Ant Colony Meta-heuristics –

Schemes and Software Framework

Lim Min Kwang
(B.Eng (Computer Engineering) (Hons I), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2003

ACKNOWLEDGEMENTS

I would like to express sincere thanks to Dr. Lau Hoong Chuin for his kind supervision, insight, and contributions to this thesis, as well as work related, without whom the accomplishment would have took a much longer. Thanks are due too to Mr. Wan Wee Chong and the folks in the Computational Logistic Lab (CLL) as well as the several of the research engineers at TLI-AP for several collaborative efforts of this thesis. Also, I would take this opportunity to expresse gratitude to Ms. Loo Line Fong and Ms. Agnes Ang who handled the administrative aspects for my program.

Academic facets aside, this work would not be possible without support from the managers at Cisco Systems, Inc. for their support of the research relating to this thesis. Also, I like to thank my family and Ms. Yeo Shu Ern for their support and understanding for the time that was spent on the computer working overtime on this thesis, instead of being with them. I really like the snacks that seem to appear besides my computer periodically.

And finally to everyone who helped, support, and kept my spirits and motivations high. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Ant Colony Optimization (ACO) was first proposed as Ant System (AS) by [Dorigo et al., 1991], revised in [Dorigo et al., 1996], as a strain of swarm intelligence algorithm, exploiting the foraging behavior of ants. Ants individually are sub-intelligent species, but share information using a chemical called *pheromone* that allows the colony to seek out optimal amount of food. ACO had efficiently been utilized to solve many NP-hard problems such as the Quadratic Assignment Problem (QAP) [Gambardella et al.^{2}, 1999], Traveling Salesman Problem (TSP) [Dorigo et al., 1991; Leguizamon and Michalewicz, 1999; Stutzle and Dorigo, 1999], Knapsack Problem (KP) [Fidanova^1, 2002; Fidanova^2, 2002], as well as more complex problems extended from these problems. The nature of the algorithm is such that it is extremely suited to solve assignment type problems, commonly a feature of combinatorial and assignment optimization problems. However, for complex problems with increasing number of constraints, ACO by itself tend to be less powerful, due in part to redundant solution construction cycles. Hence, most implementation of ACO in the literature either breaks down a complex problem into smaller parts, or integrates ACO with another local search heuristics, most commonly Tabu Search, to achieve results that perform better than the individual component algorithms. This motivates a need for an ACO software framework. This forms the primary objective of this thesis. By examining the logic and operation of ACO, a C++ software framework is proposed that is capable of implementation by itself, or integrating with other heuristics software framework via a higher level meta-heuristics framework. In particular, the concept of reuse is essential, to exploit the similarities of many problems, particular those extended from simpler problems. It is demonstrated how instance implementations for the Vehicle Routing Problem with Time Window (VRPTW), using generic Traveling Salesman Problem (TSP) implementations, are solved, and further extended to solve the Inventory Routing Problem with Time Window (IRPTW) with promising results. Aside from the software framework, the secondary objective of the thesis explores various factors of the ACO algorithm that are exploitable to achieve efficient results for complex problems, such as multi-period scheduling problems like IRPTW or the Multi-Period Multi-Dimensional Knapsack Problem (MPMKP). This allowed the development of an ACO scheme that specializes in handling the third dimension of most extended problems – time. Results from solving the various problems (VRPTW, IRPTW, and MPMKP) are then presented to prove the case. It is demonstrated how the framework allowed reuse which saved development time, yet providing excellent results by extending implementations solving VRPTW to the IRPTW; and the results for MPMKP showed the effectiveness of the proposed ACO scheme that is good at handling time notation problems.

CHAPTER 1

Introduction

The humanly instinctive solution to any arbitrary search or optimization problems would be an exhaustive brute-force approach. The discovery of the notion of non-deterministic polynomial (NP) completeness in complexity theory [Garey and Johnson, 1979] unveils the property that there are many NP-hard search or optimization problems whose solutions are easy to verify in polynomial time, but computationally intractable to find. Brute-force approach is not feasible in such instances under the existing von-Newman machine model. This motivates the development of intelligent exact methods able to achieve good results in efficient time.

However, exact methods, while ensuring optimality, are often not feasible or practical when solving NP-hard problems, especially those of large problem size. This led to the development of meta-heuristics, which manages approximate methods (heuristics), to search for optimal solutions. Such approaches have been developed to achieve very good results for solving NP-hard problems in record time, making industry application, in particular in the field of logistics, very efficient. However, most of the best performing meta-heuristics to date had been algorithms of the optimization phase using a two-phase approach (construction phase and local improvement phase). This motivates a need for a good construction phase algorithm.

In this research of meta-heuristics, the community has recently progressed to a new age. The observation of nature has yielded many interesting algorithms adept at solving problems of many types. Nature has many creatures which have existed for a long time, based on their ability to survive. In particular interest are creatures of extremely low intelligence, but yet are able to persist. One such survivor is the ant, which date back at least 92 million years [http://www.antcolony.org/oldest_ant.htm].

Ant Colony Optimization (ACO) was first proposed by Marco Dorigo in his PhD thesis [Dorigo et al., 1991]. It was originally used to solve hard combinatorial optimization problems like the Quadratic Assignment Problem (QAP), and the Traveling Salesman Problem (TSP). ACO is also capable of solving dynamic problems such as network flow in an environment like the Internet. For instance, [Schoonderwoerd et al., 1997] developed an ACO algorithm called *ABC* for routing and load balancing in circuit switched telecommunications networks, and [Di Caro and Dorigo, 1998] proposed *AntNet*, another ACO algorithm applied to routing in packet switched telecommunications network. As a developing meta-heuristics, ACO swiftly achieved recognition when it was shown to be able to achieve excellent results for many other problems like the Vehicle Routing Problem with Time Window (VRPTW) and the Multi-Dimensional Knapsack Problem (MKP).

ACO is inspired by the foraging behavior of an ant colony. ACO is a particular class of meta-heuristics derived from nature, amongst other categories that include evolutional algorithms, neural networks, and simulated annealing. In particular, ACO is a type of swarm intelligence algorithms that had been gaining popularity. For instance, there are algorithms of similar classes that follow the behavior of bees (Particular Swarm Optimization) [Parsopoulos and Vrahatis, 2002], bird flocks (Squeaky Wheel Optimization) [Joslin and Clements, 1999], and even mammals like lab rats [Yufik and Sheridan, 2002]. ACO, however, proves to be more welcomed in the community, in part due to the algorithm being naturally intuitive, easy to understand and implement, but mainly because of the powerful results that have been obtained, especially in collaboration with other established local search algorithms like Tabu Search (TS), Genetic Algorithm (GA), Simulated Annealing (SA), etc., for more complex problems. This is especially true of combinatorial and assignment type problems like TSP, QAP and the sequential ordering problem, where ACO outperform all known algorithms on the majority of classes in benchmark problems. It should, however, be noted that there are differences in the operations of the algorithm as opposed to the way real ants work. Regardless, this rest of this thesis will stick to the terminology to properly address the community.

Colonies of real ants uses a chemical substance called *pheromone* as they traverse to and from the nest and food sources. They lay a higher concentration of pheromone on trails which have a correspondingly better quality food (as defined by the ants themselves, like more appropriate food type for ants, or larger quantities, etc.). Instinctively, the better quality food sources would have more ants traversing the route in between. As such, the pheromone laid on the trail would be of a higher concentration then otherwise. This pheromone is also the cause of why troops of ants tend to travel in a trail rather than haphazardly, which happens only when an ant is *exploring* for food. For most of the time, an ant would *exploit* a trail with a higher concentration of pheromones, since it is more probable to lead to better food. This pheromone will also evaporate into the air as time passes, and the ants will need to constantly reinforce the pheromone. As the quality of the food sources diminished, lesser ants will travel on the route, and hence lesser pheromone.

Using the pheromone trail as an inspiration for the communication medium, ACO is developed which allowed individual ants – corresponding to a single optimization agent – to have simple intelligence (and hence simple to understand

and implement), but be able to share information which allowed a synergistic effect for the entire colony. By exploiting the power of probability common in many successful meta-heuristics, ACO allows individual agents to *explore* and *exploit* the pheromone trails, using the pheromone trail to construct solutions of good quality. The soundness in the logic lies in that the ants will have heavy pheromone concentration on the best found solution thus far, and then search around the neighborhood of this best found solution, which tend to yield improving results. However, this usually leads to local optimal which traps the search process, as in many other meta-heuristics. The ACO algorithm then deviates from how biological ants work by proposing new modifications beyond the ants metaphor. Techniques are introduced in the optimization community, such as the concept of using local decay to diversify the search, is not what real ants do, among other proposals. Hence, technically, real ants do not operate as "optimally", and there is a wide variant, as well as arguable ground on how close the algorithm works according to its original inspiring creature.

These modifications arises because the initial proposal of ACO, Ant System (AS), while yielding encouraging initial results, could not compete with the best algorithms in the community for most problems like the TSP. However, the simplicity but potential of the idea stimulated research that added many variants which had to date been used to solve many benchmark problems with good performance.

The recent interest in the community regarding the developing ACO motivates this thesis. For instance, there are many specific conferences dedicated to ACO [http://iridia.ulb.ac.be/~mdorigo/ACO/conferences.html], as well as many publications about ACO in logistics and optimization conferences and journals.

Also, due to the nature of randomized solution construction by each ant in an ACO iteration, the algorithm can be seen as a type of traditional construction heuristics in a typical two phase heuristics. Furthermore, ACO by itself is also quite capable of performing optimization, using the pheromone trails. Such flexibility suggests an excellent potential for collaboration with other strains of ACO, or with other local search heuristics which complements ACO well, and motivates the development of a software framework.

As such, the primary objective of the thesis is to present a software framework that provides the guideline for a developer to easily implement ACO. There had been ACO *algorithm* framework proposals [Dorigo and Di Caro, 1999; Blum and Dorigo, 2001], but no popular *software* framework on ACO. This is in part due to the difficulty to create a software framework that is to become popular, as apparent from the lack of such framework. As such, this framework should also be generic enough that it allows easy integration and collaboration with other meta-heuristics, as the most powerful results in the literature are usually those of hybrid techniques.

Another key factor in the software framework would be re-use, such that implementation for one problem can be easily re-cycled to solve similar or extended problems. Reusability will significantly reduce the costs of development and research. These will contribute to the success of any meta-heuristics framework. By allowing ease of use, easy reusability, and easy collaboration with any other arbitrary technique, the thesis hopes to present a framework that will be a base standard in ACO development. In particular, this thesis demonstrates a powerful collaboration with the existing Tabu Search Framework (TSF++) [Wan, 2002; Lau et al.[1], 2003].

Furthermore, the simplicity in which ACO works further inspires research for this thesis. Each agent (artificial ant) has only primitive intelligence (a basic if-else decision making most of the time, such as “to go here, or there”), but made use of a shared medium to communication, thereby achieving synergistic effect that can beat the results achieved by another intellectively superior species. The thesis hopes to explore certain properties of ACO that makes it powerful, and present schemes that exploit these properties that is adept at solving different classes of problems. For instance, ACO had also been very successful in solving problems that had extended constraints. There are variations of ACO that had solved the VRPTW with top results in the literature [Gambardella et al.[1], 1999]. Similarly, we see ACO proposals in the literature solving the MKP [Fidanova[1], 2002; Fidanova[2], 2002; Leguizamon and Michalewicz, 1999]. Respectively, VRPTW is derived from VRP, which in turn is derived from TSP, with additional constraints, as such time window; while MKP is a generalization of the 0-1 Knapsack Problem (KP). In the analogy of geometry, these extended problems have constraints that correlate in the dimension of space (second dimension). This thesis proposes schemes for ACO that is conducive to solving a further generalized form of these generic problems, which even more closely corresponds to practical scenarios in the industry – time period. This dimension correlates to the dimension of time in geometry. Examples of such problems are the Inventory Routing Problem with Time Window (IRPTW), extended from the VRPTW; and the Multi-Period Multi-Dimension Knapsack Problem (MPMKP), both which are examined in this thesis.

The rest of this thesis is organized as follows. In Chapter 2, we review related works in the literature. Chapter 3 provides further background and mathematical formulations for the some problems that are examined in this thesis.

Chapter 4 provides a detailed examination into the operations of the ACO meta-heuristics, and how it has the potential to adapt to a software framework. Chapter 5 demonstrates how the actual ACO software framework is designed, with relations to a generic meta-heuristics development framework comprising of other algorithms as well. Chapter 6 focuses on the secondary objective of the thesis and presents several ACO schemes that can be exploited to improve results according to the problems to be solved. Chapter 7 examines the hybridization of ACO with Tabu Search and the solution approach to obtain the results to the benchmarks problems presented in Chapter 8, which also provides discussions on the results obtained. Chapter 9 concludes the thesis.

CHAPTER 2

Literature Review

This chapter examines the works in the literature that leads to and including the development of ACO. The first section looks at traditional problem solving approaches – exact methods – prior to the introduction of meta-heuristics. The second section then delves in detail on heuristics and meta-heuristics. This leads to the class of meta-heuristics that follow observations of nature, of which include ACO, where related works are presented in the third section. The chapter then concludes with a section on current meta-heuristics software framework in the community.

2.1 Traditional Approaches

Exact methods are guaranteed to find optimal solution, and for problems of polynomial complexity, optimality can be achieved in polynomial time. Some popular fundamental exact methods include divide and conquer [Bentley, 1980; Knuth, 1968; Knuth, 1973], branch and bound [Narendra and Fukunaga, 1977; Nemhauser and Wolsey, 1988; Zhang, 1993; Crowder and Padberg, 1980], cutting plane [Applegate et al., 1995; Bahn et al., 1994; Elhedhi and Goffin, 2001; Padberg and Grotschel, 1995; Padberg and Rinaldi, 1987; Padberg and Rinaldi, 1990; Fleischmann, 1985], branch and cut [Agarwal et al., 1989; Araque et al., 1994; Augerat et al., 1995; Crowder and Padberg, 1980; Grotschel and Holland, 1991; Junger and Stormer, 1995; Lin and Kernighan, 1973; Naddef and Rinaldi, 2000; Ralphs, 2003; Ralphs et al., 2003], and dynamic programming [Bellman, 1957].

Unfortunately, many important combinatorial optimization problems are NP-hard in nature. The theory of computational complexity [Garey and Johnson, 1979; Papadimitriou, 1994] present a rich collection of such problems. It is well-known that for NP-hard problems, exact methods take an exponential amount of computational resources in the worst case, which renders them impractical for large-scale instances. Heuristics method then became the feasible way to solve complex problems. A heuristics approach tends to be significantly faster and provide good solutions. The downside, however, is that they do not guarantee optimality. But for most practical application, especially real-time situations, it is usually sufficient to obtain near optimal results in the shortest time possible.

2.2 Heuristics Approaches

The feasibility of heuristics approach in solving hard problems is a key factor in their widespread popularity in the literature, as they strive to discover near optimal, and sometimes optimal, solutions to hard problems in record time. The word "heuristics" arises from the Greek verb *heuriskein*, meaning "find" or "discover". Heuristics in current context means "rules of thumb" or techniques that improve the average-case performance of a problem-solving task.

Meta-heuristics are a further development from heuristics. It literally means "heuristics for managing heuristics", and controls the collaboration of one or more heuristics, searching for a better solution than any single heuristics. Meta-heuristics focus on *what* makes a good solution, rather than *how* to find a good solution. By induction, meta-heuristics algorithms are well-suited to solving hard problems too complex or time-consuming to solve using traditional approaches, since it is easier to define *what* makes a good solution then *how* to find one, especially for NP-hard problems. This is especially useful in the earlier part of the solution search to reduce the scope of the problem to feasible size (sometimes even small enough to be solve using exact methods).

2.2.1 Two-phase approach

Classically, most meta-heuristics are either construction algorithms, optimization algorithms, or a mix of both (two phase approach), (e.g. [Bentley, 1980; Gehring and Homberger, 2001; Schulze and Fahle, 1999]). The two types of approaches are considerably different. Construction algorithms work on empty or partial solution and try to extend them in the best possible way to complete problem solutions, while optimization algorithms work on an already completed solution and look around the solution space trying to upgrade the quality of the current solution. Thus far, better results are achieved with optimization algorithms, mostly in conjunction with problem-specific construction algorithms like Tabu Search, Simulated Annealing, Genetic Algorithm, etc. However, these algorithms require an initial solution, usually provided by construction algorithms like variations of Greedy Algorithm adapted to the problem being solved.

2.2.2 Meta-heuristics

A meta-heuristics is as flexible as the ingenuity of the included heuristics, but there are several more popular characteristics of meta-heuristics such as hill-climbing techniques, iterative improvement heuristics, and knowledge-based search methods. These form the basis of many more advanced techniques, including guided search like Simulated Annealing (SA) which is inspired from the way metal cools; procedures of temporary elimination of backtracking like Tabu Search (TS), and algorithms based on principles of nature and biological evolution, which included ACO focused in this thesis, amongst many other methods. These are examined in the next subsection.

By definition, a meta-heuristics can include hybrids of one or more of any arbitrary heuristics, meta-heuristics and exact methods. It is observed that many of the best solvers for classic hard problems tend to be hybrid methods, such as the approaches proposed in [Bent and Hentenryck, 2001; Lau et al.^{2}, 2003; Gambardella et al.^{1}, 1999; Gambardella et al.^{2}, 1999]. This arises from the concept that each algorithm has their forte and weakness, and it is easy to exploit each algorithm separately to achieve optimal effect. The remainder of this section provides background on some of the various individual approaches.

Hill-climbing (a.k.a Greedy Algorithm)

Hill-climbing techniques in general search for better solutions when given an initial solution as a starting point, and are a key ingredient in many other meta-heuristics. However, it has the downside of being easily trapped in local optimal. For instance, consider Figure 2.1.

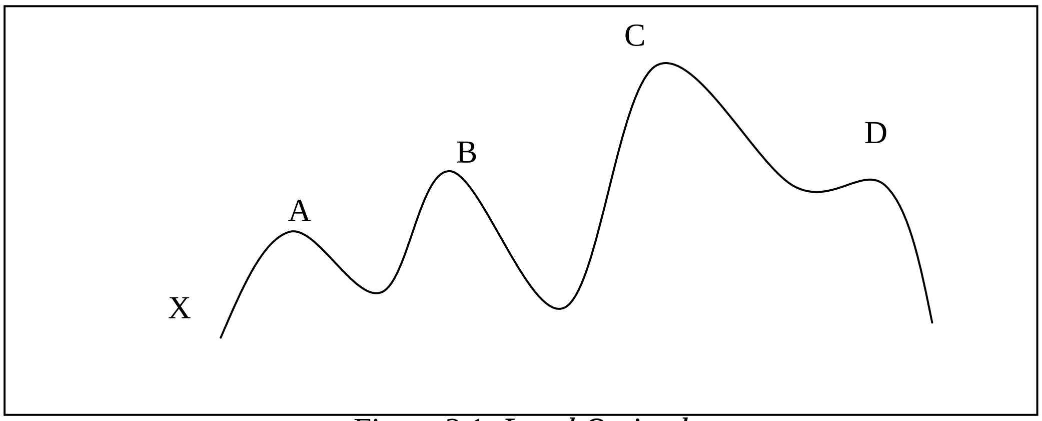


Figure 2.1: *Local Optimal*

Point A, B and D are local optimal, and point C is the global optimal. Suppose the search start at point X. In most hill-climbing heuristics (in fact, in most other heuristics), depending on the function of the graph, the search will end at point A, since the search will have no better knowledge of the existence of B, C or D. On reaching point A, the function will see only decreasing values no matter which way it goes, and hence concluded erroneously it has find optimality. Examples of hill-climbing approaches included [Distante and Piuri, 1989; Tomov, 1994].

Iterative Improvement

Iterative improvement heuristics, like hill-climbing, is also another fundamental technique applied by many other heuristics. Techniques such as bottleneck reduction [Chakradhar and Raghunathan, 1997] improve solutions through repeated application of iterations. Iterations allow control over how long to repeatedly execute an algorithm. For instance, most of the best meta-heuristics to date, like Tabu Search and ACO, uses repeated (similar) iterations to continuity attempt to improve the solution. With the inclusion of a variable factor, like the pheromone trail in ACO, iterations simplify code and implementation, while allowing powerful performance. [Dorn et al., 1994] provides a comparison of several iterative improvement techniques for scheduling optimization.

Knowledge-based search

Knowledge-based search method is also another popular and fundamental building block in meta-heuristics, such as the approach in [Jin and Reynolds, 2000] used to guide evolutionary search. This characteristics focus on one or more attributes of candidate solution which is defined to be more important than others, and search future solution based on this interest. For instance, in the more popular version of the classic VRPTW, it is more important to focus on reducing the number of vehicles rather than total distance traveled.

