optimization algorithm for routing in ad hoc network,” Master’s thesis, Department of Computer Science, University of Manitoba, Winnipeg, MB, CA, 2004.
- [44] M. T. Islam, P. Thulasiraman, and R. K. Thulasiram, “A parallel ant colony optimization algorithm for all-pair routing in MANETs,” in *Proceedings (CD-ROM) of the IEEE Computer Society Fourth IPDPS workshop on Parallel and Distributed Scientific and Engineering Computing with Applications (PDSECA-2003)*, Nice, France, April 2003.

- [45] E. S. Osagie, "An Evaluation of an Ant Colony Optimization algorithm for MANETs using simulation (thesis in progress)," Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, MB, CA, 2006.
- [46] J. Wang, "HOPNET: A Hybrid ant colony OPTimization routing algorithm for Mobile ad hoc NETwork (thesis is progress)," Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada, 2007.
- [47] L. M. Gambardella and M. Dorigo, "An ant colony system hybridized with a new local search for the sequential ordering problem," *INFORMS J. on Computing*, vol. 12, no. 3, pp. 237–255, 2000.
- [48] S. Rahmail, I. Shiller, and R. K. Thulasiram, "Different estimators of the underlying asset's volatility and option pricing errors: parallel Monte-Carlo simulation," in *Proceedings of the International Conference on Computational Finance and its Applications (ICCFA)*, Bologna, Italy, 2004, pp. 121–131.
- [49] C. Keber and M. G. Schuster, "Generalized ant programming in option pricing: Determining implied volatilities based on American put options," in *Proceedings of the IEEE International Conference on Computational Intelligence for Financial Engineering*, Hong Kong Convention and Exhibition Centre, Hong Kong, March, 2003, pp. 123–130.
- [50] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, Mass.: MIT Press, 1992.
- [51] L. M. Gambardella and M. Dorigo, "Solving symmetric and asymmetric TSPs by

- ant colonies,” in *International Conference on Evolutionary Computation*, Nayoya University, Japan, 1996, pp. 622–627.
- [52] S. Kumar, G. Chen, R. K. Thulasiram, and P. Thulasiraman, “Pricing derivatives using ACO algorithm,” in *Midwest Instruction and Computing Symposium [CD-ROM]*, Grand Forks, ND, USA, April, 2007.
- [53] M. Randall and A. Lewis, “A parallel implementation of ant colony optimization,” *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1421–1432, 2002.
- [54] P. Delisle, M. Krakecki, M. Gravel, and C. Gagné, “Parallel implementation of an ant colony optimization metaheuristic with OpenMP,” in *International Conference of Parallel Architectures and Compilation Techniques, Proceedings of the Third European Workshop on OpenMP*, Barcelona, Spain, September 2001, pp. 8–12.
- [55] W. Gropp, A. Lusk, and A. Skjellum, *USING MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, Mass.: MIT Press, 1994.
- [56] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*. San Francisco, CA, USA: Morgan Kaufmann, 2001.
- [57] P. Delisle, M. Gravel, M. Krajecki, C. Gagn, and W. L. Price, “Comparing parallelization of an ACO: Message passing vs. shared memory.” in *Hybrid Metaheuristics*, ser. Lecture Notes in Computer Science, M. J. Blesa, C. Blum,

A. Roli, and M. Sampels, Eds., vol. 3636. Springer, 2005, pp. 1–11. [Online].

Available: <http://dblp.uni-trier.de/db/conf/hm/hm2005.html#DelisleGKGP05>