2.2.3 Guided search

This subsection examines three popular guided search techniques employed in the literature. In particular, simulated annealing (SA) is first examined; followed by Tabu Search, a local search with memory; and finally algorithms that arises as a result of biological evolution (Genetic Algorithm) and observations of nature, of which ACO has relations to.

Simulated Annealing

Simulated annealing (SA) guides the search for good solution by allowing solutions of lower quality to be temporarily qualified. SA is inspired from the way metal cools and freezes into a minimum energy element (annealing process). SA is based on the work of [Metropolis et al. 1958], who originally proposed presented SA as a method of finding the equilibrium configuration of a collection of atoms at a given temperature. [Pincus, 1970] discovered the connection between the algorithm of Metropolis and the mathematical minimization, which led [Kirkpatrick et al., 1983] to proposed SA as the basis of an optimization technique for combinatorial problems. A feature that sets SA aside from the previously mentioned techniques in this section is its ability to avoid being trapped in local optimal. As mentioned, SA can accept lower quality solution temporarily, and if referencing Fig 3.1 again, supposed the solution has reached point A, SA may allow the search to continue down the graph towards point B. If the search passes the minimum point between point A and B, the normal hill-climbing approach would bring it to the higher point B. The key issue here is to know how much lower quality ("cooling schedule") the search can accept.

Tabu Search

Tabu Search is another powerful meta-heuristics, proposed by Glover [Glover and Laguna, 1997], with many hybrids and techniques. There are several TS implementation with excellent results in solving classic problems, such as those seen in [Rochat and Taillard, 1995; Taillard et al., 1997]. The overall concept of TS differs from SA and GA (Genetic Algorithm, next sub-section), which can be classified as "memory-less", in that it relies on memory to avoid entrapment in cycles, by forbidding or penalizing (*tabu*-ing) moves that takes the solution in the next iteration to points in the solution space previously visited. It will be seen later that the local decay feature of ACO follows this concept to a less zealous extend. Besides the adaptive memory, TS also advocate responsive exploration, which adds certain degree of intelligence by giving the search the ability to response to differing events. TS employ search strategies which attempt to exploit the key mechanism of adaptive memory and responsive exploration. [Aboudi and Jornsten, 1994; Dammeyer and Voss, 1993] provides more works related to Tabu Search.

Biological Evolution (Darwinian Theory)

Another recent trend in the logistics and optimization research dwells on algorithms based on principles of nature or biological evolution. Darwinian Theory [Darwin, 1979] of "the survival of the fittest" and natural selection had inspired techniques such as Genetic Algorithm (GA), in the aim that the survivor in nature must follow some optimal or near optimal way (or at least better than the non-survivors) to sustain existence. Approaches following the concept of biological evolution, like GA, are powerful problem-solving methods in which a population of candidate solutions “evolves” to get better and better, much like a creature adapting to environment. As [Mangano, 1995] summarized, “Genetic Algorithm are good at taking large, potentially huge search spaces and navigate them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime”.

Observations of Nature

On the other end of this class of algorithms are those following observations of nature, in particular swarm creatures like ants [Dorigo and Di Caro, 1999], bees [Parsopoulos and Vrahatis, 2002], termites [Bonabeau et al., 1997], and mammals like birds flocks [Joslin and Clements, 1999], and rats [Yufik and Sheridan, 2002]. Of particular interest and performance are the many sub-classes of swarm intelligence (SI) [Bonabeau, 1999; Bonabeau and Theraulaz, 2000; Hoffmeyer, 1994; Ward, 1998]. SI is a system whereby the collective behaviors of low intelligence agents interacting locally with their environment cause coherent functional global patterns to become apparent. This allows collective or distributed problem solving without centralized control. The particular subclass of swarm intelligence dealt with in this thesis is the behavior of ants.

2.3 Ant Colony Optimization

Social insects such as ants, bees, termites and wasps exhibit a collective problem solving ability [Deneubourg and Goss, 1989; Bonabeau et al., 1997], with particular interest found in the activity of several ant species which are capable of selecting the shortest pathway between their nest and food sources [Berkers et al, 1990]. [Dorigo et al., 1991] introduced the Ant System based on this idea to solve TSP. It was further applied to many other problems like the Job Shop Scheduling Problem [Colorni et al., 1993], Graph Coloring Problem [Costa and Hertz, 1997], Quadratic Assignment Problem [Maniezzo et al., 1994], and also to dynamic problems like network flow across a changing environment like the Internet [Schoonderwoerd et al., 1997].

Ants lay a chemical (called pheromone) trail as they travel, which attracts other ants to follow the same path. The amount of pheromone laid depends on the distance of the trail. Intuitively, there will be a higher concentration of pheromone on shorter/easier trail. With time, the pheromone trail will also evaporate into the air, which allows food sources which had diminished or expired to gradually be ignored. [Dorigo et al., 1991; Dorigo et al., 1996; Dorigo et al., 1999] formulated and developed the ACO taking advantage of the concept of pheromone trail. While able to find solutions fast, the pheromone evaporation also avoided early convergence to low quality solutions, yielding excellent results for many classical NP-hard combinatorial optimization problems.

ACO is an adaptive algorithm, and its most powerful implementation so far had been in collaboration with other techniques, in part due to its capability to be a construction algorithm, but in part also due to certain pitfalls in the algorithm which tend to cause solution cycling with increasing iterations. For instance, many of the more powerful ACO proposals included a “local search” component, suggesting a hybrid approach. In particular, these “hybrids” had been used to solve hard complex problems with excellent results such as that of [Gambardella et al.[1], 1999] for

VRPTW and [Lau et al.^{2}, 2003] for IRPTW, while GA had been proposed by [Goldberg, 1989] to evolve the usually hand-tuned ACO parameters to fine-tune them for different problem types and instances.

To date, the results achieved by ACO had proven it to be more than just another algorithm. Research Institutes like IRIDIA [http://iridia.ulb.ac.be/~mdorigo/ACO/about.html] currently focus much of their research on ACO and its derivation, and there are many conferences dedicated to ACO only [http://iridia.ulb.ac.be/~mdorigo/ACO/conferences.html], as well as special proceedings and tracks from other acclaimed logistic and optimization conferences.

2.4 Software Framework

Currently, the community sees a lack of popular and effective meta-heuristics software framework. A few more recognized or ongoing works included OpenTS Framework for Tabu Search, EASYLOCAL++, HOTFRAME, Localizer, TSF++, among others.

OpenTS was the result of an initiative by the Common Optimization Interface for Operations Research (COIN-OR) group in IBM. OpenTS is a Java-based application following an Object-Oriented (OO) style inherent in the Java language proposed by [Harder, 2001]. However, OpenTS is limited strictly to Tabu Search, and it is complex to integrate other techniques into the engine.

EASYLOCAL++ is another OO framework proposed by [Gaspero and Schaerf, 2000], but in C++, for the development and analysis of local search algorithms. It is more generic than OpenTS in that it integrates many local search techniques like TS, SA, and local search. ACO, however, is not part of

EASYLOCAL++. EASYLOCAL++ and OpenTS, however, both do not provide some form of centralized control mechanism nor library to support development of advanced search strategies.

To acknowledge this problem, [Fink et al., 1998] proposed the Heuristics OpTimzation FRAMEwork (HOTFRAME) to provide a set of ready-to-use software components for heuristics search, in addition to a reliable architecture for placing the components. HOTFRAME is an ongoing project and to-date had variants of Tabu Search and Simulated Annealing. However, HOTFRAME is somewhat complex and documentation is not readily available currently.

Less relevant to the framework context, but related nonetheless, [Michel and Van Hentenryck, 1999] proposed an attempt to support the implementation of local search via the use of a modeling language Localizer close to the informal descriptions in scientific papers. While providing ease of configuration, a user must provide Localizer the formulation, albeit of any local search, to construct an algorithm. This differs from the work afore-mentioned which provides a constructed framework, whereby the usage is simpler since the user only had to implement specific (and algorithm specific) interfaces/classes.

Another framework that provides centralized control is TSF++ [Wan, 2002]. By allowing flexible components and an architectural base, as well as an easily extensible library of components, TSF++ proved effective for most situations. However, like OpenTS, TSF++ comprises only of Tabu Search and currently is under development to become a sub-component of a higher level framework, of which the ACO framework proposed in this thesis is another sub-component.

Furthermore, while there had been ACO algorithmic framework [Dorigo and Di Caro, 1999; Blum and Dorigo, 2001], the community appears to lack a popular ACO software framework, in part due to the variation of ACO that can be implemented. As a developing algorithm, an ACO framework needs to be flexible to allow different schemes, both present and still un-thought of, that exploits properties of the algorithm, without limiting it.

CHAPTER 3

Problem Definition

This chapter presents the formulation and mathematical interpretation for the classical problems examined with regards to this thesis. In particular, the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (with Time Window) (VRPTW), the Inventory Routing Problem (with Time Window) (IRPTW), the Multi-Dimensional Knapsack Problem (MKP), and the Multi-Period Multi-Dimensional Knapsack Problem (MPMKP) are considered.

The choice of TSP, VRPTW, and IRPTW is a generalization of many real-world optimization problems, which tend to have multiple objectives and constraints. For instance, the IRPTW considers inventory costs across multiple period of VRPTW, which in turn is the VRP extended with time window, which in turn is extended with optimal fleet (vehicles) size objective from the classic and NP-hard TSP. Similarly, the MPMKP is MKP fixated with multiple periods, and the MKP is an extended case of the 0-1 Knapsack Problem. The extensions of NP-hard problems with more constraints and objectives provide increasing approximate analogy to practical application, increasing the value of solving these problems optimally. As such, these problems are chosen to demonstrate the power of re-use in the framework in solving similar or extended instances of a problem. By logic of induction, the ACO framework would be applicable for re-use in other problems as long as a solution can be formulated for the base problem.

Furthermore, the thesis looked at the more complex IRPTW and MPMKP, both of which are multiple time-period problems. As computing power and algorithmic strength improve, the community had gradually shifted to increasingly complex problem instances, and one current trend is a focus on the "third-dimension" extensions of multiple time-periods with constraints binding between time periods. By providing an ACO solution to both problems, this thesis also presents a scheme of ants conducive to solving problems with time-period constraints.

3.1 Traveling Salesman Problem (TSP)

The Traveling Salesman Problem is a classic NP-hard problem, and the mathematical basis related to TSP was treated as earlier as the 1800s by the Irish mathematician Sir William Rowan Hamilton and the British mathematician Thomas Penyngton Kirkman. [Held and Karp, 1969] provides a look at the TSP and the related Capacited Minimum Spanning Tree, while [Biggs et al., 1976] provides discussion on the works on the afore-mentioned mathematicians. The development of the general form of TSP, as well as other classic combinatorial optimization problems, is studied by [Schrijver]. While the problem was well-known, there appears a lack of reference in the literature to earlier work, and it was not until 1954 that the most popular TSP definition came from [Dantzig et al., 1954].

TSP definitions for general and variant forms of the problems are easily available. In the context of this thesis, TSP is defined as follows:

Let

$G = (V,A)$ *be a graph,*

where $V\{ v_1, v_2, \ldots , v_n \}$ *be a set of cities* (*vertex set*), *and*

$A = \{ (v_i,v_j) : v_i,v_j \in V, i \neq j \}$ *be the edge set,*

$C(r,s) = C(s,r)$ *be a cost measure associated with edge* (r,s) *w.r.t. A.*

A tour is defined as a *Hamiltonian* circuit passing exactly once through each point in *V*. The TSP objective is to *find a tour of minimum costs/distance*.

For the interested reader, full historical mathematical formulations of TSP can be found at [http://rodin.wustl.edu/~kevin/dissert/node11.html], and [Finke et al., 1984; Lawler et al., 1985; Naddef and Rinaldi, 1991; Naddef and Rinaldi, 1993] provides more readings on the problem.

3.2 Vehicle Routing Problem (with Time Window) (VRPTW)

The Vehicle Routing Problem [Toth and Vigo, 2002] is a generic class of complex combinatorial optimization problems extended from the TSP and the Bin Packing Problem (BPP), and was first formulated by [Dantzig and Ramser, 1959]. The VRP is a generalization of the TSP, with additional *m* constraints, the *m*-TSP, inductively making VRP NP-hard. Inversely, the TSP is the VRP with one un-capacitated vehicle (which is the elementary version of VRP, the Capacitated Vehicle Routing Problem – CVRPT), no depot, and customers with no demand. Such observation inspired some approach to solving VRP using a divide and conquer method to break VRP into several Multiple TSP (MTSP, a TSP with *m* identical duplicated origin and *m* salesman) (e.g., [Bullnheimer et al., 1997]). VRP and its variations had been well examined and solved using various techniques from exact methods (e.g., [Baldacci et al., 1999; Balinski and Quandt, 1964; Christofides and Eilon, 1969; Christofides et al., 1981; Cook and Rich, 1999; Cullen et al., 1981;

Fisher, 1988; Fisher and Jaikumar, 1981; Foster and Ryan, 1976;]), to heuristics and meta-heuristics (e.g., [Braysy, 2001; Chiang and Russell, 1997; Cordeau et al., 2000; Gillet and Miller, 1974; Rousseau et al., 1999]).

A popular and important variant to the VRP, the Vehicle Routing Problem with Time Windows (VRPTW), introduce additional constraints to the original definition, specifying that each costumer must be served within a specific time window. Other variants of the problem are multi-depot, fixed routes, fixed areas, etc. Such variants are formulated as they better approximate practical scenarios.

This thesis in particular looks at VRPTW, which is defined as follows:

Let

$G = (V, A)$ *be a graph,*

where $V = \{v_0, v_1, \dots, v_n\}$ *is the vertex set, and*

$A = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ *is the edge set.*

This definition is similar to the TSP definition. The difference is in the additional constraints. The depot vertex v_0, has m identical vehicles, each with a maximum load capacity Q and a maximum route duration D. The remaining vertex $v_i \in V$ represent customers to be serviced, each with a non-negative demand q_i, a service time s_i, and a service time window comprised of a ready time r_i and a due time l_i. A waiting time w_i is incurred if customer i is serviced before its ready time. Each edge (v_i, v_j) has an associated non-negative $cost_{ij}$, interpreted as the travel time t_{ij} between location i and j. A complete tour is defined by the order in which the n customers are serviced by m vehicles, and the objective of VRPTW is *to determine a complete tours starting and ending at the depot, such that each customer is visited exactly once within its time window, the total demand of any vehicle route does not* *exceed Q, the duration of any vehicle route does not exceed D and the total cost of all routes is minimized.*

Due to the number of constraints in the problem, there are many definitions on the problem optimality. A widely debated factor is whether to consider distance or number of vehicles as the primal optimality factor, with more researchers focusing on the latter as the primary factor with the former as the secondary factor, due in part to the challenge among the community in solving [Solomon, 1987] benchmark test cases. [Larsen, 1999; Mester, 2002; Mester and Braysy, 2002] provides further references on the VRPTW.

3.3 Inventory Routing Problem with Time Window (IRPTW)

The Inventory Routing Problem with Time Window (IRPTW) follows as a natural extension from the VRPTW, with the additional constraint over multiple time-periods, which better reflect practical scenarios of a known future period planning. Despite the complexity, literature survey showed that IRPTW can be solved optimally if major restrictions are imposed. [Carter et al., 1996] proposed a Lagrangean heuristic to solve a single-supplier, single-warehouse instance of the problem, but it is sensitive to the values of several parameters where there are no good heuristics for setting them, and is unable to guarantee feasibility. [Chan et al., 1998] modeled a *single-item, constant demand* distribution system and presented worst case as well as probabilistic bounds. However, it is doubtful that any of the asymptotically optimal heuristic proposed will perform well for realistic problems with time-varying demand due to the unrealistic assumption on demand. [Campbell et al., 1998] proposed a computationally intensive integer programming approach to a similar problem. [Lau et al., 2000; Lau et al., 2002] proposed a divide and conquer approach of decomposing IRPTW into two sub-problems, then defined an interface to allow the two corresponding algorithms to collaborate in a *master-slave* fashion and provided a proof of convergence. This approach is unable to guarantee feasibility, when the output of the first module is infeasible for the second; and the quality of solution is necessarily low, since there is no provision for an iterative improvement heuristics. This approach is improved upon by [Lau et al.^{2}, 2003] which is derived as a product of this thesis, and is the implementation used to obtain the results presented in Chapter 7 for this problem.

IRPTW is defined as follows:

Given

S:	*set of suppliers*
R:	*set of retailers*
J:	*set of items*
T:	*consecutive days in the planning period {1,2,…,n}*
D_{ijt}:	*demand of retailer I for item j on day t*
Q_v:	*vehicle capacity*
Q_w:	*warehouse storage capacity*
Q_i:	*storage capacity of retailer i*
W_i:	*time window of retailer i*
C_j:	*inventory holding cost per unit item j per day at the warehouse*
C_{ij}:	*inventory holding cost per unit item j per day at retailer i*
B_{ij}:	*backlog cost per unit item j per day at retailer i*
T_{ik}:	*transportation cost incurred by visiting retailer i followed by k on the same route*

and

$G = (V,A,T)$ is a multi-period graph

where $V = (v_1,v_2,\ldots,v_i,\ldots,v_m)$ *is the vertex set, and*

$A = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ *is the edge set, and*

T : as defined above

Output the following:

[1] The distribution plan denoted by

x_{sjt}:	integral flow of amount of item *j* from supplier *s* to warehouse on day *t*, and
x_{ijt}:	integral flow amount of item *j* from the warehouse to retailer *i* on day *t*

[2] The set of daily transportation routes Φ, which carry the flow amounts in (1) from the warehouse to the retailers such that the sum of the following linear costs is minimized:

(a) inventory cost at the warehouse (C_j)

(b) inventory cost at the retailer (C_{ij})

(c) backlog cost (B_{ij})

(d) transportation cost from the warehouse to the retailers (T_{ik})

Work arising from this thesis improves upon the work of [Lau et al.[1], 2002], by decomposing IRPTW into VRPTW and the Dynamic Lot-sizing Problem (DLP), and in conjunction with the Tabu Search Framework (TSF++) [Wan, 2002; Lau et al.[1], 2003], presented a powerful algorithm for the IRPTW [Lau et al.[2], 2003].

3.4 Multi-Dimensional Knapsack Problem (MKP)

The Multi-Dimensional Knapsack Problem is an extension of another classic NP-hard problem, the 0-1 Knapsack Problem (KP). Every resource has a cost and value, so it becomes a decision to seek the maximum value for a given cost. The typical formulation in practice is the 0-1 Knapsack problem, where each item must be selected entirely or not at all (hence 0 *or* 1). This property makes the knapsack problem hard, as a simple greedy algorithm can find the optimal selection if objects can be subdivided arbitrarily. This NP-hard property has inspired use of problem as the underlying basis of cryptography systems. However, the simple knapsack system was broken using polynomial time algorithms by [Shamir, 1982], the Graham-Shamir system by [Adleman, 1983], and the iterated knapsack by [Brickell, 1984] who exploited the singular use of modular multiplication as the only method being used to hide the knapsack. These do not suggest that NP-hard problems are solvable, but that the knapsack problem holds some property allowing a backdoor approach to solving the problem. The polynomial time solution relies on the existence of a particular class of knapsack problems which can be solved trivially, and extrapolated to a harder problem. MKP, however, is not extensible from these classes of problems.

MKP is defined as follows:

Given

$$\begin{array}{lll} \textit{Maximize} & \sum_{j=1}^{n} p_j x_j & \\ \textit{Subject to} & \sum_{j=1}^{n} r_{ij} x_j \le c_i & i = 1,2,\dots,m \\ & x_j \in \{0,1\} & j = 1,2,\dots,n \\ \textit{Assuming} & p_j > 0 & \text{for all } j \in J \\ & r_{ij} \le c_i \le \sum_{j=1}^{n} r_{ij} & \text{for all } i \in I \text{ and } j \in J \end{array}$$

For the 0-1 Knapsack Problem, $m=1$. If m is greater than 1, than the problem becomes the m-dimensional knapsack problem. This result to this problem is not presented in this thesis, but it exists to provide the fundamental basis for the next problem, the MPMKP.

[Shih, 1979] presented a branch and bound algorithm for the MKP. Another branch and bound algorithm was developed by [Gavish and Pirkul, 1985] where various relaxations of the problem were used, their algorithm was compared with the exact algorithm of [Shih, 1979] and was found to be faster by at least one order of magnitude. Other previous exact algorithms, with only limited success reported, include the dynamic programming based methods.

[Loulou and Michaelides, 1979] presented a greedy-like method based on [Toyoda, 1975] primal heuristic. [Balas and Martin, 1980] used linear programming by relaxing the integrality constraints and heuristically setting the fractional solution to become integral while maintaining feasibility. [Pirkul, 1987] presented a heuristic algorithm which makes use of surrogate duality. Freville and Plateau (1994) presented an efficient preprocessing algorithm for the MKP. [Freville and Plateau, 1997] presented a heuristic for the special case, the bidimensional knapsack problem, their heuristic incorporated a number of components including problem reduction, a bound based surrogate relaxation and partial enumeration.

3.5 Multi-Period Multi-Dimensional Knapsack Problem (MPMKP)

The motivation for the MPMKP arises from a business practice known as the available-to-promise capacity, which basically is a supplier's ability to "promise" to match real-time customer requests with available items for all items over a certain planning horizon of multiple periods. This available-to-promise problem is modeled as a specifically becoming the MPMKP. Like IRPTW, MPMKP impose a late penalty cost (equivalent to the holding cost of IRPTW) if request cannot be satisfied in a certain period, or corresponding a decreasing profit with each period that the request is unfulfilled.

In a practical available-to-promise application, it is often more useful for vendors to be able to make quick decisions for the availability to promise to their customers than find optimal solutions over a long time. Global optimality usually takes too long, given the NP-hard nature of the problem. The objective of this problem then is to find heuristic approaches to get a near-optimal solution fast, to allow vendor to make quick decisions in offering availability-to-promise capability.

There is currently limited work on MPMKP, but the multiple time periods in this period inspires the study of this problem with respect to other similar multiple time-period problems like the IRPTW, which increases the number of constraints with respect to multiple time-periods.

MPMKP is defined as follow:

Given

T: *number of periods in the planning horizon*

N:	*total number of requests over the planning horizon*
M:	*total number of items*
t^j:	*prescribed period where request j is due to be fulfilled*
a_{ij}:	*order size for item i in request j*
$b_{i\,t}$:	*incoming quantity of item I at the beginning of period t*
p_{jt}:	*net profit if request j is fulfilled in period t*

for $1 \le j \le N$, $1 \le i \le M$ and $1 \le t \le T$

Assume

All requests from customer and quantity of restock are known at the beginning of each period.

Then

Maximize $$Z = \sum_{j=1}^{N}\sum_{t=1}^{T} p_{jt}x_{jt}$$

Subject to $$\sum_{j=1}^{N} a_{ij}x_{jt} + s_{it} = b_{it} + s_{i,t-1}$$

$$\sum_{j=1}^{N} r_{ij}x_j \le c_i$$

$$x_j \in \{0,1\} \qquad j = 1,2,\ldots,n$$

Assuming $$p_j > 0 \qquad \text{for all } j \in J$$

$$r_{ij} \le c_i \le \sum_{j=1}^{n} r_{ij} \qquad \text{for all } i \in I \text{ and } j \in J$$

In the context of this thesis, the profit function p_{jt} of any request *j* is a concave function with respect to *t* and partial delivery is not allowed (which is extended from the requirement in the 0-1 Knapsack Problem causing the problem to be hard). Furthermore, it can be seen that when *T=1*, the MPMKP is reduced to the

MKP. Similarly, if *T*=1 and *M*=1, the MPMKP is further reduced to the 0-1 Knapsack Problem.

CHAPTER 4

Ant Colony Optimization

This chapter introduces the Ant Colony Optimization meta-heuristics and its algorithmic structure. The observations from this chapter will demonstrate the potential to adapt into a generic software framework presented in the next chapter.

4.1 The Ants Metaphor

As already mentioned, ACO is based on the behavior of real ant colonies. Real ants are creatures lacking in the normal visual and audio sense utilized by other species. Despite that, ants are able to find the shortest path from a food source to their nest. Their primary sense is via their feelers, in particular to sense for the presence or absence of a chemical call pheromone which they can give out as they travel, much like the naturally secreted hormones of certain mammals. Ants follow, based on probability, pheromone that were previously deposited by other ants. Figure 4.1, extracted from [Dorigo et al., 1996], provides the classic demonstration of such a phenomenon.

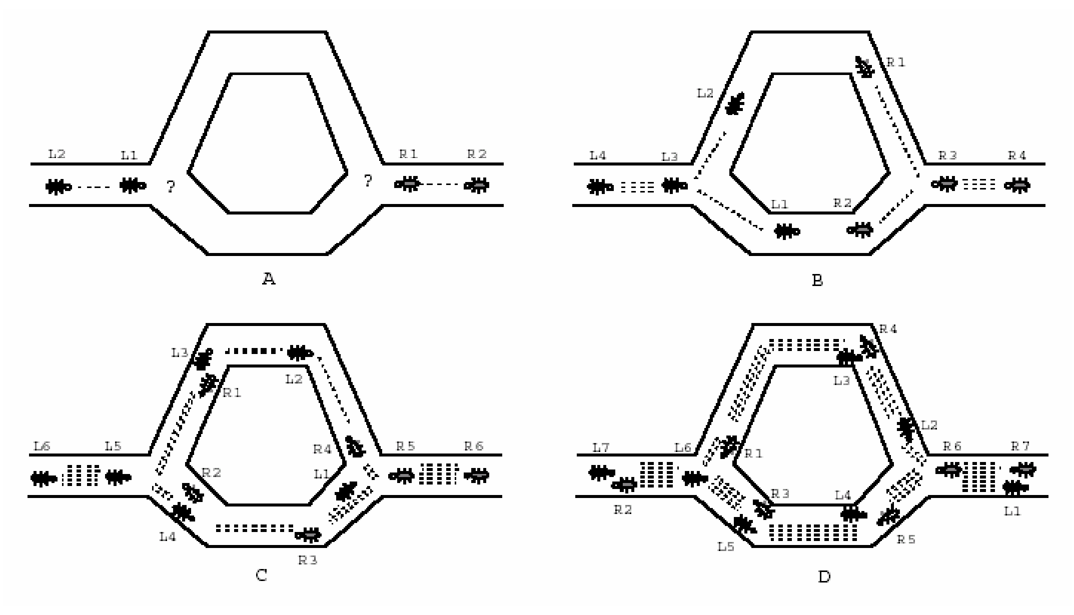


Figure 4.1: Real ants finding shortest path

In Figure 4.1(A), ants arrive at a decision point in which they have to decide whether to take the upper or lower path. Ants that originate from the left are labeled L*X*, and ants that came from the right labeled R*X*. At this stage, the choice is totally random, which statistically which imply that half of the ants will take the upper path, and the other half taking the lower path, as in Figure 4.1(B). However, since the lower path is shorter than the upper path, it can be expected that an ant will complete the path faster, hence implying that the pheromone trail on the path will be laid faster, as seen in Figure 4.1(C), where the dashed lines are roughly proportional to the amount of pheromone laid on the trail. As time passes, the difference in the pheromone level on the two paths will become significant enough to influence the random choice as ants come to the decision point. Ants will be attracted to the higher concentration of pheromone on the lower trail, as seen in Figure 4.1(D), which in turn causes a positive feedback effect. As more time passes, all ants will be using the shorter path. Real ants stimulate a fast and complete convergence to a solution. However, pheromone evaporates into the surrounding gradually. Hence, suppose that an even shorter path should suddenly appear. The evaporation of the trail will ensure that the ants on the existing path are not prevented from shifting to the new (better) path, though sometimes the concentration may be so high that this does not occur (akin to being trapped in local optimal). However, it is noted that that the reason why ants find the shortest path in Figure 4.1 is (1) the concurrent activity of many ants and (2) two-way traffic in the network (or one way traffic with return trip).

4.2 Algorithmic Structure

Following the observation of real ants in the previous section, this section presents the specific algorithmic structure relevant to the ACO meta-heuristics. Figure 4.2 reflects an improved ACO algorithmic framework modified from [Dorigo and Di Caro, 1999].

```
Procedure: ACO_meta_heuristics()
    While (termination-criterion-not-satisfied)
        Schedule_activities
            Ants_generation_and_activity()
            Global_Pheromone_Update
            Pheromone_Evaporation
            Daemon_actions()
        end Schedule_activities
    end While
end Procedure

Procedure: Ants_generation_and_activity()
    Schedule_creation_of_new_ant()
    While (available_resources)
        New_active_ant()
    end While
end Procedure

Procedure: New_active_ant()
    Initialize_ant();
    M = obtain_ant_shared_memory()
    While (current_state != target_state)
        A = read_local_ant_routing_table
        P = compute_transitional_probablities(A,M)
        Next_state = apply_ant_decision_policy(P)
```

```
            Move_to_next_state(next_state)
            Next_state = perform_local_search_improvement
            **If** (online_step-by-step_pheromone_update)
                Local_pheromone_Update
            **end If**
        **end While**
        **If** (online_delayed_pheromone_update)
            **Foreach** visited_arc do
                Local_Pheromone_Update
            **end Foreach**
        **end If**
    **end Procedure**
```

Figure 4.2: *ACO algorithmic framework*

Figure 4.2 depicted the ACO algorithmic framework. The main procedure *ACO_meta_heuristics* performs the iterative improvement steps bounded by the *termination_criteria*. Usually, this is in the form of a fix amount of real/cpu time or number of iterations. This main procedure spawns a single iteration of ants activity, specified by procedure *Ants_generation_and_activity()*, which simply organized the activities of individual ants in the iteration. Each ant activity is defined by procedure *New_active_ant()*. Each ant then decides on the path to take.

The algorithmic framework in Figure 4.2 is the definitive ACO framework in the literature. Despite this work, there is no software framework, due in part to the flexibility of the ACO algorithm in adapting to different situations. Each guideline in the algorithmic framework hence can be implemented in many different ways.

4.3 Standard Features of ACO

This section examines the standard features of ACO common to an arbitrarily generic ACO implementation. These features form the fundamental rules and propositions by which the ACO software framework will be built upon.

4.3.1 Transitional Probabilities

To decide on which path to take, each ant considers two main factors. First, the ant considers the natural judgment on whether to take the trail, specified by the local heuristics. The second factor to consider is the current concentration of pheromone on the specific trail in consideration. Each of these factors are assigned a weight, respectively α and β for the local heuristics and pheromone trail. In particular, the probability of moving from node r to node s is given generally by

$$p_k(r,s) = \begin{cases} \dfrac{[t\,(r,s)]^a \,.[h(r,s)]^b}{\sum_{u \in J_k(r)} [t\,(r,s)]^a \,.[h(r,s)]^b} & if \quad s \in J_k(r) \\ 0 & otherwise \end{cases} \qquad \ldots\ldots\ldots \text{(1)}$$

where $\tau(r,s)$ = pheromone for moving from node r to node s

$\eta(r,s)$ = local heuristics for moving from node r to node s

i.e., ants decide a path out of m path using two main factors – local heuristics and pheromone trail.

4.3.2 Local heuristics composition

There are instances of problems, especially those of increased complexity that a single local heuristics does not suffice. For instance, there had been implementations of VRPTW with multiple combined local heuristics [Bullnheimer et al., 1997]. In such instances, $\tau(r,s)$ from equation (1) can be formulated as

$$t\,(r,s) = \sum_{j=1}^{n} [t_{\,j}\,(r,s)]^{a_j} \qquad \ldots\ldots\ldots \text{(2)}$$

where $\alpha_j \geq 0$ and symbolize the weights of the local heuristics

i.e., local heuristics can be a composite of many separate heuristics.

4.3.3 Default Pheromone Value

The pheromone trail t should be initialized to be fixed value across of the trails prior to being used, and the value it is initialized to, t_0, is usually given by a generic "baseline" solution to the problem. This solution can be evaluated using any construction algorithm like Greedy Algorithm, or even ACO itself (using a generic pheromone trail initialized to any arbitrary value). t_0 is a function of this initial solution.

i.e., there should be a default baseline pheromone value, t_0.

4.3.4 Exploration and Exploitation

The probabilities derived from equation (1) and (2) can be utilized in any ways that meets the need of the problem in question. In particular, works in the literature used mainly two main ways – *exploration* and *exploitation*. Exploration is the process whereby the ant decides which path to take based on the concentration probabilities calculated. Hence, there is a higher chance of taking a path with higher calculated probabilities. Exploitation is the decision of taking the path with the highest calculated probability. This decision is performed based on a probability factor q_0, the *exploitation factor*, or the probability that the move under decision will be exploited.

Suppose there are k possible nodes to be chosen for the next node s. Let p_i ($1 \leq i \leq k$) denote the transition probability to move to node i. Let p_{max} denote the maximum probability among all the possible moves. Then, the following procedure is used to determine the next node.

```
generate a random number $r_{E/E}$ between 0 and 1
if ( $r_{E/E} \geq q_0$ )   // i.e. explore
    generate another random number r ($0 \leq r \leq 1$)
    choose $s$ to be the node $i$ s.t. $\Sigma^{i}_{j=1} p_j \leq r \leq \Sigma^{i+1}_{j=1} p_j$……… (3)
else  // i.e. exploit
    choose $s$ to be the node with the highest probability $p_{max}$
```

At this point, the ant would have decided on the next move to take from the current state, and perform the updates necessary to effect this move, *i.e., ants can use the pheromone trail in different ways.*

4.3.5 Local Pheromone Decay/Deposit

After each move is completed, the ant may choose to perform a local pheromone decay or deposit. If no such action is performed, each of the ants in the iteration will be non-collaborative and use only the pheromone trail at the beginning of the iteration. While there are implementations without local pheromone updates with good results, it was generally found that local pheromone update improves solution quality. The logic is that unlike real-ants, the solver of an optimization problem need to traverse the best path once to record it, and implement other ways to enforce this knowledge (global pheromone update). Meanwhile, it is necessary to search as much of the solution space as possible, and in most cases, it is better to lower the pheromone concentration from a taken trail, so that other ants may try the path less trodden, which allows search around the neighborhood of a good solution as well as prevent solution cycling. There are many formulas (if implemented) for local pheromone update, but generally,

$$t(r,s) \leftarrow (1 - r_l).t(r,s) + r_l.t_0 \qquad \text{……… (4)}$$

where t_0 represents the default pheromone level

r_l represents the local decay factor

Local pheromone update can be performed in two ways. The first, ***step-by-step*** update, is performed as each ant takes a move. The nature of this process makes it more suited for a parallel implementation. The second, online-delayed pheromone update, is performed as each ant completes a solution build, and is more suited for a serial implementation.

i.e., ants may optionally decay or deposit on the pheromone trail in the local context.

4.3.6 Global Pheromone Decay/Deposit

While the local pheromone update may be optional, the global pheromone update that occurred at the end of an iteration is compulsory. The justification for such an action is by counter-intuition. Suppose there is no pheromone update. Then, each ant will repeatedly find the same probabilities on all the moves. The only variable then is the random choice. While this progresses the solution, it does so very gradually. Furthermore, there tend to be an excessive amount of solution cycling due to the constant nature of the probabilities. This completes the intuition that the pheromone trail should be updated.

Global pheromone update can be done in several ways. Some implementations proposed using the trail from all the ants in the iteration, others advocate using only the best route in the iteration, and most suggest using the best route found so far. Generally,

$$t(r,s) \leftarrow (1 - r_g).t(r,s) + r_g.\Delta t(r,s) \qquad \text{……….} (5)$$

where r_g represents the global decay factor

i.e., ants must deposit on the pheromone trail in the global context.

4.3.7 Pheromone Evaporation

In synch with global pheromone update is the optional pheromone evaporation. One idea is to use additional reinforcement for unused movements, with equation (6), while other approaches perform a simple evaporation on all trails with equation (7), for all *i* and *j*:

$$t(i,j) \leftarrow t(i,j) + r_e.t_0 \qquad \text{……….} (6)$$

$$t(i,j) \leftarrow (1 - r_e).t(i,j) \qquad \text{……….} (7)$$

where r_e represents the evaporation factor

i.e., ants may optionally perform decay evaporation for additional reinforcement on unused trails.

4.4 Observations on ACO implementations

The previous section provides the proposition assumed to be valid in the complete operations of ACO. The rest of this section presents observations from work in the literature as well as a result of the development of this thesis that assists in the development of the software framework.

4.4.1 Pheromone Trails

From the formulation of the ACO algorithm above, there are a few parameters relevant to any arbitrary ACO implementation. Primarily, the pheromone trail t is a compulsory factor, as well as an optional initial value for the pheromone trail t_0. However, the optimal construct of t is specific to the problem or the definition of the implementer. For instance, the typical implementation of ACO has the pheromone trail as the edges between all nodes in the problem, such as in TSP, VRPTW, etc. The pheromone trail is a very importantly component of the ACO algorithm (as much as the local heuristics function) that directly influenced the effectiveness of the algorithm in solving a problem. A study on the pheromone trail and its relation to performance can be found in [Dorigo and Gambardella, 1997].

4.4.2 Fundamental ACO parameters

Other parameters included the weights value of a and b, as well as the inclusive subsets of these weights in the case of section 4.3.2. [Dorigo, 1991] found from experimental results that good values of a and b (for TSP at least) are 1 and 5 respectively. A greater weight is usually placed on the local heuristics (affected by b) to prevent fast convergence to local optimal. Another argument for a larger b is also given by section 4.4.4 below.

Another key parameters in the ACO algorithm is the decay factors r. These factors are generally a floating point value between 0 and 1, to signify the percentage of decay/evaporation (0 means no decay, 1 means complete decay). Decay factors can be subdivided into three separate parameters (local decay, global decay, and evaporation), although most classic ACO uses the same value for them.

Exploration or exploitation is another important factor in the ACO algorithm. A complete exploitation reduce the algorithm simply to the power of the local heuristics, in most cases just a greedy approach. Exploration allows an opportunity to search around the best found solution, a technique that works often in non-linear problems, such as the classes of problem (combinatorial optimization problems) dealt with here. The decision of exploration or exploitation is defined by the factor q_0, the exploitation factor, which is a floating point value between 0 and 1. When q_0 is 0, the ants explore all the time; when q_0 is 1, exploitation occurs all the time.

4.4.3 Number of ants

There are many arguments on the optimal number of ants, *num_ants*, in the literature. In particular, the two most commonly argued value for this parameter is a constant value (e.g., 10) or *n* (problem size) [Bullnheimer et al., 1997; Dorigo et al., 1996]. While this parameter will be implementation specified in the framework, it is the author's opinion that a constant number of ants is a better figure for most problems from experimental observations. Furthermore, choosing *n* will increase the computational complexity of the problem by another factor of *n*. Based on *x* iterations and n^2 for the probability calculation as well as move[1] choosing a constant number of ants give $O(xn^2)$, whereas *n* ants gives $O(xn^3)$, hardly an efficient approach. However, the value of the pheromone decay and collaboration might compensate for the computational complexity, and allow the search for a better solution to be found in significantly less iterations. Hence, both arguments are valid, and the decision on the value should be up to the implementer.

[1] It is possible to reduce this time complexity to *log(n)* with optimized implementation

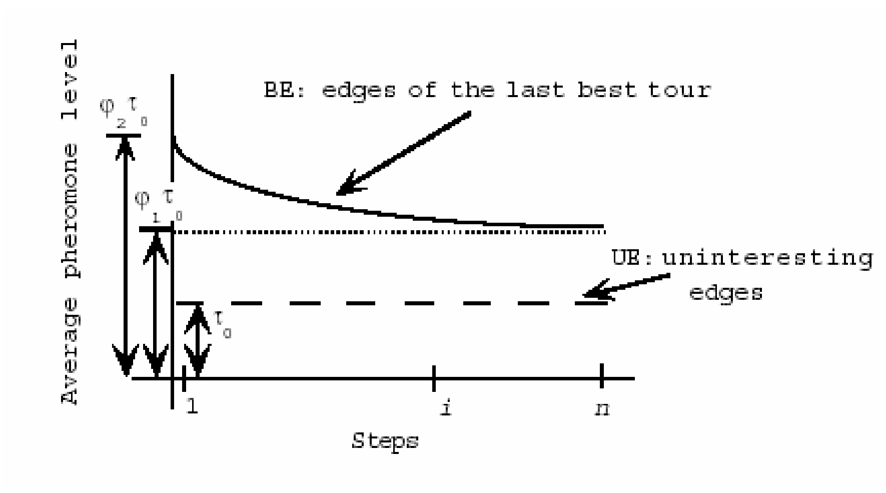


Figure 4.3: <u>*Change in pheromone trail*</u>

Consider Figure 4.3, extracted from [Dorigo and Gambardella, 1997] to provide for this observation. *BE* represents the pheromone trail average with increasing iterations when exploiting the Best Edges found so far, while *UE* defines the Uninteresting Edges, which are edges that have not participated in the best found solutions. UE also defines the cut off point given by the initially found and lower bound value t_0. $j_1 t_0$ represents the average pheromone level at the end of each iteration (before global pheromone update is applied). Hence, assuming that the generic pheromone update formulas from equation (4) is applied in the implementation,

$$t_i(r,s) \leftarrow (1-r).t_{i-1}(r,s) + r.t_0 \qquad \ldots\ldots\ldots (8)$$

$$è \quad t_i(r,s) \leftarrow t_1(r,s).(1-r)^i - t_0.(1-r)^i + t_0 \qquad \ldots\ldots\ldots (9)$$

where $t_1(r,s) = j_2 t_0$

$t_i(r,s) = j_1 t_0$

i = approximate number of ants that update edges (hence affecting a graph change as seen in Figure 4.3) è $j_1 \leftarrow j_2.(1-r)^i - (1-r)^i + 1$ ………. (10)

Since the best edges are chosen with probability ≥ q_0, then

$i \approx m.q_0$ ………. (11)

where m = optimal number of ants

è $m = \dfrac{\log(j_1 - 1) - \log(j_2 - 1)}{q_0.\log(1-r)}$ ………. (12)

Equation (12) showed that the optimal number of ants is a function of j_1 and j_2. The difficulty, however, is that j_1 and j_2 are reliant on the structure of the pheromone trail as well being problem specific.

4.4.4 Importance of local heuristics function

Experimental results from many implementations of ACO showed that the local heuristics is a key deciding factor in the effectiveness of the algorithm. This point alone makes ACO an excellent candidate for a software framework accommodating other heuristics. This is also the reason why ACO performs well when collaborating with other local search heuristics. ACO provides a communicate medium (pheromone) for a synergistic effect to take place, while the power of the entire algorithm is dependent on the local heuristics function.

Some simple and effective algorithm for the local heuristics function included Greedy Algorithm, which is the main local heuristics algorithm used in many ACO implementations. Other effective techniques included collaborating with advanced techniques like Tabu Search, Simulated Annealing, etc. It is also from this observation that b is usually given a higher weight, as the local heuristics performs the main guidance for the progress of the algorithm. The pheromone trail (defined by a), will accumulate with increasing ants and iteration to influence the local heuristics.

4.4.5 Parallel vs. Serial implementation

It is obvious that ACO, being an agent-based (each ant being an agent) algorithm, is excellent when using a parallel implementation. However, most ACO performances were compared using a serial implementation, enhancing its importance to the community. A generic ACO software framework will provide the guideline for implementation of either implementation.

4. 5 Hypothesis of ACO

The previous section provides the fundamental observations that are obvious and obtainable from results and literature survey. The remainder of this section provides hypothesis on the performance of ACO for different and hard scenarios, as well as suggest solutions appropriately.

4.5.1 Large problem instance

From the algorithmic structure of ACO seen in Figure 4.2 earlier, as well as the O notation discussed in section 4.4.3, it is seen that ACO is hardly an efficient algorithm, at $O(xn^3)$ at best, with a typical $O(xn^2)$ for a constant number of ants. This suggested that with increasing n, ACO poses a problem. The workaround to this is to attempt to ensure that ACO is able to find a good solution in a minimal amount of iterations. Also, the implementer should note to reduce computational requirements as much as possible. The key to this workaround lies in the pheromone trail structure (for the inner-most loop performance), and the local heuristics (to find a good solution fast).

4.5.2 ACO solution cycles with increasing iteration

Another problem with ACO is due to the exploitation performed by ants. The probability of this action is determined directly by the value of the exploitation factor q_0. A low q_0 influences the algorithm to explore more often, but if the value is too low, the search tends to be too erratic and unguided, despite the pheromone trail. Hence, the suggested value for q_0 is usually on the higher end of the probability scale (0.7-0.9). A high exploitation q_0 however, has two main pitfalls. First, it could cause the solution to converge too fast to escape local optimal, and secondly, the algorithm might cycle through the same solutions, especially with increasing iterations, due to the positive reinforcement of the pheromone trail. This observation suggests that it is advantageous to gradually modify the exploitation factor q_0 if the solution is not improving after a fixed number of iteration to counter the positive reinforcement. With a good value of q_0, it is also true that when this observation occurs, the search has reached local optimal, since there is no neighboring moving that can improve the solution. Hence, it is also a feasible approach to provide a temporary radical shift to q_0. No work in the literature has applied such either of these two approaches nor stated such an observation.

4.5.3 ACO works better with other local search, and vice versa

Section 4.4.4 stated that ACO worked better with local search. This is especially true for extended problems with multiple constraints and dimensions. The key reason for this is due to the general performance of the ACO algorithm, stated by sections 4.3.3 and 4.5.1. In such instances, a good approach is to apply another good heuristics into either an inner loop of the algorithm, or apply ACO as a powerful construction algorithm, by limiting the number of iterations to a practical value. As such, to accommodate these sufficiently complex problems, the ACO software framework should allow easy collaboration with other techniques in any portion of the algorithm. In addition, many other local search heuristics, e.g., Tabu Search, which operates mainly as an optimization phase algorithm, is better able to achieve good results with a correspondingly good initial solution provided by the construction phase algorithm. ACO potentially provides such an algorithm.

4.5.4 ACO is an effective construction phase algorithm

Since ACO operates by allowing each ant to construct a solution, this hypothesis is trivial. This nature of ACO causes it to escape from the ease by which many other heuristics get trapped in local optimal, but it also increases the computational intensity of the algorithm. Regardless, it is this nature that enables ACO to be an excellent construction phase algorithm, and especially if placed in collaboration with other local search heuristics.

CHAPTER 5

Software Framework

This chapter capitalized on the observations and hypothesis from the previous chapter to present the ACO framework. In particular, the first section presents the Meta-Heuristics Development Framework (MDF), an over-seer meta-heuristics framework that integrates any heuristics and allows collaboration between the algorithms. A brief introduction to the other algorithms existing in MDF is given, with particular emphasis on the ACO framework component (ACF), the gist of the thesis.

5.1 Meta-Heuristics Development Framework

Solving planning and scheduling problems using meta-heuristics has become popular, mainly due to the ineptitude of exact methods in producing quality solutions for large problem sizes. In particular, recent researches had revealed many new innovative techniques, evolution of existing algorithms and hybridization of one or more such meta-heuristics. The maturity and popularity of these techniques brings about rapid growth of differing approaches which shared many similarities. This diffusion, while healthy for seeding new ideas into the community, is met with such diversity in implementation that renders experimental benchmarking difficult. This elevates a need to develop a generic framework that provides some basis for comparison and collaboration of existing and newly developed techniques. For the framework to be appealing to the community, it should offer code reusability, thereby reducing development time on one hand, as well as flexibility for researchers to effectively inject algorithmic strategies that are peculiar to their proposed ideas. A collaboratively result of this thesis yields a high level framework, the Meta-heuristics Development Framework (MDF), designed to meet those goals. MDF presents a model to facilitate multi-algorithm inter-operability. This nature of MDF does not require that each algorithmic engine be as generic as possible. While it is advantageous that each is a specific lower level algorithm framework, if implementations of similar algorithm must be varied, there can be framework for variations of the same algorithm in the framework.

The sections that follow present the overview of MDF, as well as the individual framework for ACO, TS, and SA, which are by no means the limitations of which algorithm can be integrated, but rather a sample of currently existing components.

5.2 Overview of Framework

This section presents the general concepts of the MDF framework, depicted by Figure 5.1. MDF uses abstraction and inheritance as the primary mechanism to build adaptable components or interfaces. The general behavior of local search is factored out and grouped into generic interfaces, thus rendering the framework to be robust yet flexible. These common interfaces include *Solution*, *Objective Function*, *Move, Constraint* and *Neighborhood Generator*. The *Solution* is used as a representation for the problem output. The *Objective Function* evaluates the objective value of solution. The *Move* translates a Solution object into a new solution while the *Constraint* checks on the degree of violation. Finally, the *Neighborhood Generator* generates a list of feasible neighbors through the Move and Constraint interfaces. Each of these generic interfaces makes no assumption on any specific meta-heuristic or the problem that it is acting. For example, the solution interface did not restrict developers to any formulation or data structures. Rather, the framework manipulates the solution indirectly through compulsive virtual methods and inheritance behaviors. MDF also includes an *Engine* interface that outlines the rudimentary controls performed by the any meta-heuristic applications. Some of these controls include *Iterations-To-Go*, *Start-Solving*, *Stop-Solving,* and *Stopping criteria*.

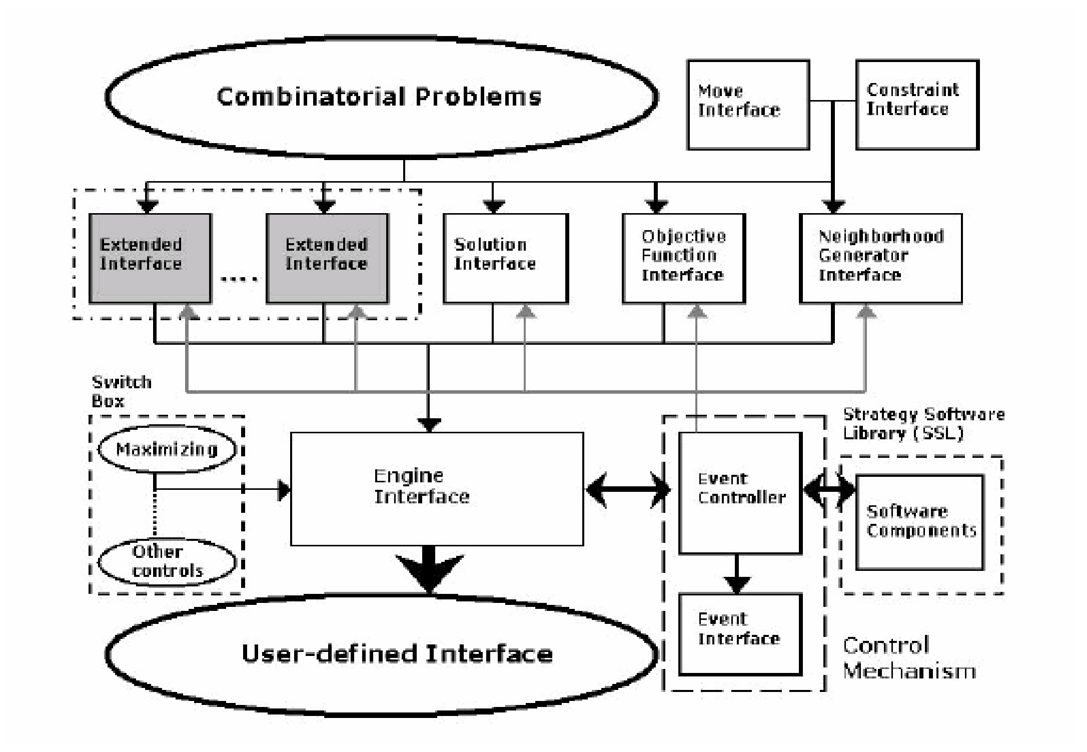


Figure 5.1: Architecture of MDF

MDF also includes a *Switch Box*, a *Control Mechanism* and a *Strategy Software Library (SSL)*. The *Switch Box* consists of a set of switches used to operate the framework engine. The maximizing control is an essential control for all derived frameworks and is used to determine if the search is performing maximization or minimization. The strength of MDF lies in the inclusion of the *Event Controller*, which allows adaptive control over the search sequence, thus giving developers the ability to guide the framework engine. The Event Controller uses an *Event* interface that is used to define a situation that may be experienced in the search. Some examples are *new best-found solutions*, *series of non-improving solutions* and *no feasible solution found.* As the search procedures of each meta-heuristic approach are different, the Event Controller varies across each derived framework.

The SSL is used to facilitate the development of algorithmic strategies. SSL has a set of generic components that offers a quick and easy means to deploy their strategies. Some of these components include a fundamentally useful functions such as a *Percentage Random Generator* that randomly generate a floating float value between 0 and 1, a *Permutation Generator* that returns a permutated set over a given set of objects and a *Elite List* data structure that is used with the Event Controller to collect solutions whose objective value are above a user-specified threshold. These components also provide a means for the developers to collect useful search information such as information regarding *Recency and Frequency* of partial solutions [Glover and Laguna, 1997].

5.3 Ant Colony Framework (ACF)

This section presents the Ant Colony Framework (MDF-ACF) based on the generic ant colony framework (Figure 4.2) with potential extensions built into consideration via the event controller interface, allowing for most known implementations of ACO, as shown in Figure 5.2. MDF-ACF is twofold; it aims to serve both as an ACO software framework by itself, as well as being a component engine of MDF.

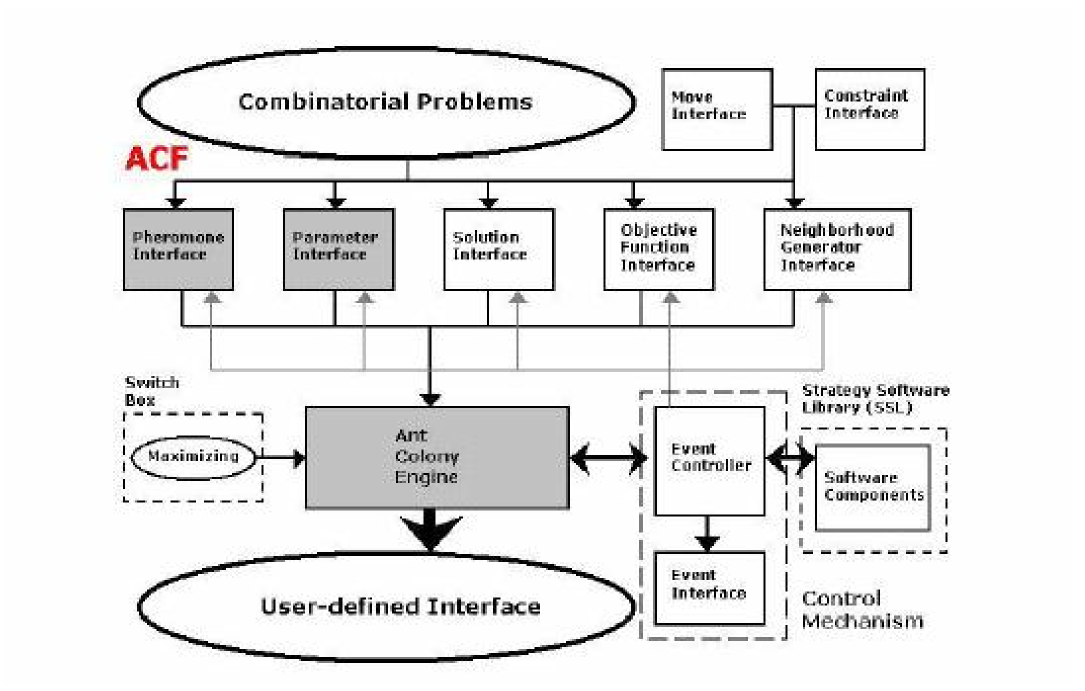


Figure 5.2: *Ant Colony Framework architecture (in the context of MDF)*

MDF-ACF Parameters Interface

ACO works mainly using parameters tuning, where one of the key factors affecting the effectiveness of an implementation is the values of the parameters. As such, this interface is concerned mainly with these necessary parameters inherent in ACO. In particular, there is the following:

a) size of problem,

b) number of ants,

c) decay factors (local decay, global decay, evaporation),

d) exploitation factor (q_0).

As with any other interfaces in the framework, the implementer can extend this interface to include additional parameters such as power of elitist ants, value of α (weight of pheromone trails), value of β (weight of local heuristics), etc. The weights are not included in the parameter base interface since the implementer is free to design his own vector of power values, such as when there are multiple local heuristics being used in the implementation (as specified in section 4.3.2).

MDF-ACF Pheromone Interface

Another key component of ACO is the pheromone trails. This interface enforces this requirement by requiring the implementer to state how the pheromone trail is structured.

MDF-ACF EventController

The EventController class is a key class in allowing flexible implementation. Besides generic events which are placed throughout the engine, this interface also requires the user's implementation for how the engine should perform certain actions, in particular the following related to the ACO algorithm:

a) local pheromone update,

b) global pheromone update,

c) pheromone evaporation,

d) when to stop an iteration or active ant,

e) how to calculate the probability for a node,

f) where is the starting location for a new active ant, and

g) what to do when a new current or best solution is found.

These presents the possibilities to code many proposed ACO implementations, as well as allow hybrid techniques to be inserted into appropriate places in the ACO execution, such as running another local search algorithm after finding a best solution to optimize this solution locally.

5.4 Tabu Search Framework (TSF)

MDF-TSF is improved from TSF++ [Wan, 2002; Lau et al [1], 2003] and inherits the fundamental architecture of MDF with the addition of TabuList and AspirationCriteria interfaces, an MDF-TSF EventController and a tabu search engine. The tabu search engine uses the set of interfaces to epitomize the routine tabu search procedures. Furthermore, it also exploits the EventController to adaptively guide the search in accordance to events experienced during the search. As such, the MDF-TS have more potential than a simple tabu search implementation and yet remain as a black box that can allow great flexibility to both developers and researchers alike.

MDF-TSF TabuList Interface

The adaptive memory is the key component of tabu search where the quality of solutions often relies heavily on the effectiveness of the tabu list. The *TabuList* gives developers the flexibility on the choice of objects to be tabu. Some commonly tabu objects are the solutions themselves, the translating moves, recurring partial solutions and objective values.

MDF-TSF AspirationCriteria Interface

The *AspirationCriteria* is an optional interface in the framework. The function of aspiration criteria is to override the tabu status of a move if it meets certain criteria. This is used mainly to avoid missing good solutions or moves that are otherwise tabu-ed.

MDF-TSF EventController

In a tabu search, it is often desirable for the tabu search engine to respond to search events by readjusting the elements in the interfaces. For example, a reactive tabu list would need to readjust its tenure (duration in which a move remains tabu-active) in respond to the success or failure in obtaining a better solution, during the search. Hence there must be means of controlling the tabu search to make such dynamic readjustments. The EventController acts as a centralize control unit that provides interaction between the tabu search engine and the interfaces. When the tabu search engine detected any search events, it would reflect them to the EventController, which would then respond to these events in accordance to strategies set by the users. Usually, these strategies will affect one or more of the interfaces that will in turn re-adjust the search approach adopted by the engine. Going back to our example of implementing a reactive tabu list, the "triggering-event" detected by the tabu search engine can be the number of non-improving moves made since the last best-found solution. Once the engine triggered this event, the control will be passed to the EventController. The strategy will be to readjust the tabu tenure in *TabuList*, using parameters such as number of iterations completed, and number of remaining iterations and even from the history from such past readjustment.

5.5 Simulated Annealing Framework (SAF)

MDF-SAF is the third illustration on the generic aspect of the MDF. This derived framework has only one extended interface called the *CoolingFunction*. In a similar architecture to the MDF-TSF, MDF-SAF has its own EventController and search engine. The engine again performs the standard routine of the meta-heuristic and uses the EventController to re-adjust its search. One possible use of the EventController is to dynamically switch between different cooling functions in accordance to the quality of solutions. Another popular use for the MDF-SAF is to hybridize with the MDF-TSF as the diversifier for the tabu search when the search reaches a local optimal. This hybridization is again easily achieved through the use of EventController.

MDF-SAF CoolingFunction Interface

The CoolingFunction determines the probability of a neighbor being accepted. SA typically only required iteratively modification for values of different parameters to obtain different cooling schedules to produce solutions of different qualities. MDF caters for the common scenarios as well as having the advantage of the CoolingFunction interface to provide further flexibility for the developers to have any number of cooling functions to be implemented, and allow the uses of EventController to switch between these functions, if such an implementation is desired.

CHAPTER 6

ACO Schemes

This chapter presents several ACO schemes derived from the observations and hypothesis from chapter 4, with respect to the ACO software framework. These schemes are recommended for use by implementers, specializing in different classes of problems. These schemes are individual modules that may be built alone or with each other into any implementation from the framework.

6.1 Basic ACO scheme

The natural choice is to present first the basic ACO scheme generic to most of the implementations in the literature. This scheme derives from the classic algorithmic framework in Figure 4.2, as well as sections 4.3 and 4.4. In short, it is the fundamental ACO as proposed by [Dorigo and Di Caro, 1999], adapted into a software framework, taking into consideration most of the ACO implementations and proposals in the literature. As such, this basic ACO scheme may be extended to easily reproduce current works.

6.2 Variable exploitation factor schemes

Sections 4.5.1 and 4.5.2 presented a potential pitfall with the ACO algorithm. Section 4.5.1 stated the performance of the algorithm with increasing iteration, and section 4.5.2 expanded the problem with a decrease in effectiveness at the same time. As such, ACO is not intrinsically designed for an excessive amount of iterations. Good results must be obtained within a reasonable threshold, beyond which the ACO under-performs. To address this issue, the following two schemes exploiting section 4.4.1 are proposed:

Scheme 1: **If there is no improving move within t_h number of iterations, adjust (decrease) the exploitation factor q_0, such that ACO explores more often.**

Or

Scheme 2: **If there is no improving move within t_h number of iterations, adjust (decrease) the exploitation factor q_0 by a significant amount *temporarily*, such that the excessively exploration will disrupt the overly concentrated pheromone trail.**

Both schemes relied on the modification to q_0. As best known to the author, no works in the literature had attempted such an approach. Observations that arises from the comparisons of these two schemes against the basic ACO scheme suggested variable results (not reflected in this thesis), depending upon the value of t_h and q_0 modification. The proper value to use is problem and parameters specified. In most cases, Scheme 1 performs better than the Scheme 2, which relied more on the operation of the pheromone update functions. Both schemes are also dependent upon the value of the best found solution at the moment q_0 is changed, which is directly related to the effectiveness of the local heuristics. Since most of the experiments were tried on known techniques, the local heuristics tend to be already good. However, that the schemes improve results in many cases, especially when the local heuristics function is weak, reflected their potential. Hence, these two schemes are worth considering, especially for countering known weakness of the local heuristics, which is hard to specify for certain problems; or to counter the effect of bad parameter setting. Furthermore, these schemes are extremely valuable when the algorithm needed to be executed for long number of iterations for any reason, such as when solving increasing complex problems.

6.3 Schemes that handle Time explicitly

Optimization that involves multiple time-period constraints is more complex. For instance, the IRPTW and MPMKP discussed in this thesis extend standard NP-hard problems, like the VRPTW and MKP respectively, with time-period constraints, which increase the practicality of the problem to industrial applications. The classic pheromone trail specified using node to node value are not appropriate for such scenarios, and a new scheme for the pheromone trail and ACO algorithm conducive to this environment that improve the performance of ACO for such classes of problem is proposed.

By virtue of the ACO basic scheme, there are certain properties of the underlying algorithm that can be logically extended to provide a scheme specializing in handling the notion of time, as can be seen from the experimental results presented in the next chapter. The first and most instinctively of these schemes that exploit the basic operation of ACO is:

Scheme 3: In addition to the basic pheromone trail presenting a pheromone value from one node to another, add another dimension to the pheromone trail structure, hence presenting a 3-D pheromone corresponding to the geometric 3-D of 2-D space and 1-D time.

Figure 6.1(b) below reflects the new pseudo-code structure that results. However, Scheme 3 presents an obvious problem. As already analyzed, ACO is already computational intensive. To add another dimension of time, *T*, to the problem increases the complexity of the algorithm in the inner-loop (where the pheromone trail are used to calculate individual movement probability) to O(xn^2T). As such, the following scheme is more appropriate for multiple time-period problems.

Scheme 4: Modify the pheromone trail from a node-to-node trail to a node-to-period trail. This pheromone trail will specify the attractiveness of placing a node in a time-period.

For instance, instead of a typical pheromone structure like Figure 6.1(a), which is commonly applied in most implementation of ACO, Figure 6.1(c) shows the structure that would be implemented instead using Scheme 4. Note the change from a node[2]-to-node structure to a node-to-period structure, which reflects the preference of placing a node to a time-period, as opposed to the typical preference of placing a node after another node.

```
Class ACOParameters : ACOParameters
{
    double pheromoneTrail[MAX_NODE] [MAX_NODE];
}
```

(a) Typical Scheme

```
Class ACOParameters : ACOParameters
{
    double pheromoneTrail[MAX_PERIOD][MAX_NODE] [MAX_NODE];
}
```

(b) Scheme 3

```
Class ACOParameters : ACOParameters
{
    double pheromoneTrail[MAX_PERIOD] [MAX_NODE];
}
```

(c) Scheme 4

Figure 6.1: *Pseudo-code structure of the pheromone trail*

[2] A node can represent a vertex, a request, or any point of reference by which the solution is constructed

The structure resulting from Scheme 4 (Figure 6.1(c)) arises from the work of this thesis, in particular related to IRPTW and MPMKP. There are several reasons that Scheme 4 as better than Scheme 3. First, the computational complexity becomes O(*xnT*). It can be seen that the complexity may actually be reduced, if the number of time-periods, *T*, is asymptotically smaller than the number of nodes *n*, which is usually the case. Furthermore, this structure is logically better than the classic version in dealing explicitly with time, since it directly informs the algorithm where the best period place node is. This easily allows the problem to be broken into smaller sub-problems (such as IRPTW into VRPTW), solve the single time-period problem, and then allow the basic ACO scheme to take care of the less complex problem.

6.4 Decay Schemes

This aspect of ACO, noted previously in sections 4.3.3, 4.3.5, 4.3.6, and 4.3.7, is where there is the most research activity. There are many proposals presenting different decay functions combinations for the local decay, global decay, and evaporation, often combining the functions together, such as seen in [Gambardella et al.[1], 1999; Fidanova[1], 2002]. The pheromone trail can be updated in myriad of ways to achieve different results. This section consolidates these works, extracting their decay approaches, and presents them in explicit schemes. Some good schemes included:

Scheme 5: **Pheromone trails are updated using the best found solution multiplied by a factor. This symbolizes trails taken by *Elitist Ants* which found the best found solution.**

Scheme 6: **Do not perform one or more of the three decay functions.**

Scheme 7: **Combine one or more of the decay functions together.**

Scheme 8: **Use a constant value or a fast heuristics for choosing the default decay value τ_0.**

Scheme 5 presents the approach taken by some successful ACO technique for the global pheromone update to use the best route found for the update, but multiplied by a factor, symbolized by elitist ants which are able to find better solutions and lay more pheromones than “normal” ants, seen in [Bullnheimer et al., 1997] using a multiplier of 5.

Scheme 6 is a scheme used in almost all ACO implementations, especially with regards to the evaporation function, although many early ACO proposals ignore the local pheromone update as well. The lack of an evaporation function is justified if the global and local decay functions update the trail appropriately, though it may cause an increasing pheromone concentration (which is disadvantageous for the operation of ACO in general) if the local decay functions are absent, does not decrease the pheromone traversed by the ant, or there is a lack of ants in each iteration. However, ACO usually does not execute long enough for the effect to be detrimental, and experimental observations arising from this thesis suggested the lack of an evaporation function, and is a method used in all the implementations used to obtain results for the next chapter. The lack of a local pheromone update function, conversely, might cause the same problem, depending on the other two decays function. However, it was found that the lack of this function affects the effectiveness negatively most of the time, and also, it is advisable to reduce the number of ants in each iteration if the local pheromone update function is missing, as the only ways the ants will work collaboratively is through the global update and/or evaporation function outside of every iteration, and it is proven experimentally in [Dorigo and Gambardella, 1997] that the ants perform better collaboratively.

In the same line of thought, Scheme 7 is closely related to Scheme 6. The usual case of Scheme 7 is when the global pheromone update function is integrated with the evaporation function. It should be noted that usually the evaporation function applies to all the pheromone trails, while the global update function operate only on the solution trail. As well, some implementation may delay the local update function until the end of an iteration, effectively operating like a complex global update function.

Scheme 8 provides some methods of finding t_0. This value is can either be a fix constant (where 0 is a commonly used value), or found using some fast construction algorithm such as Greedy Algorithm. Alternatively, t_0 can be extracted from the objective function using an initial pass of the ACO algorithm itself, since ACO is itself a capable construction algorithm. The choice of this value provides may affect the decay functions, which in turn affect the pheromone trail, in turn affecting the effectiveness and efficiency of the algorithm with increasing iteration.

CHAPTER 7

Solution Approach [3]

Having examined the MDF framework in chapter 5, and the ACO framework as well as some potential schemes presented in Chapter 6, this chapter presents our proposed solution approach to solve the problems presented in chapter 3, with the results presented in the next chapter. The first section presents hybridization with a sample implementation of MDF, using ACF and TSF as the key component. This hybridization, HASTS, is the solution approach taken for the VRPTW, which bases its implementation from a generic ACO implementation for TSP; as well as the IRPTW, which reuses the implementation from VRPTW with certain extensions. The last section further presents the solution approach for MPMKP using a time-period ACO scheme (Scheme 4) with ACF to solve the MPMKP.

The choice of VRPTW and IRPTW justifies the power of reuse in the framework, showing that a good solver for VRPTW can be reused to provide another good solver for the extended problem of IRPTW. Meanwhile, IRPTW and MPMKP are presented to demonstrate the effectiveness of Scheme 4, showing the proposed strain of ACO that is effective at solving multiple time-period problems.

7.1 Sample Implementation – HASTS

This section presents a sample implementation of MDF, using the ACF and TSF framework, exploiting the hypothesis noted in section 4.5.3. This

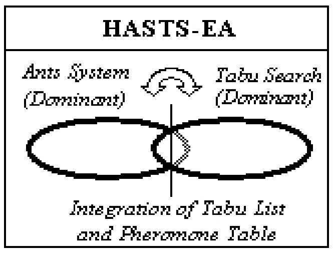
[3] Part of this chapter appears in [Lau et al.[2], 2003]

implementation described by [Lau et al.[2], 2003], called Hybrid Ant System and Tabu search (HASTS), is a flexible hybrid method that spawns *derived models* that exploit the strength of meta-heuristics adept at solving certain problems. HASTS employs ACO and Tabu Search as the component heuristics, in particular using ACO as the construction heuristics and Tabu search as the local improvement heuristics. By varying the degree of importance of the inherent algorithms, various derived models are easily formulated to solve subsets of the problem.

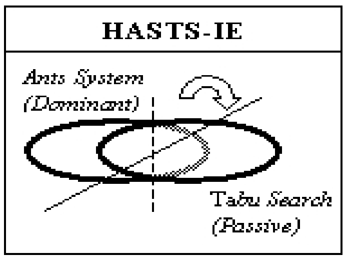
The intrinsic flexibility and potential for heuristical collaboration of MDF allows HASTS to vary the importance of the component heuristics. ACO and TS are argued to be good complements to each other, as ACO works using a preference list, given by the pheromone trail, while TS operates using a forbidden (or tabu) list. The algorithmically opposite techniques offered a high potential that when one algorithm reaches a local optimal, the other algorithm has a higher chance of bring it out and improving the solution henceforth.

HASTS improves results by adjusting the importance level and degree of collaboration of the component meta-heuristics in the hybrid technique, via the framework provided by MDF. Each variant of HASTS has a set of algorithms as the *core* algorithm, while the other algorithm(s) serves as the *aide* algorithm(s). Each of these variant becomes a *derived model* of HASTS. The advantage of the derived models lies in the ability to adapt search to exploit the strength and cover the weakness of the meta-heuristics under the scheme. As such, HASTS is especially suitable for solving complex problems using a divide-and-conquer approach, by first breaking down and identifying the objectives of the sub-problems, and solving these using the best approach optimally.

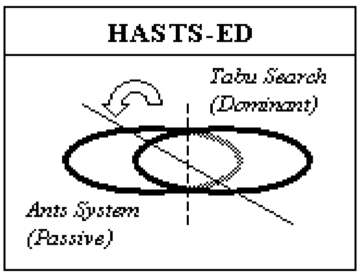
Figure 7.1 showed four possible derived models of HASTS, extracted from [Lau et al.[2], 2003], which saw use in solving the IRPTW, by adjusting the relative importance of the hybrid collaborators to adapt to the needs of each different sub-problems. The framework design also ensured that information sharing between the sub-problems can be easily achieved, using the shared components provided by the MDF framework, to potentially yields better results than if a separate technique should solves the problem (or sub-problems) individually.



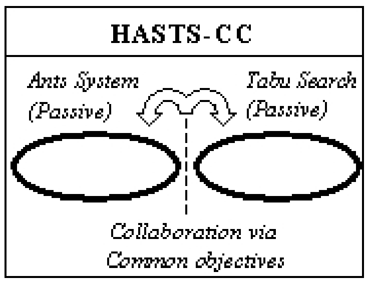
(A) HASTS-EA



(B) HASTS-IE



(C) HASTS-ED



(D) HASTS-CC

Figure 7.1: Derived Models of HASTS

The four derived models are respectively *Empowered Ants* (HASTS-EA) (Figure 7.1(A)), *Improved Exploitation* (HASTS-IE) (Figure 7.1(B)), *Enhanced Diversification* (HASTS-ED) (Figure 7.1(C)), and *Collaborative Coalition* (HASTS-CC) (Figure 7.1(D)). The framework design ensured that each of these derived models reuses the same implementation for each of the component algorithms. The difference is mainly in where to separate the algorithm, as well as the communication between the algorithms. Hence, for HASTS, MDF guarantees that a generic ACO and TS component engine can be used.

HASTS-EA *(Empowered Ants)*

This derived model arises from the hypothesis of sections 4.5.1 and 4.5.2 that when the ants system reaches local optimal solutions, it suffers from a tendency of solution cycling in the near optimum region due to their emphasis on the strong pheromone trails. By empowering the ants with memory, it reduces the chances of reconstructing the same solution. An analogy can be drawn where each ant becomes more intelligent to find a better trail by *not* following false tracks laid by previous ants. Tabu search uses a tabu list to reduce cycling on the same set of solutions. While the ants system optimizes the solution based on its pheromone trails as a “preference” memory, solution cycling is reduced via the tabu list. Furthermore, tabu search can be applied to modify the solutions radically, hence encouraging exploration that helps to escape from local optimality. This implementation, however, suffers from a slight increase in computational needs, as well as more memory for the additional tabu list. This tradeoff is usually justified by the increase in performance, especially over large iterations. From an implementation viewpoint, HASTS-EA modifies ACO to include a tabu list, which records the solution made by each ant in a single iteration. Subsequently, each ant in the iteration would check if the next move is tabu-ed. If it is, the move will be dropped and a new move will be generated. The tabu list is reset at the end of the iteration. A pseudo-code of HASTS-EA is shown in Figure 7.2.

```
Procedure: HASTS – EA ()
    While (termination-criterion-not-satisfied)
            While (Max_Ant_Not_Reached)
                    Ants_generation_and_activity
                    Pheromone_Evaporation
                    Reset_Tabu_List
                    Daemon_actions
            end Schedule_activities
    end While
end Procedure

Procedure: Ants_generation_and_activity ()
    While (available_resources)
            Schedule_creation_of_new_ant
            New_Solution = New_active_ant
        update_Tabu_List (New_Solution)
    end While
end Procedure

Procedure: New_active_ant ()
    Initialize_ant;
    M = read_Pheromone Trail
    T = read_Tabu_List
    While (current_state != target_state)
            A = read_local_ant_routing_table
            P = compute_transitional_probabilities (A, M)
            Foreach Next_state do
                    Next_state = apply_ant_decision_policy(P)
            end Foreach
            While (check_Tabu_List (Next_state) == non-tabued)
            Move_to_next_state (next_state)
            If (online_step-by-step_pheromone_update)
                    Deposit pheromone
                    Update M
            end If
    end While
    If (online_delayed_pheromone_update)
            Foreach visited_arc      do
                    Deposit pheromone
                    Update M
            end Foreach
    end If
end Procedure
```

Figure 7.2: *Pseudo-code of HASTS-EA*

HASTS-IE *(Improved Exploitation)*

In this model, tabu search is embedded in ACO to conduct *intensification* search on the best solution. A similar design has been employed in [Stutzle and

Dorigo, 1999] to produce good solutions for TSP. This model offers two advantages. First, by updating the pheromone trail only after intensifying the best solution, we increase the probability of finding a better solution by subsequent ants. Second, due to the probabilistic guided nature of ants system, this narrows the chances of reaching an optimal solution if it happens to be radically different from local optimum. For example, it is well known that for TSP, the ants system may take a long time before it reaches optimality, due to the presence of "crossings" in the tour, such as those in Figure 7.3. With the help of tabu search, such crossings can be eliminated easily by swap moves such as 2-opt. HASTS-IE, on the other hand, is computational expensive, though it can be extremely effective in situations with many "crossings" in the solution.

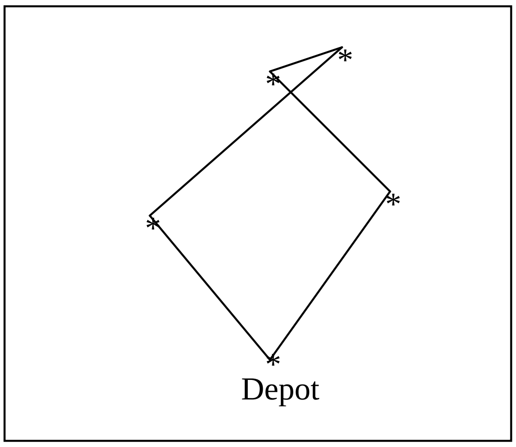


Figure 7.3: *Example of a "crossings"*

HASTS-ED ***(Enhanced Diversification)***

In this model, ants system is proposed as a diversifier for tabu search. As tabu search suffers from local optimality, a diversification strategy is to apply another meta-heuristic as a diversifier [Li and Lim, 2001]. HASTS-ED uses ACO as the TS diversifier with the following rationales. First, the probabilistic nature of the ants system gives a higher chance of successfully diversifying from the local optimum. Second, the diversifier should make a radical move from the current solution so as to explore new regions. Although a random restart is a good strategy, the new starting solution is often poor. Ants system provides a remedy to this by reconstructing quality solutions. However, appropriate parameters for the ACO diversifier should be set, such as a low q_0 that is unusually in most other effective ACO implementation.

HASTS-CC *(Collaborative Coalition)*

HASTS-CC proposes a collaborative coalition between the ants system and tabu search. This model offers the least coupling between the two meta-heuristics but allows great flexibility in the formulation of the problem. One configuration of HASTS-CC is to espouse the two-phase approach as advocated by [Schulze and Fahle, 1997]. This approach consists of a construction phase follow by an optimization phase. ACO work extremely well for the construction phase as it could be used independently to obtain quality solutions. Being an optimization heuristic, tabu search fit naturally into the second phase of the approach. Such collaboration exploits the natural heritage of each meta-heuristic as noted by section 4.5.4.

7.2 VRPTW

The problem being solved in this instance, the VRPTW, is an NP-hard multi-objective optimization problem. Traditional approach in solving VRPTW involves projecting all objectives into a single dimension. However, the correlation between these various objectives are usually weak and difficult to express using a common aspect. In addition, during the search, the optimizer has no insight to which objective it is improving. This resulted in redundancy spent in optimizing the secondary objectives while the primary objective is being optimized. To resolve this, an approach is to optimize the problem by independently considering each of its objectives, allowing precise strategies to be employed. In solving this problem, a decision can be made to decompose the problem into the following objectives:

1. *Minimize the number of vehicles given a set number of customers. The dual problem is to maximize the total number of customers given a set of vehicles.*
2. *Minimizes the total distance traveled given a fixed set of vehicles.*

This divide-and-conquer formulation suggests the suitability of using HASTS. HASTS, described in the previous section, is the solver for VRPTW used to obtain the results presented in the next chapter. As had been mentioned earlier, each derived model of HASTS share the same implementation for the component algorithm. It is also seen that VRPTW is an extension of the TSP in Chapter 3. Hence, in the implementation, HASTS utilizes a generic ACO implementation for TSP built from the ACO framework (ACF), and reuse this implementation with modifications to handle the additional constraints in VRPTW, to provide an ACO solver for VRPTW. This solver is then extended by each derived model, and modified according to the specifications of the sub-problem it is assigned to solve. Figure 7.4 shows the evolution of the ACO implementation in solving VRPTW using HASTS.

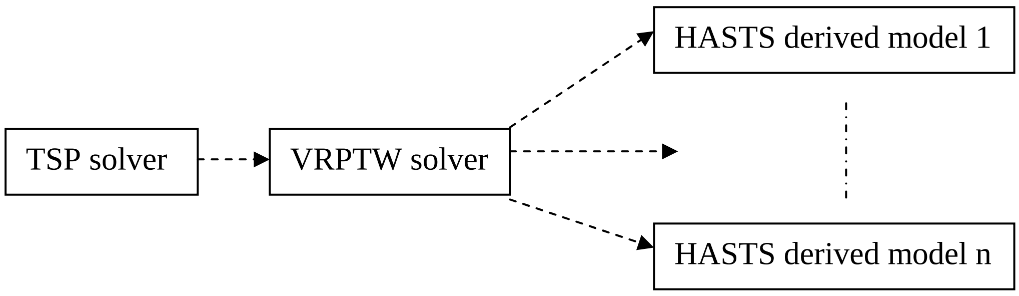


Figure 7.4: Reuse of ACO implementation

For this problem, HASTS requires only two derived models, HASTS-IE and HASTS-ED described earlier.

Objective 1: *Minimize the number of vehicles given a set number of customers. The dual problem is to maximize the total number of customers given a set of vehicles.*

This objective can be reformulated to its dual model and writing it as maximizing the customers served in given a set of vehicles, and reduce the required vehicles each time a solution that serves all the customers is found with the lesser fleet size. The *HASTS-EA* derived model is appropriate for this sub-problem. ACO is a good meta-heuristic for this objective as it optimizes the solution quality through reconstruction. TS, although possible, is not a suitable candidate as it tries to 'pull' the solution to feasibility through optimizing the customers' sequence in the tour, which is a slow process. Instead, tabu search is used to empower the ant system by intelligently rupturing the pheromone trails left by the ants, and in doing so, helped the ants from being ensnared in a local optimum.

Initially *m* vehicles are obtained by applying a greedy heuristic to serve all customers. The algorithm then reduces the value of *m* by 1 and seeks to construct a feasible solution that services all the customers. Once a feasible solution is found, the number of vehicles is reduced to the best-found number of vehicles and the process is repeated for a new feasible solution. Figure 7.5 provides the pseudo-code fragment for the event used to solve this objective. This sub-problem requires search so as to find a configuration where the customers can fit into the pre-set vehicles. HASTS-EA performs well since the tabu list assists each ant in an iteration to construct a radically different solution. Although other derived models can also be used, they lack of the intensified exploration that HASTS-EA provides.

```
Class DecreasingFleetSizeEvent : Event
{
    int m = MAX_FLEET_SIZE;     // m = fleet size of best found solution
    Solution* initSol = get_initial_solution();
    virtual bool ACO_started (ACO* aco) {
        m = initSol->vehUsed – 1      // attempt to find solution ≤ initSol->vehUsed – 1
    }
    virtual bool afterActiveAnt(ACO* aco) {
        Solution* currentSol = aco->getCurrentSolution();
        if (currentSol->visited_cust(m) > tempSol->visited_cust(m)) {
            tempSol->copyFrom(currentSol);
            if (tempSol->vehUsed <= m) {
                m = tempSol->vehUsed – 1;
            }
        }
    }
}
```

Figure 7.5: *Code Fragment implementation for VRPTW objective 1*

Objective 2: *Minimizes the total distance traveled given a fixed set of vehicles.*

Objective 2 is attempted after Objective 1 had been optimized, and as a result, this sub-problem will consist of a tighter solution space. In spite of the success by HASTS-EA in optimizing the number of vehicles, this derived model is not very effective for this objective because of the difficulties involved in constructing different feasible solutions on an allowed number of vehicles due to the nature of ACO. Instead, another derived model, *HASTS-ED*, is employed to minimize the total distance on a fixed set of vehicles. HASTS-ED uses tabu search as the core heuristic with ants system acting as the diversifier. Tabu search is effective in solving this sub-problem as it optimizes the route distance rather than reconstructs the solutions. However, tabu search still faces the danger of being entrapped in a local optimum during its search. To address this issue, when tabu search encounters a local optimum, it randomly selects some of the routes to be reconstructed by ACO, which assists tabu search by radically re-configuring the selected partial routes. Details on this objective relies mainly on the operations of Tabu Search and is examined in further detail in [Lau et al.^{2}, 2003].

7.3 IRPTW

Chapter 3 previously described one approach by [Lau et al. 2000] to solve the IRPTW, an example of a multiple periods and multiple constraints problem, by decomposing this complex problem into the relatively simpler VRPTW and DLP. Since VRPTW can be further broken down using its separate objectives as described in the previous sub-section, IRPTW then can be formulated to the following three sub-objectives:

1. *Minimize the number of vehicle used subject to customer time windows of the given set of customers.*
2. *Minimize the total distance traveled, subject to customer time windows and the given fleet of vehicles.*
3. *Minimize the inventory holding and backlog costs, subject to the vehicle capacity and retailer holding capacity constraints.*

It can be seen that objectives 1 and 2 forms the VRPTW part of the problem, while objective 3 specifies the DLP sub-problem. Having previously used HASTS to solve VRPTW, it become logical to reuse this implementation to solve IRPTW once it was apparent IRPTW can be broken down into the VRPTW and DLP.

Objective 1: *Minimize the number of vehicle used subject to customer time windows of the given set of customers.*

and

Objective 2: *Minimize the total distance traveled, subject to customer time windows and the given fleet of vehicles.*

These two objectives, being equivalent sub-problems of the VRPTW, allow easy reuse of the HASTS-EA and HASTS-ED derived model. The reuse of ACF and TSF are both trivial and natural.

Objective 3: *Minimize the inventory holding and backlog costs, subject to the vehicle capacity and retailer holding capacity constraints.*

In order to reduce inventory or backlog, more frequent deliveries have to be made, hence increasing the transportation cost. Hence, the goal here is to minimize the number of retailers (or customers) served each day without increasing the total cost. That is, the objective is to delete retailers from routes in a manner that does not incur additional costs. Many techniques are available to handle this objective, but in line with reusing HASTS, which is already used to solve the problem involving the other two objectives, it is a straightforward matter to reuse the same ACO and TS engines by employing another derived model catered to the problem, *HASTS-IE,* such as in Figure 6.6. HASTS-IE uses ACO to construct different solutions. It then uses tabu search to improve its exploitation to reduce missing elite solutions. Figure 7.6 presented the pseudo-code fragment for the event class solving this objective. The tabu search uses the standard add, delete and swap moves that attempt to improve the solution quality found by the ACO. The output is a distribution plan that induces the set of customers to be served for objective 1, to facilitate iterative improvement. Since this objective involves multiple time-periods, the ACO implementation in HASTS-IE employs Scheme 4.

```
Class ImprovedExploitationEvent : Event
{
    virtual bool newCurrentSolutionFound(ACO* aco, Solution* currentSol) {
        TabuSearch* TS = getTSEngine();
        TS->restart(currentSol, iterations);
        TS->startSolving();
        currentSol = TS->getBestSolution();
    }
}
```

Figure 7.6: *Code Fragment implementation for IRPTW objective 3*

7.4 MPMKP

MPMKP is a single dimensional extension of the classical MKP, with the additional dimension being the multiple time-periods. This increases the number of constraints in carrying over requests and inventory from one time-period to the next. Furthermore, unlike IRPTW, MPMKP need not be broken down into multiple objectives, since it is a single dimensional extension. IRPTW is extended from VRPTW in more than just time-periods, but also several additional constraints. Hence, for MPMKP, the same objective as MKP is optimized.

MKP Objective: *Maximize* $\sum_{j=1}^{n} p_j x_j$ *, i.e., Maximize the amount of profit that can be obtained, subject to the (known) constraints.*

MKMKP Objective: *Maximize* $Z = \sum_{j=1}^{N}\sum_{t=1}^{T} p_{jt}x_{jt}$, *i.e., Maximize the amount of profit that can be obtained, subject to the (known) constraints and inventory top-up rate over the (known) time-period.*

With the difference over MKP in the additional time-period dimension, we simply replace the pheromone trail using a node-to-time-period structure over a node-to-node structure directly, without need to replace anything else, as described by Scheme 4. Figure 7.7 first presented the pseudo-code for a MKP implementation, and Figure 7.8 then presented the pseudo-code that extends from the MKP implementation to solve the MKMKP.

```
Class MKPACOParameters : ACOParameters
{
    double pheromoneTrail[MAX_REQUEST] [MAX_REQUEST];
}

Class MKPEvent : Event
{
    ACOParameters* = param;
    virtual bool beforeActiveAnt(ACO* aco) { // j : items
        items[j] = capacity[j];
    }
    virtual double* calculateProbabilityOfNodes(ACO* aco) {
        for (all valid request r) {                 // i : current node
            double sumTightness = 0.0;
            for (all items j) sumTightness += request[r]->amount[j] / items[j];
            visitProbability[r] = (profit[j] / sumTightness)^α * (pheromoneTrail[i][r])^β
        }
        return visitProbability;
    }
}
```

Figure 7.7: *Code Fragment implementation for MKP*

```
Class MPMKPACOParameters : ACOParameters
{
    double pheromoneTrail[MAX_PERIOD] [MAX_REQUEST];
}

Class TimeSchemeEvent : Event
{
    ACOParameters* = param;
    virtual bool beforeActiveAnt(ACO* aco) { // t : period; j : items
        items[t][j] = capacity[j];
        cumItems[t][j] = capacity[j]*(t+1)        // cumulative items
    }
    virtual double** calculateProbabilityOfNodes(ACO* aco) {
        for (all valid request r) {
            double sumTightness = 0.0;
            for (all items j) sumTightness += request[r]->amount[j] / cumItems[T-1][j];
            visitProbability[r][t] = (profit[t][j] / sumTightness)^α * (pheromoneTrail[t][r])^β
        }
        return visitProbability;
    }
```

Figure 7.8: Code Fragment implementation for MPMKP

Note the similarity between the pseudo-code shown in Figure 7.7 and Figure 7.8. The extension to MKMKP from MKP is one-dimensional only in the time-period context. As such, MPMKP is not overly concerned with the ordering of nodes within each time-period (like in IRPTW), and the necessary changes involved mainly the structure of the pheromone trail (advocated by Scheme 4) as well as related modifications to calculations using the pheromone trail. It should be noted that all other factors of the implementation (such as the basis of the formula for the calculation of the probability; the weight ratio of pheromone to local heuristics; the relevance of item tightness to the problem; etc.) follows from generic implementations used to solve the MKP in the literature [Fidanova[1], 2002; Fidanova[2], 2002; Leguizamon and Michalewicz, 1999].

CHAPTER 8

Results and Discussions

This chapter presents the specific problem parameters and results obtained using the approaches described in the previous chapter for three NP-hard problems, namely the VRPTW, IRPTW and the MPMKP, as well as discussions on the results obtained. All ACO implementations use the proposed parameter values by [Dorigo et al., 1991] of *a* = 1, *b* = 5, *numberOfAnts* = 10, all *decay* values of 0.2, and q_0 = 0.8.

8.1 Results for VRPTW

VRPTW, as mentioned, as extended from the TSP. The classical and most common comparison for VRPTW solvers in the literature is with the Solomon's VRPTW benchmark [Solomon, 1987], consisting of a total of 56 test cases covering different scenarios. These test cases included a set of problems consisting of *Clustered* nodes (C101-C109, and C201-208), which generally is best solved by assigning vehicles to service the same or nearby clusters in the problem; a set of problems consisting of *Random* nodes (R101-R112, and R201-R211), which has nodes randomly assigned, and solving it optimally will be problem specific; and a set of problems consisting of a combination of *Random and Clustered* nodes (RC101-108, and RC201-208). Table 8.1 tabulates the results obtained.

Table 8.1: *Results for VRPTW from the Solomon's original test cases (n=100) in (fleet size/distance)*

Test cases	**TS**	**ACO**	**HASTS**
C101	10/828.94	10/855.07	10/828.94
C102	10/852.97	10/1072.24	10/845.61

C103	10/858.62	10/1435.26	10/840.88
C104	10/856.87	10/1182.64	10/857.57
C105	10/828.94	10/936.47	10/828.94
C106	10/828.94	10/958.91	10/828.94
C107	10/828.94	10/877.99	10/828.94
C108	10/828.94	10/1033.81	10/828.94
C109	10/828.94	10/1900.94	10/828.94
R101	19/1686.24	19/1929.05	19/1686.24
R102	18/1518.93	18/1886.77	18/1493.31
R103	14/1301.64	14/1679.71	14/1301.64
R104	11/1072.04	10/1198.69	10/1025.38
R105	14/1459.84	14/1651.43	14/1458.60
R106	13/1324.38	12/1564.99	12/1314.69
R107	11/1165.87	10/1144.72	10/1140.27
R108	10/1002.56	10/1117.25	10/ 994.66
R109	12/1287.62	12/1502.57	12/1207.58
R110	11/1218.33	11/1348.78	11/1166.65
R111	11/1104.93	11/1239.53	11/1172.66
R112	10/1039.55	10/1242.24	10/1041.36
RC101	15/1742.29	15/1899.97	15/1698.50
RC102	13/1605.30	13/1780.98	13/1551.32
RC103	11/1337.04	11/1567.12	11/1371.40
RC104	11/1249.13	10/1353.87	10/1187.97
RC105	15/1633.39	14/1899.54	14/1618.01
RC106	12/1428.88	12/1620.67	12/1434.33
RC107	12/1312.84	11/1468.59	11/1266.92
RC108	11/1258.40	10/1326.94	10/1273.12
C201	3/591.56	3/ 591.56	3/ 591.56
C202	3/591.56	3/ 993.62	3/ 591.56
C203	3/617.32	3/1065.81	3/ 605.23
C204	3/673.46	3/1046.87	3/ 594.80
C205	3/604.67	3/ 913.03	3/ 588.88
C206	3/632.35	3/ 647.29	3/ 588.49
C207	3/621.02	3/ 646.69	3/ 588.49
C208	3/588.88	3/ 646.72	3/ 588.49
R201	4/1308.84	4/2048.31	4/1366.34
R202	4/1123.34	3/1755.11	3/1239.22
R203	3/1013.59	3/1625.26	3/1000.29
R204	3/817.60	3/1159.14	3/ 781.86
R205	4/1022.02	3/1678.53	3/1063.29
R206	4/963.94	3/1525.34	3/ 955.34
R207	3/863.60	3/1258.12	3/ 866.35
R208	3/761.94	2/1016.07	2/1016.07
R209	4/934.45	3/1551.01	3/ 979.30
R210	3/1000.53	3/1659.90	3/ 968.32
R211	3/816.33	3/1143.96	3/ 865.51
RC201	4/1704.92	4/2226.23	4/1445.00
RC202	4/1265.78	4/1878.00	4/1204.45
RC203	3/1118.19	3/1706.48	3/1091.71
RC204	3/884.70	3/1342.81	3/ 826.27
RC205	4/1435.06	4/2271.26	4/1469.25
RC206	4/1162.96	3/1717.62	3/1259.12

RC207	4/1178.01	3/1733.47	3/1127.19
RC208	3/931.76	3/1422.07	3/ 937.78

Table 8.1 is read as follows: *TS* refers to the results obtained using a standard Tabu Search implementation on MDF-TSF. *ACO* refers to the results after passing the data through derived model HASTS-EA, a predominantly ACO technique implemented with MDF-ACF that focus on solving the first objective (minimizing the fleet size of vehicles). Finally, the *HASTS* column tabulates the results obtained after the entire HASTS process mentioned earlier – in effect after a combination of HASTS-EA and HASTS-ED.

Note the effectiveness of the hybrid HASTS compared against TS and ACO, which adequately showed the effectiveness of MDF and a divide and conquer hybrid approach. Also, the results from TS are generally better than ACO in this instance due to the different objectives of the approach. TS has an objective of minimizing distance, and perform it so well that for some instances, such as R202, it performs better in terms of distance, but is worse off by the problem definition specifying the fleet size as primary priority, while the ACO results focuses mainly on reducing the fleet size of vehicles.

It should also be further noted that the development of the TS implementation takes about 3 months man-hours, while the ACO implementation takes a lesser amount of time at about 2 months, due to its simpler nature. Meanwhile, with the availability of both software frameworks, HASTS requires only less than a week man-hours to develop.

8.2 Results for IRPTW

The results for IRPTW are obtained from an implementation that reuses the implementation for the VRPTW. There are no well-known sets of test cases for the IRPTW, but there are implementations in the literature that extends the set of test cases from the Solomon's benchmark for the VRPTW with additional constraints. This thesis provides results for solving the same set of problems using the test instance generation strategy in [Lau et al.[1], 2002], which provided a good set of test cases for IRPTW.

Specifically, the planning period is 10 days. The vehicle capacity, locations and time-windows of the customers and depot are as specified in the corresponding Solomon instances. The demand d_{it} of customer i for day t ($t=1,\ldots,10$) is equal to the demand d_i of the Solomon instance, by partitioning the value $10*d_i$ into 10 parts, i.e. d_{i1}, $d_{i2},\ldots,d_{i,10}$ randomly such that d_{it} is within the range [$0.5*d_i$, $1.5*d_j$]. The capacities of consumers and warehouse are the vehicle capacity and infinity respectively. As for cost coefficients, the inventory cost and backlog cost for each customer are 1 and 2 respectively, symbolizing a preference to holding a unit of inventory over a day than suffer a lost of customer trust on a backlog of a corresponding unit of inventory. The transportation cost of each route is 10 times its total distance.

Table 8.2 shows the results of the test cases extracted from [Lau et al.[1], 2002] in comparison with the proposed approach. Only extended test cases from the C2 and R2 series are tabulated, which this thesis corresponding match. Furthermore, RC2 results are also provided to aid further studies on the problem. The columns *VRPTW*, *ILS+VRP* and *TS+VRP* denote the results obtained from [Lau et al.[1], 2002], where VRPTW is the approach taken from adopting a standard two-phase heuristics; ILS+VRP is the results obtained using Iterated Local Search [Gu 1992; Johnson 1990]; and TS+VRP employs a Tabu Search technique. The column HASTS presents the results obtained using our proposed hybrid algorithm implemented from the MDF (ACF+TSF).

Table 8.2: *Results for IRPTW extended from Solomon's original test cases in (costs)*

Test Cases	**VRPTW**	**ILS+VRP**	**TS+VRP**	**HASTS**
C201	178650	113263	112821	54905
C202	192818	117483	124312	53404
C203	200615	131920	122055	53620
C204	216447	136384	142300	54778
C205	175378	116147	109248	51907
C206	177331	123978	127876	50507
C207	177447	122204	117735	51453
C208	175268	124110	125667	52501
R201	304779	111330	116893	85014
R202	291492	116982	114717	70533
R203	247122	110215	115070	68865
R204	227381	114118	114118	61944
R205	284759	122333	123009	73455
R206	260760	120928	123251	64652
R207	223527	115438	115438	63697
R208	338033	120011	117255	59285
R209	249036	116840	120725	69200
R210	-	-	-	69545
R211	-	-	-	61816
RC201	-	-	-	97417
RC202	-	-	-	87245
RC203	-	-	-	80114
RC204	-	-	-	71795
RC205	-	-	-	92560
RC206	-	-	-	86144
RC207	-	-	-	83326
RC208	-	-	-	71740

Results for the VRPTW, ILS+VRP, and TS+VRP columns are obtained on a Pentium 666MHz machine, while the results from the HASTS column is obtained on a Pentium 1.13GHz machine, which is estimated to perform at twice the power. As such, for comparison, the HASTS implementation is obtained under 90 seconds, to compensate for the 180 seconds upper bound used for the other implementations.

With the objective being to minimize the cost, Table 8.2 amply showed that HASTS offers a set of superior results compared to previous works. While this could be due in part to the originality of IRPTW in the literature, and hence not well studied as yet, it can still be claimed that the effectiveness when solving VRPTW is not lost when reused to solve IRPTW. Furthermore, it can be seen that the framework provided generality and flexibility for reuse, which enabled development to take minimal effort and implementation to be achieved in less than 2 weeks man-hours.

8.3 Results for MPMKP

The results presented here are those using a Tabu Search approach obtained from a related work [Lau et al.[2], 2002], compared against an ACO-only implementation from ACF proposed in this thesis, to reflect the effectiveness of the scheme.

The set of test cases solved in this instance is extended from the benchmark test cases for MKP found in the OR-LIB (http://www.ms.ic.ac.uk/jeb/pub/). OR-LIB has a set of 0-1 MKPs generated with varying size and tightness. Te benchmark problems are divided into nine sets, each with a given number of items, *m*, and number of requests, *n*. There are 30 test cases within a set. The first 10 test cases are generated with a tightness ratio, a, of 0.25. The next 10 test cases are generated with a = 0.5, and the last 10 test cases are generated with a = 0.75. The parameters used in each test case are generated as follows:

$$a_{ij} = U(0,1000) \qquad \text{……….} \ (13)$$

$$b_i = a\sum_{i=1}^{n} a_{ij} \quad \text{for } a \in \{0.25, 0.5, 0.75\} \qquad \text{……….} (14)$$

$$p_j = \sum_{i=1}^{m} \frac{a_{ij}}{m} + 500U(0,1) \qquad \text{……….} (15)$$

where a_{ij} is the consumed quantity of item *i* if object *j* is fulfilled

b_i is the total inventory for item *i* ($1 \le i \le m$)

p_j is the profit of object *j* ($1 \le j \le n$)

However, this set of benchmark is for 0-1 MKP, and hence can only be used as test cases for the single-period multi-dimensional knapsack problem. For this instance, 6 sets of 7-period MPMKPs (equivalent to 7 days of a week) are generated. Table 8.3 demonstrates how to generate a set of 9 test cases for the MPMKP using the specified 7-period planning horizon from the 30 test cases in mknapcb*X* (1 ≤ *X* ≤ 9). The first test case is generated using the first 7 test cases in mknapcb*X*. The rest of the test cases are generated by choosing another 7 test cases from the mknapcb*X* test set. The main parameters in each group are *n*, the number of requests in each periods, and *m*, the number of items.

Table 8.3: *The nine test cases generated from mknapcbX*

Test Cases	**mknapcbX test cases chosen**	a
mknapcbX-m7-1-n-M-linear	1,2,3,4,5,6,7	0.25
mknapcbX-m7-2-n-M-linear	2,3,4,5,6,7,8	0.25
mknapcbX-m7-3-n-M-linear	3,4,5,6,7,8,9	0.25
mknapcbX-m7-4-n-M-linear	11,12,13,14,15,16,17	0.50
mknapcbX-m7-5-n-M-linear	12,13,14,15,16,17,18	0.50
mknapcbX-m7-6-n-M-linear	13,14,15,16,17,18,19	0.50
mknapcbX-m7-7-n-M-linear	21,22,23,24,25,26,27	0.75
mknapcbX-m7-8-n-M-linear	22,23,24,25,26,27,28	0.75
mknapcbX-m7-9-n-M-linear	23,24,25,26,27,28,29	0.75

Table 8.4 below compares the results obtained by heuristics methods. The Z_{TSA} column reflects results obtained using a Tabu Search approach [Lau et al.[2], 2002], which has a good Greedy Algorithm as the construction phase algorithm, while Z_{ACO_S3} and Z_{ACO_S4} shows the results obtained using implementations of ACF with Scheme 3 and Scheme 4 respectively. Z_{TSA} directly reference the results produced by [Lau et al.[2], 2002].

For purpose of this thesis, ACO Scheme 4 is limited to within 180 seconds, to show the capability of ACO to produce good solutions fast, as well as a basis of comparison between the heuristic approaches, while Scheme 3 which is hypothetically weaker is allowed a complete run on 1000 iterations with 10 ants.

Table 8.4: ***Results for mknapcbX ($1 \le X \le 9$) set of test cases in (profits)***

Test cases	Z_{TSA}	Z_{ACO_S3}	Z_{ACO_S4}
mknapcb1-m7-1-100-5	170394	166108.80	172991.50
mknapcb1-m7-2-100-5	170403	167630.00	174511.50
mknapcb1-m7-3-100-5	165766	163075.80	169773.70
mknapcb1-m7-4-100-5	301046	298855.40	305530.00
mknapcb1-m7-5-100-5	302202	297823.40	305165.70
mknapcb1-m7-6-100-5	300691	292329.70	301130.00
mknapcb1-m7-7-100-5	421542	415505.30	422760.60
mknapcb1-m7-8-100-5	426364	421160.10	428271.50
mknapcb1-m7-9-100-5	425243	424004.80	430210.40
mknapcb2-m7-1-250-5	422142	413640.40	419683.10
mknapcb2-m7-2-250-5	423560	418154.50	423564.00
mknapcb2-m7-3-250-5	422531	421054.40	426623.40
mknapcb2-m7-4-250-5	762490	760213.90	766470.50
mknapcb2-m7-5-250-5	766521	758573.50	765164.90
mknapcb2-m7-6-250-5	761451	750241.40	756247.30
mknapcb2-m7-7-250-5	1050350	1043238.40	1051298.40
mknapcb2-m7-8-250-5	1049140	1052362.80	1059225.60
mknapcb2-m7-9-250-5	1050460	1062571.90	1069669.60
mknapcb3-m7-1-500-5	842117	830083.10	835461.30
mknapcb3-m7-2-500-5	841800	835786.70	842117.00
mknapcb3-m7-3-500-5	848891	858047.80	863560.60
mknapcb3-m7-4-500-5	1542250	1532538.90	1539289.40
mknapcb3-m7-5-500-5	1539590	1542718.20	1548729.80
mknapcb3-m7-6-500-5	1523470	1529134.10	1534857.90
mknapcb3-m7-7-500-5	2107850	2086118.00	2094827.10
mknapcb3-m7-8-500-5	2114540	2129949.20	2135511.20
mknapcb3-m7-9-500-5	2106420	2110473.10	2115925.40

mknapcb4-m7-1-100-10	162173	154759.50	164829.00
mknapcb4-m7-2-100-10	159706	149624.70	159609.00
mknapcb4-m7-3-100-10	161311	152395.80	163004.00
mknapcb4-m7-4-100-10	301580	283156.90	294177.00
mknapcb4-m7-5-100-10	306173	303296.70	312527.20
mknapcb4-m7-6-100-10	296748	294717.00	303522.60
mknapcb4-m7-7-100-10	414000	388742.50	399290.30
mknapcb4-m7-8-100-10	420116	407112.20	415442.70
mknapcb4-m7-9-100-10	419433	408924.70	417419.10
mknapcb5-m7-1-250-10	410521	406566.70	417525.6
mknapcb5-m7-2-250-10	414048	403700.80	414371.00
mknapcb5-m7-3-250-10	411796	400603.70	411412.50
mknapcb5-m7-4-250-10	760319	761432.70	772740.60
mknapcb5-m7-5-250-10	760858	755473.40	768019.1
mknapcb5-m7-6-250-10	758738	752282.50	762944.30
mknapcb5-m7-7-250-10	1059590	1049603.20	1059568.00
mknapcb5-m7-8-250-10	1056350	1049558.70	1058926.10
mknapcb5-m7-9-250-10	1058770	1043742.00	1051996.90
mknapcb6-m7-1-500-10	821095	809800.50	818143.40
mknapcb6-m7-2-500-10	826232	820644.40	829577.60
mknapcb6-m7-3-500-10	825980	826518.00	834593.10
mknapcb6-m7-4-500-10	1502780	1508633.60	1518613.30
mknapcb6-m7-5-500-10	1518870	1507663.00	1519139.60
mknapcb6-m7-6-500-10	1512120	1509724.60	1520753.90
mknapcb6-m7-7-500-10	2106780	2100039.90	2113689.30
mknapcb6-m7-8-500-10	2107940	2081950.70	2095697.90
mknapcb6-m7-9-500-10	2103260	2091771.60	2104327.60
mknapcb7-m7-1-100-30	156032	142178.50	157007.80
mknapcb7-m7-2-100-30	156368	141355.00	157205.00
mknapcb7-m7-3-100-30	154808	141638.20	156960.40
mknapcb7-m7-4-100-30	289066	268620.40	285293.20
mknapcb7-m7-5-100-30	294527	272348.40	289700.00
mknapcb7-m7-6-100-30	294653	268390.60	285656.20
mknapcb7-m7-7-100-30	410677	390187.30	403545.20
mknapcb7-m7-8-100-30	417794	400152.80	414011.40
mknapcb7-m7-9-100-30	415666	381110.70	392733.50
mknapcb8-m7-1-250-30	399999	382107.60	401183.90
mknapcb8-m7-2-250-30	401153	386723.90	405881.50
mknapcb8-m7-3-250-30	399017	382146.10	403577.00
mknapcb8-m7-4-250-30	747851	728399.60	752444.60
mknapcb8-m7-5-250-30	750229	726118.70	745073.40
mknapcb8-m7-6-250-30	747512	727883.60	749204.90
mknapcb8-m7-7-250-30	1039310	1013280.80	1031130.80
mknapcb8-m7-8-250-30	1054980	1026651.60	1045940.70
mknapcb8-m7-9-250-30	1052480	1017275.90	1039318.60
mknapcb9-m7-1-500-30	792223	788881.80	808691.90
mknapcb9-m7-2-500-30	801077	785644.40	807627.50
mknapcb9-m7-3-500-30	806659	791132.70	814975.3
mknapcb9-m7-4-500-30	1500320	1488875.30	1510488.60
mknapcb9-m7-5-500-30	1496240	1486699.30	1514131.40
mknapcb9-m7-6-500-30	1499550	1476969.40	1500447.20
mknapcb9-m7-7-500-30	2097630	2060330.90	2086525.60

mknapcb9-m7-8-500-30	2101080	2065677.20	2091225.70
mknapcb9-m7-9-500-30	2112110	2070480.50	2092516.40

The table shows that out of the 81 test cases executed, the ACF implementation with Scheme 4 yields better results for 53 test cases, with a considerably small margin for many of these cases. On the other hand, for the 28 test cases where TS is better than ACO Scheme 4, the profit value had a lower ratio compared to the average performance of the other 53 test cases. Meanwhile, ACO Scheme 3 is comparatively weaker than both TS and Scheme 4. However, the objective of MPMKP, as defined by the available-to-promise problem, is to find good solutions fast, and Table 8.5 tabulates the time taken to achieve the results.

Table 8.5: *Run time for mknapcbX (1≤X≤9) set of test cases in (seconds)*

Test cases	**CPU$_{TSA}$**	**CPU$_{ACO\_S3}$**	**CPU$_{ACO\_S4}$**
mknapcb1-m7-1-100-5	14.87	15.38	15.68
mknapcb1-m7-2-100-5	14.71	5.57	1.92
mknapcb1-m7-3-100-5	14.68	6.33	2.86
mknapcb1-m7-4-100-5	41.77	9.89	7.03
mknapcb1-m7-5-100-5	47.89	8.44	5.37
mknapcb1-m7-6-100-5	51.71	9.45	8.33
mknapcb1-m7-7-100-5	147.69	8.48	4.14
mknapcb1-m7-8-100-5	106.53	8.46	4.28
mknapcb1-m7-9-100-5	123.12	14.48	10.77
mknapcb2-m7-1-250-5	152.51	192.67	12.36
mknapcb2-m7-2-250-5	166.85	293.07	18.10
mknapcb2-m7-3-250-5	128.88	222.57	12.11
mknapcb2-m7-4-250-5	494.74	207.18	19.30
mknapcb2-m7-5-250-5	513.16	142.51	19.54
mknapcb2-m7-6-250-5	529.77	199.06	19.10
mknapcb2-m7-7-250-5	699.59	213.93	113.92
mknapcb2-m7-8-250-5	711.66	113.30	37.89
mknapcb2-m7-9-250-5	706.03	100.51	25.27
mknapcb3-m7-1-500-5	756.36	886.24	345.23
mknapcb3-m7-2-500-5	732.61	951.93	52.52
mknapcb3-m7-3-500-5	643.13	1060.97	52.84
mknapcb3-m7-4-500-5	821.59	933.63	87.69
mknapcb3-m7-5-500-5	849.69	548.14	86.12
mknapcb3-m7-6-500-5	815.20	609.20	118.05
mknapcb3-m7-7-500-5	241.60	655.44	107.5
mknapcb3-m7-8-500-5	236.46	678.25	109.32
mknapcb3-m7-9-500-5	238.99	730.92	105.78

mknapcb4-m7-1-100-10	13.63	5.38	5.15
mknapcb4-m7-2-100-10	13.02	3.75	2.03
mknapcb4-m7-3-100-10	13.40	3.44	2.22
mknapcb4-m7-4-100-10	41.86	8.47	6.92
mknapcb4-m7-5-100-10	46.40	6.66	5.57
mknapcb4-m7-6-100-10	42.41	4.65	3.54
mknapcb4-m7-7-100-10	148.19	5.59	4.68
mknapcb4-m7-8-100-10	194.19	9.19	4.38
mknapcb4-m7-9-100-10	140.76	12.38	6.65
mknapcb5-m7-1-250-10	108.14	88.61	12.48
mknapcb5-m7-2-250-10	107.89	94.72	12.83
mknapcb5-m7-3-250-10	118.30	138.93	12.54
mknapcb5-m7-4-250-10	475.93	136.05	22.33
mknapcb5-m7-5-250-10	501.04	69.51	33.21
mknapcb5-m7-6-250-10	572.33	125.85	22.06
mknapcb5-m7-7-250-10	691.45	129.48	30.40
mknapcb5-m7-8-250-10	744.97	85.79	30.91
mknapcb5-m7-9-250-10	741.83	93.68	45.36
mknapcb6-m7-1-500-10	160.56	453.39	51.65
mknapcb6-m7-2-500-10	161.94	526.34	51.15
mknapcb6-m7-3-500-10	164.45	525.70	79.85
mknapcb6-m7-4-500-10	271.30	682.82	91.91
mknapcb6-m7-5-500-10	270.60	549.18	97.77
mknapcb6-m7-6-500-10	273.84	605.56	100.96
mknapcb6-m7-7-500-10	345.11	587.36	112.00
mknapcb6-m7-8-500-10	338.53	630.59	114.46
mknapcb6-m7-9-500-10	349.79	577.76	170.86
mknapcb7-m7-1-100-30	17.28	6.29	2.78
mknapcb7-m7-2-100-30	18.11	5.99	2.93
mknapcb7-m7-3-100-30	15.93	26.21	26.79
mknapcb7-m7-4-100-30	53.57	12.88	7.35
mknapcb7-m7-5-100-30	61.70	9.64	4.54
mknapcb7-m7-6-100-30	57.02	40.64	37.82
mknapcb7-m7-7-100-30	141.62	60.93	59.33
mknapcb7-m7-8-100-30	158.40	21.97	11.62
mknapcb7-m7-9-100-30	145.26	21.93	11.15
mknapcb8-m7-1-250-30	145.23	60.83	15.96
mknapcb8-m7-2-250-30	140.72	82.88	49.33
mknapcb8-m7-3-250-30	165.32	76.75	16.90
mknapcb8-m7-4-250-30	560.62	122.78	28.91
mknapcb8-m7-5-250-30	599.01	69.53	28.75
mknapcb8-m7-6-250-30	559.06	92.29	28.58
mknapcb8-m7-7-250-30	816.74	107.42	57.06
mknapcb8-m7-8-250-30	858.27	128.04	59.63
mknapcb8-m7-9-250-30	838.59	104.44	39.13
mknapcb9-m7-1-500-30	145.23	470.73	69.17
mknapcb9-m7-2-500-30	286.24	495.76	68.62
mknapcb9-m7-3-500-30	288.55	478.81	80.31
mknapcb9-m7-4-500-30	286.41	571.76	165.83
mknapcb9-m7-5-500-30	500.72	578.27	122.91
mknapcb9-m7-6-500-30	507.14	575.78	121.01
mknapcb9-m7-7-500-30	502.03	756.60	172.06

mknapcb9-m7-8-500-30	642.25	737.34	161.69
mknapcb9-m7-9-500-30	641.26	644.54	170.15

Tables 8.5 sufficiently show the time efficiency with which TSA and the two ACO approaches obtain near-optimal solutions. Interestingly, ACO Scheme 4 comes within 6% deviation of the upper bound obtained from CPLEX[4]). ACO which operates by solution reconstruction of each ant, coupled with an effective scheme, local heuristics and parameters tuning, is effective in gauging the optimality of the problem overall. All the heuristic approaches are also able to provide a constant estimation of the problem optimality at a fixed (early) time, though the random nature of ACO makes it a slightly less suitable candidate in such situations. In particular, this slight disadvantage is however easily redeemed by the speed of Scheme 4. This overall performance enhances our argument that Scheme 4 is an effective ACO scheme for hard scheduling problems. This is further noted when sample implementations using generic ACO approaches, as well as the implementation using Scheme 3 is executed in the lab, where it is seen from Table 8.4 that the results are way under par compared to Scheme 4 or the TSA approach. Furthermore, the CPU time for Scheme 3, seen from Table 8.5, approximates the CPU time for the TSA approach, which is weaker than the Scheme 4 implementation in most cases.

It is also observed that TSA is somewhat dependent on the tightness ratio of the problem, rather than on the problem size. ACO is less sensitive than TSA to the tightness ratio. The disadvantage of ACO, however, is in the intrinsic way it operates using a new solution construction in all iterations, and hence it is more sensitive to problem size. This is as due to its complexity, which in uses an O(xn^2)

[4] The upper bounds were obtained after running CPLEX for 2 hours.

and O(xn^3) ACO implementation for Scheme 4 and Scheme 3 respectively, and hence there is not many iterations that can be executed, unless the code is further optimized. Hence, when the problem size is very large, such as when number of requests is much greater than 500, or the time-period much greater than 7, TSA will prove to be a more effective approach, although in the set of test cases experimented (mknapcb9 being the largest problem size with 500 requests per period and 30 items), ACO Scheme 4 still outperform TSA in terms of the ability to allow for quick decision. Regardless, when the problem size gets large, and given the limitations of the ACO algorithms as discussed in this thesis, the algorithm requires diversification or extreme exploration, or collaboration of ACO Scheme 4 as a construction phase algorithm together with a local search technique like Tabu Search for the optimization phase. The results also proved the potential of the time-period scheme specified by ACO Scheme 4 over a generic scheme like ACO Scheme 3 in dealing with time-period constraints.

CHAPTER 9

Conclusion

This thesis presented the operations of the Ant Colony Optimization, and extracted observations on the algorithm to develop a generic ACO software framework. ACO, which is inspired from the foraging behavior of real ants colony, does not work exactly as how the real ants operate, using additional modifications which adapt it better into an optimizing algorithm, in particular when employing the concept of diversification. However, the modifications have allowed promising applications to many problems, although the terminology generally used in the community, in relation to the real ant colonies, may arguably be inappropriate.

The Ant Colony Framework (ACF) is developed with a twofold purpose capable of serving as a standalone ACO software framework, as well as being a component framework in a higher level Meta-heuristics Development Framework (MDF) that is an overseer framework capable of integrating any number of separate heuristics to aid algorithmic collaboration and performance comparisons.

A primary objective of the thesis is to demonstrate the potential of reuse in the framework, which can decrease developmental resources and increase productivity. In particular, it is shown how ACF, together with other heuristics framework of MDF, can be easily adapted from one implementation solving VRPTW (which uses a TSP implementation not explicitly established in this thesis) to solve IRPTW, which is an extended problem of VRPTW. The VRPTW implementations obtained good results, even when compared against state-of-the-art techniques in the literature, and when reused for IRPTW, the excellent results achieved clearly show the value of reuse in this instance. By induction, it is logical to state that as long as a good implementation is found for a base problem, it is simple to reuse that implementation for similar or extended scenarios of that base problem. While the ACO algorithm can be implemented in a serial and parallel manner, this thesis follows the classic serial approach applied in most works in the literature, although a parallel formulation makes more sense, as this will more closely follows the behavior of real ants.

A secondary objective of this thesis deals with presenting useful schemes which developers of the ACO algorithm can exploit for various situations. Besides a few generic and/or common schemes, of concern is the current state of the community in moving towards the time-period dimension of many classic problems to better approximate practical applications. As such, schemes for ACO that exploits the time-period properties are examined and encouraging running time performance for the MPMKP, which is a single time dimension extension of MKP, are obtained.

The current development opens avenue for future works, such as the further development of MDF, in the context of improving the collaboration between the existing engines, as well as introducing new algorithms. As well, more schemes can be formulated to cater for different classes of problems. Another potential is the use of ACF to hybridize with other engines in MDF to solve for other classic and/or industrial problems not examined previously. Furthermore, it is also interesting to explore a complete parallel implementation which was not examined.

REFERENCES

[Aboudi and Jornsten, 1994] R. Aboudi and K. Jornsten. Tabu search for general zero-one integer programs using pivot and complement heuristic, *ORSA Journal on Computing, 6, 82-93*, 1994

[Agarwal et al., 1989] Y. Agarwal, K. Mathur, and H.M. Salkin. Set Partitioning Approach to Vehicle Routing, *Networks 7, 731*, 1989

[Ahuja et al., 2003] R. K. Ahuja, K. C. Jha, J. B. Orlin, D. Sharma. Very Large-Scale Neighborhood Search for the Quadratic Assignment Problem, *MIT Working paper*, 2003

[Adleman, 1983] L. Adleman. On breaking the iterated Merkle-Hellman public-key cryptosystem, *Proceedings of the 15th ACM Symposium on the Theory of Computing, 402--412. 15,* 1983

[Applegate et al., 1995] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding Cuts in the TSP, *DIMACS Technical Report 95-05*, 1995.

[Araque et al., 1994] J.R. Araque, G. Kudva, T.L. Morin, and J.F. Pekny. A Branch-and-Cut Algorithm for Vehicle Routing Problems, *Annals of Operations Research 50, 37*, 1994

[Augerat et al., 1995] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberan, D. Naddef, and G. Rinaldi. Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem, *Research Report 949-M, Universite Joseph Fourier, Grenoble, France,* 1995

[Bahn et al., 1994] O. Bahn, O. du Merle, J.L. Goffin, and J.P. Vial. Cutting Plane Method from Analytic Centers for Stochastic Programming, *Technical Report, Dept. of Management Studies, University of Geneva, 1211 Geneva, Switzerland, Dec*, 1992, revised 1994

[Balas and Martin, 1980] E. Balas and C. H. Martin. Pivot and complement--A heuristic for 0-1 programming, *Management Science, 26, 86-96*, 1980

[Baldacci et al., 1999] R. Baldacci, A. Mingozzi, and E. Hadjiconstantinou. Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a two-commodity network flow formulation, *Technical Report 16, Department of Mathematics, University of Bologna*, 1999

[Balinski and Quandt, 1964] M.L. Balinski, and R.E. Quandt. On an Integer Program for a Delivery Problem, *Operations Research 12 (1964), 300*, 1964

[Bellman, 1957] R. Bellman. Dynamic Programming, *Princeton Univ. Press, Princeton, N.J.*, 1957

[Bent and Hentenryck, 2001] R. Bent and P. Van Hentenryck. A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows, *Technical Report CS-01-06, Department of Computer Science, Brown University*, 2001

[Bentley, 1980] J.L. Bentley. Multidimensional divide-and-conquer, *Communications of the ACM, 23, 214-229*, 1980

[Biggs et al., 1976] N.L.Biggs, E.K.Lloyd, and R.J. Wilson. Graph Theory 1736-1936, *Clarendon Press, Oxford*, 1976

[Blum and Dorigo, 2001] C. Blum, A. Roli, and M. Dorigo. HC-ACO: The hyper-cube framework for ant colony optimization, *Proceedings of the Metaheuristics International Conference, MIC 2001, Porto, Portugal, (2) 399-403*, 2001

[Bonabeau et al., 1997] E.W. Bonabeau, G. Theraulaz, J-L. Deneubourg, S. Aron, and S. Camazine. SelfOrganization in Social Insects, *Trends in Ecology and Evolution 12:188-193*, 1997

[Bonabeau, 1999] E. Bonabeau. Biological and economic networks, *Making Sense Of Networks, 4 MAY*, 1999

[Bonabeau and Theraulaz, 2000] E. Bonabeau, and G. Theraulaz. Swarm smarts, *Scientific American, pp. 72-79, March*, 2000

[Bullnheimer et al., 1997] B. Bullnheimer, R. F. Hartl, C. Strauss. Applying the Ant System to the Vehicle Routing Problem, *Proceedings of the 2nd Metaheuristics International Conference (MIC-97), Sophia-Antipolis, France,* 1997

[Braysy, 2001] O. Braysy. A Reactive Variable Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows, *Working Paper, University of Vaasa, Finland*, 2001

[Brickell, 1984] E. Brickell. Solving low-density knapsack, *Proc Crypto 83, pp 25-37*, 1984

[Chakradhar and Raghunathan, 1997] S. T. Chakradhar and A. Raghunathan. Bottleneck elimination algorithm for dynamic compaction and test cycles reduction, *IEEE Transactions on Computer-Aided Design, October*, 1997

[Campbell et al., 1998] A. Campbell, L. Clarke, A. J. Kleywegt, and M. W. P. Savelsbergh. The Inventory Routing Problem, in *T.G. Crainic, and G. Laporte, (eds), Fleet Management and Logistics, Kluwer Academic Publishers, pp. 95-113*, 1998

[Carter et al., 1996] M. W. Carter, J. M. Farvolden, G. Laporte, J. Xu, Solving an Integrated Logistics Problem Arising in Grocery Distribution, *INFOR, Vol 34:4, pp. 290-306*, 1996

[Chan et al., 1998] L. M. Chan, A. Federgruen and D. Simchi-Levi. Probabilistic Analysis and Practical Algorithms for Inventory-Routing Models, *Ops Res, Vol. 46:1. pp. 96-106*, 1998

[Chiang and Russell, 1997] W. Chiang and R. A. Russell, A Reactive Tabu Search Metaheuristic for Vehicle Routing Problem with Time Windows, *INFORMS Journal on Computing, Vol 8, No 4*, 1997

[Christofides and Eilon, 1969] N. Christofides and S. Eilon. An Algorithm for the Vehicle Dispatching Problem, *Operational Research Quarterly 20, 309*, 1969

[Christofides et al., 1981] N. Christofides, A. Mingozzi, and P. Toth. Exact Algorithms for Solving the Vehicle Routing Problem Based on Spanning Trees and Shortest Path Relaxations, *Mathematical Programming 20, 255*, 1981

[Colorni et al., 1993] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop schedul-ing. *Belg. J. Oper. Res., Stat. and Comput. Sci. 34, 39-53*, 1993

[Cook and Rich, 1999] W. Cook, J. L. Rich. A parallel cutting-plane algorithm for the vehicle routing problem with time windows, *Department of Computational and Applied Mathematics Technical Report TR99-04, Rice University*, 1999

[Cordeau et al., 2000] J. F. Cordeau, G. Laporte, and A. Mercier. A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows, *Centre for Research on Transportation, Montreal, Canada*, 2000

[Costa and Hertz, 1997] D. Costa and A. Hertz. Ants Can Colour Graphs, *Journal of the Operational Research Society, 48, 295-305*, 1997

[Crowder and Padberg, 1980] H. Crowder and M. Padberg. Solving Large Scale Symmetric Traveling Salesman Problems to Optimality, *Management Science 26, 495*, 1980

[Cullen et al., 1981] F.H. Cullen, J.J. Jarvis, and H.D. Ratliff. Set Partitioning Based Heuristic for Interactive Routing, *Networks 11, 125*, 1981

[Dammeyer and Voss, 1993] F. Dammeyer and S. Voss. Dynamic tabu list management using reverse elimination method, *Annals of Operations Research, 41, 31-46*, 1993

[Darwin, 1979] C. Darwin. Origin of Species, *Avenel Books, Crown Publishers*, 1979

[Dantzig et al., 1954] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large scale traveling salesman problem, *Operations Research, 2:393-410*, 1954

[Dantzig and Ramser, 1959] G.B. Dantzig and J.H. Ramser. The Truck Dispatching Problem, *Management Science 6, 80*, 1959

[Di Caro and Dorigo, 1998] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks, *Journal of Artificial Intelligence Research, 9:317--365*, 1998

[Distante and Piuri, 1989] F. Distante and F. Piuri. Hill-climbing heuristics for optimal hardware dimensioning and software allocation in fault-tolerant distributed systems, *IEEE transactions on Reliability*, *vol. 38:1, ISSN: 0018-9529, pp. 28-39*, 1989

[Dorigo and Di Caro, 1999] M. Dorigo, G. Di Caro. Ant Colony Optimization: A New Meta-Heuristic, *Proc. 1999 Congress on Evolutionary Computation, July 6-9, 1999, pp. 1470-1477*, 1999

[Dorigo and Gambardella, 1997] M. Dorigo and L.M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Trans on Evolutionary Computation, Vol. 1, No. 1*, 1997

[Dorigo et al., 1991] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: An Autocatalytic Optimizing Process. *Technical Report No. 91-016 Revised,* Politecnico di Milano, *Italy*, 1991

[Dorigo et al., 1996] 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, 26(1):29-41*, 1996

[Dorn et al., 1994] J. Dorn, M. Girsch, G. Skele, and W. Slany. Comparison of Iterative Improvement Techniques for Schedule Optimization, *Proceedings of the 13th UK Planning Special Interest Group, Glasgow (also available as CD-TR 94-61)*, 1994

[Dorigo et al., 1999] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant Algorithms for Discrete Optimization, Artificial Life, *5(2):137-172. Also available as Tech.Rep.IRIDIA/98-10, Université Libre de Bruxelles, Belgium*, 1999

[Elhedhi and Goffin, 2001] S. Elhedhli and J.L. Goffin. The Integration of an Interior-Point Cutting-Plane Method within a Branch-and-Price Algorithm, http://www.crt.umontreal.ca/~jlg/, *March*, 2001.

[Fidanova[1], 2002] S. Fidanova. ACO Algorithm for MKP Using Different Heuristic Information, *5th Int Conference of Numerical Methods and Applications,Lecture Notes in Computer Science, Springer-Verlag, Germany , 434-440*, 2002

[Fidanova[2], 2002] S. Fidanova. Evolutionary Algorithm for Multiple Knapsack Problem, In Proceedings of PPSN-VII, *Seventh International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany*, 2002.

[Fink et al., 1998] A. Fink, S. Voß, D. Woodruff. Tutorial Building Reusable Software Components for Heuristic Search, *INFORMS/CORS conference in Montreal, April*, 1998

[Finke et al., 1984] G. Finke, A. Claus, and E. Gunn. A two-commodity network flow approach to the traveling salesman problem, *Congress Numerantium 41, 167*, 1984

[Fisher, 1988] M.L. Fisher. Optimal Solution of Vehicle Routing Problems Using Minimum k-Trees, *Operations Research 42, 141*, 1988

[Fisher and Jaikumar, 1981] M.L. Fisher and R. Jaikumar. A Generalized Assignment Heuristic for Solving the VRP, *Networks 11, 109*, 1981

[Fleischmann, 1985] B. Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks, *European J. Oper. Res. 21: 307-317*, 1985

[Freville and Plateau, 1997] A. Freville and G. Plateau. The 0-1 bidimensional knapsack problem: toward an efficient high-level primitive tool, *Journal of Heuristics, 2, 147-167*, 1997

[Foster and Ryan, 1976] B.A. Foster and D.M. Ryan. An Integer Programming Approach to the Vehicle Scheduling Problem, *Operational Research Quarterly 27, 367*, 1976

[Gambardella et al.[1], 1999] L.M. Gambardella, E. Taillard, G. Agazzi. MACS-VRPTW: A Multiple Colony System For Vehicle Routing Problems With Time Windows, *Technical Report IDSIA, IDSIA-06-99*, 1999

[Gambardella et al.[2], 1999] L.M. Gambardella, E.D. Taillard, and M. Dorigo. Ant Colonies for the QAP, *Journal of the Operational Research Society, 50:167--176*, 1999

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, *Freeman, San Francisco, CA*, 1979.

[Gaspero and Schaerf, 2000] L. Di Gaspero and A. Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms, *Res. Report UDMI/13/2000/RR*, 2000

[Gavish and Pirkul, 1985] B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint Zero-One knapsack problems to optimality, *Mathematical Programming, 31, 78-105*, 1985

[Gehring and Homberger, 2001] H. Gehring and J. Homberger. A Parallel Two-phase Metaheuristic for Routing Problems with Time Windows, *Asia-Pacific Journal of Operational Research, 18, 35-47,* (2001)

[Gillet and Miller, 1974] B.E. Gillet, L.R. Miller. A Heuristics Algorithm for the Vehicle Dispatch Problem, *Operations Research, 22:340-349*, 1974

[Glover and Laguna, 1997] F. Glover and M. Laguna. Tabu Search, *Readings, Kluwer Academic Publishers, Boston/Dorderecht/London*, 1997

[Goldberg, 1989] D.E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning, *Addison-Wesley, Reading, Mass*, 1989

[Grotschel and Holland, 1991] M. Grotschel, and O. Holland. Solution of large--scale symmetric travelling salesman problems, *Math. Programming 51: 141-202*, 1991

[Gu, 1992] J. Gu. Efficient Local Search for Very Large-Scale Satisfiability Problem, *SIGART Bulletin, 3, 8-12*, 1992

[Harder, 2001] R.Hearder, IBM OpenTS Homepate, http://opents.iharder.net, 2001

[Held and Karp, 1969] M. Held, and R.M. Karp. The Traveling Salesman Problem and Minimal Spanning Trees, *Operations Research 18, 1138*, 1969

[Hoffmeyer, 1994] J. Hoffmeyer. The swarming body, *Proc. 5th Congress of the International Association for Semiotic Studies, Berkeley*, 1994

[Homberger and Gehring, 1999] J. Homberger and H. Gehring. Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows, *INFOR, VOL. 37, 297-318*, 1999

[Jin and Reynolds, 2000] X. Jin, R. G. Reynolds. Using knowledge-based systems with hierarchical architectures to guide evolutionary search, *International Journal on Artificial Intelligence Tools, Vol. 9, No. 1, 27-44,* 2000

[Johnson, 1990] D.S. Johnson, Local Optimization and the Traveling Salesman Problem, Procs of the 17th Colloquium on Automata, Languages and Programming, 446-461, 1990

[Joslin and Clements, 1999] D. E. Joslin and D. P. Clements. Squeaky Wheel Optimization, *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1999

[Junger and Stormer, 1995] M. Junger and P. Stormer. Solving large-scale traveling salesman problems with parallel Branch-and-Cut, *Technical Report No. 95.191*, 1995

[Kirkpatrick et al., 1983] S. Kirkpatrick, C. D. Gerlatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing, *Science 220, 671-680*, 1983.

[Knuth, 1968] D.E. Knuth. The art of programming, *Vol. 1, Addison-Wesley*, 1968

[Knuth, 1973] D.E. Knuth. The art of programming, *Vol. 3, Addison-Wesley*, 1973

[Larsen, 1999] J. Larsen. Vehicle Routing with Time Window – Finding optimal solutions efficiently, *DORSnyt (engl.), no. 116, Sep 15*, 1999

[Lau et al., 2000] H. C. Lau, A. Lim, and Q. Z. Liu. Solving a Supply Chain Optimization Problem Collaboratively. *Proc. 17th National Conf. on Artificial Intelligence, 780-785*, 2000

[Lau et al.^{1}, 2002] H. C. Lau, H. Ono, and Q. Z. Liu. Integrating Local Search and Network Flow to Solve the Inventory Routing Problem. *Proc. 19th National Conf. on Artificial Intelligence, 9-14,* 2002

[Lau et al.^{2}, 2002] H.C. Lau, S.C. Lua, and Y. Y. Song. The Multi-Period Multi-Dimensional Knapsack Problem, *Manuscript, TLI-AP*, 2002

[Lau et al.^{1}, 2003] H. C. Lau, W. C. Wan and X. Jia, A "Generic Object-Oriented Tabu Search Framework", *Metaheuristics International Conference*, 2003

[Lau et al.^{2}, 2003] H. C. Lau, M.K. Lim, W. C. Wan, H. Wang and X. Wu. Solving Multi-Objective Multi-Constrained Optimization Problems using Hybrid Ants System and Tabu Search, *Metaheuristics International Conference*, 2003

[Lawler et al., 1985] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, *Wiley, New York*, 1985

[Leguizamon and Michalewicz, 1999] G. Leguizamon and Z. Michalewicz. A New version of Ant System for Subset Problem, *Proceedings of the Congress of Evolutionary Computation, pp. 1459-1464*, 1999

[Li and Lim, 2001] H. Li and A. Lim. A Metaheuristic for the Pickup and Delivery Problem with Time Windows. *13th IEEE Int'l Conf on Tools with Artificial Intelligence*, 2001

[Lin and Kernighan, 1973] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem, *Oper. Res. 21: 498-516*, 1973

[Loulou and Michaelides, 1979] R. Loulou and E. Michaelides. New greedy-like heuristics for the multidimensional 0-1 knapsack problem, *Operations Research, 27, 1101-1114*, 1979

[Mangano, 1995] S. Mangano. Man Machine Interfaces, *Computer Design, May*, 1995

[Maniezzo et al., 1994] V. Maniezzo, A.Colorni, and M.Dorigo. The ant system applied to the quadratic assignment problem, *Tech. Rep. IRIDIA/94-28*, *Université Libre de Bruxelles, Belgium*, 1994

[Mester, 2002] D. Mester. An Evolutionary Strategies Algorithm for Large Scale Vehicle Routing Problem with Capacitate and Time Windows Restrictions, *Working Paper, Institute of Evolution, University of Haifa, Israel* (2002)

[Mester and Braysy, 2002] D. Mester and O. Braysy. Guided Evolution Strategies for Large Scale Vehicle Routing Problem with Time Windows, *Working Paper, Institute of Evolution, University of Haifa, Israel* (2002)

[Metropolis et al. 1958] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines, *J. Chem. Phys. 21, 1087- 1092*, 1958.

[Naddef and Rinaldi, 1991] D. Naddef, and G. Rinaldi. The Symmetric Traveling Salesman Polytope and its Graphical Relaxation: Composition of Valid Inequalities, *Mathematical Programming 51, 359*, 1991

[Naddef and Rinaldi, 1993] D. Naddef, and G. Rinaldi. The Graphical Relaxation: A New Framework for the Symmetric Traveling Salesman Polytope, *Mathematical Programming 58, 53*, 1993

[Naddef and Rinaldi, 2000] D. Naddef and G. Rinaldi. Branch and Cut, *P. Toth and D. Vigo, eds., Vehicle Routing, SIAM*, 2000

[Narendra and Fukunaga, 1977] P. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection, *IEEE Trans on Computers, 26:917-922*, 1977

[Nemhauser and Wolsey, 1988] G.L. Nemhauser and L.A. Wolsey. Integer and Combinatorial Optimization, *John Wiley, Chichester, UK*, 1988

[Padberg and Grotschel, 1995] M.W. Padberg and M. Grotschel. Polyhedral computations, *The Traveling Salesman Problem (E. L. Lawler et al., eds), Wiley, Chichester, pp. 307-360*, 1995

[Padberg and Rinaldi, 1987] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut, *Oper. Res. Lett. 6(1): 1-7*, 1987

[Padberg and Rinaldi, 1990] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope, *Math. Programming* 47: 219257, 1990

[Padberg and Rinaldi, 1991] M. Padberg, and G. Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Traveling Salesman Problems, *SIAM Review 33, 60*, 1991

[Papadimitriou, 1994] C.H. Papadimitriou. Computational Complexity, *Addison-Wesley, Reading, MA*, 1994

[Parsopoulos and Vrahatis, 2002] Konstantinos E. Parsopoulos, Michael N. Vrahatis. Particle Swarm Optimization Method for Constrained Optimization Problems, *Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies, pp. 214-220, IOS Press (Frontiers in Artificial Intelligence and Applications series, Vol. 76)*, 2002

[Pincus, 1970] M. Pincus, A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems, *Oper. Res. 18, 1225-1228*, 1970.

[Pirkul, 1987] H. Pirkul. A heuristic solution procedure for the multiconstraint Zero-One knapsack problem, *Naval Research Logistics, 34, 161-172*, 1987

[Ralphs, 2003] T.K. Ralphs. Parallel Branch and Cut for Vehicle Routing, *Parallel Computing 29, 607*, 2003

[Ralphs et al., 2003] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, , and L.E. Trotter Jr. On the Capacitated Vehicle Routing Problem, *Mathematical Programming Series B 94, 343*, 2003

[Rochat and Taillard, 1995] Y. Rochat, and E.D. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing, *Journal of Heuristics 1, 147-167*, 1995

[Rousseau et al., 1999] L.M. Rousseau, M. Gendreau and G. Pesant. Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows, *Journal of Heuristics*, 1999

[Schoonderwoerd et al., 1997] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks, *Adaptive Behavior, vol. 5, no. 2*, 1996

[Schrijver] A. Schrijver. On the history of combinatorial optimization (till 1960), http://www.cwi.nl/~lex/files/histco.ps

[Schulze and Fahle, 1999] J. Schulze, and T. Fahle. A Parallel Algorithm for the Vehicle Routing Problem with Time Window Constraints, *Combinatorial Optimization: Recent Advances in Theory and Praxis, J.E. Beasley, Y.M. Sharaha (eds.), Baltzer, Special Volumn of Annals of Operations Research, 86, 1999, 585-607*, 1999

[Shamir, 1982] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem, *in Proc. of 23rd FOCS, pages 145--152. IEEE*, 1982

[Shih, 1979] W. Shih. A branch and bound method for the multiconstraint Zero-One knapsack problem. *Journal of the Operational Research Society, 30, 369-378*, 1979

[Stutzle and Dorigo, 1999] T. Stutzle and M. Dorigo. ACO Algorithms for the Traveling Salesman Problem, In *Evolutionary Algorithms in Engineering and Computer Science, pp. 163-183, Wiley*, 1999

[Solomon, 1987] M. M. Solomon. Algorithms for Vehicle Routing and Scheduling Problem with Time Window Constraints, *Operation Research Vol. 35, pp. 254 – 265*, 1987

[Taillard et al., 1997] E. Taillard, P. Badeau, M. Gendreau, F. Geurtin, and J. Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows, *Transportation Science, 31, 170-186*, 1997

[Tomov, 1994] N. Tomov. Hill-climbing heuristics for solving constraint satisfaction problems, *4th year project report, Department of Artificial Intelligence, University of Edinburgh*, 1994

[Toth and Vigo, 2002] P. Toth, and D. Vigo. The Vehicle Routing Problem, *SIAM Monographs on Discrete Mathematics and Applications*, 2002

[Wan, 2002] W.C. Wan. A Guided Generic Tabu Search Framework for Combinatorial Optimization Problems, *B.Eng thesis, National University of Singapore*, 2002

[Ward, 1998] M. Ward. There’s an ant in my phone, *New Scientist, 24* January, 1998

[Toyoda, 1975] Y. Toyoda. A simplified algorithm for obtaining approximate solutions for zero-one programming problems, *Management Science, 21, 1417-1427*, 1975

[Michel and Van Hentenryck, 1999] L. Michel and P. Van Hentenryck. Localizer: A modeling language for local search, *INFORMS Journal of Computing, 11(1):1-14*, 1999

[Yufik and Sheridan, 2002] Y. M. Yufik and T. B. Sheridan. Swiss Army Knife and Ockham’s Razor: Modeling and Facilitating Operator’s Comprehension in Complex Dynamic Tasks, *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 32, No. 2, March,* 2002

[Zhang, 1993] W. Zhang. Truncated branch-and-bound: A case study on the asymmetric tsp., *Working Note of AAAI 1993, Spring Symposium: AI and NP-Hard Problems, pp 160-166, Stanford, CA, March 22-25,* 1993