MULTILEVEL ANT COLONY OPTIMIZATION TO SOLVE CONSTRAINED FOREST TRANSPORTATION PLANNING PROBLEMS

Pengpeng Lin
University of Kentucky, pli222@g.uky.edu

MULTILEVEL ANT COLONY OPTIMIZATION TO SOLVE CONSTRAINED FOREST TRANSPORTATION PLANNING PROBLEMS

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College of Engineering at the University of Kentucky

By

Pengpeng Lin

Lexington, Kentucky

Director: Jun Zhang, Ph.D., Professor of Computer Science

Lexington, Kentucky

2015

ABSTRACT OF DISSERTATION

MULTILEVEL ANT COLONY OPTIMIZATION TO SOLVE CONSTRAINED FOREST TRANSPORTATION PLANNING PROBLEMS

In this dissertation, we focus on solving forest transportation planning related problems, including constraints that consider negative environmental impacts and multi-objective optimizations that provide forest managers and road planers alternatives for making informed decisions. Along this line of study, several multilevel techniques and mataheuristic algorithms have been developed and investigated. The forest transportation planning problem is a fixed-charge problem and known to be NP-hard. The general idea of utilizing multilevel approach is to solve the original problem of which the computational cost maybe prohibitive by using a set of increasingly smaller problems of which the computational cost is cheaper.

The multilevel techniques are devised consisting of two parts. The first part is to recursively apply a graph coarsening heuristic to the original problem to produce a set of coarser level problems of which the sizes in terms of number of problem components such as edges and nodes are in decreasing order. The second part is to solve the set of the coarser level problems including the original problem bottom up, starting with the coarsest level. We propose that if coarser level problems inherit important properties (such as attribute value distribution) from their ancestor during the coarsening process, they can be treated as smaller versions of the original problem. Based on this hypothesis, the multilevel techniques use solutions obtained for the coarser level problems to solve the finer level problems.

Mainly, we develop multilevel techniques to address three problems, namely a constrained fixed-charge problem, parameter configuration problem, and a multi-objective transportation optimization problem in this study. The performance of the multilevel techniques is compared with other commonly used approaches. The statistical analyses on the experimental results indicate that the multilevel approach can reduce computing time significantly without sacrificing the solution quality.

KEYWORDS: Metaheuristic, Approximation Algorithms, Parameter Turning, Graph Coarsening, Multilevel Approaches.

Pengpeng Lin

Thursday 11th June, 2015

MULTILEVEL ANT COLONY OPTIMIZATION TO SOLVE CONSTRAINED FOREST TRANSPORTATION PLANNING PROBLEMS

By

Pengpeng Lin

Jun Zhang, Ph.D.

Director of Dissertation

Miroslaw Truszczynski, Ph.D.

Director of Graduate Studies

Thursday 11th June, 2015

Date

ACKNOWLEDGMENTS

The PhD study at the University of Kentucky has been an important and memorable time in my life. Without the guidance from my advisor and other committee members, helps from friends, and supports from my family, I would never have been able to finish my dissertation.

First of all, I would like to express my deepest appreciation to my academic advisor, Dr. Jun Zhang, for his excellent guidance, patience, encouragement, and caring. It is Dr. Zhang who trained me for becoming a researcher. Because of his broad knowledge and keen professional insight, I can start my research in a correct and clear direction and finally form the contents of my dissertation.

Next, I would like to thank the other faculty members of my Advisory Committee: Dr. Grzegorz Wasilkowski (Department of Computer Science), Dr. Zongming Fei (Department of Computer Science), Dr. Marco Contreras (Department of Forest Management) for serving as my Ph.D. committee members.

Also, I would like to thank Dr. Jun Zhang, Dr. Nill Moore, Dr. Marco Contreras, and Dr. Huanjing Wang for their kind assistance with writing recommendation letters and helping me in my job search.

Thanks also go to all members in the Laboratory for High Performance Scientific Computing & Computer Simulation and Laboratory for Computational Medical Imaging & Data Analysis during my study: Dr. Yin Wang, Dr. Xuwei Liang, Dr. Changjiang Zhang, Dr. Dianwei Han, Dr. Ning Cao, Dr. Zhenhuan Zhou, Dr. Yongbin Ge, Dr. Lingjuan Li, Dr. Yingjin Lu, Dr. Jue Wu, Dr. Nirmal Thapa, Dr. Ruxin Dar, Dr. Lian Liu, Mr. Xiwei Wang. I want to thank them for their helpful suggestions and creating a friendly working environment.

At last, I would like to thank my family. I thank my father and mother for giving me life and endless love. I thank my wife for her wisdom, support and encouragement.

Table of Contents

List of Tables

List of Figures

1 Introduction

Metaheuristic algorithms are defined as a set of higher-level procedures that provide sufficiently good solutions to optimization problems when limited computational resources are available [8]. In comparison to exact algorithms that exhaustively search for every possible option, metaheuristics do not guarantee the globally optimal solutions. However, when the exact algorithms fail to tackle large-scale problems because of the limited computational resources, metaheuristics can usually provide high-quality solutions with much less efforts.

A major strain of the metaheuristic in solving optimization problems arises from the research on swarm intelligences (SI), which can be described as the collective behavior that emerges from a group of social insects, of which their team works involve both self-organizations and interactive-coordinations in a colony [13]. Studies of this collective behavior include a variety of nature based activities such as nest building, foraging, item sorting, swarming, flocking, herding, schooling, and so on. Based on these studies, many applications of SI have been developed for conventional optimization problems where different SI designs are employed in library materials acquisitions, communications, medical dataset classification, dynamic control, transportation planning, heating system planning, moving objects tracking, and predictions.

This dissertation mainly focuses on studying and developing a multilevel metaheuristic paradigm that aims at solving large-scale optimization problems modeled with underlying networks. In particular, we target a constrained forest transportation planning problem (CFTPP). However, the techniques presented can be easily extended to other problem domains. For the purpose of multilevel computing, graph coarsening heuristics and a SI framework, namely ant colony optimization, are used to provide hierarchical computations, where larger problems are solved in a back-track fashion by their smaller versions. The following sections introduce the related techniques and preliminary knowledges.

1.1 Ant Colony Optimization (ACO) Metaheuristic

1.1.1 Background of ACO

ACO was developed in the mid 1990s by Marco Dorigo [25, 26]. The algorithm was inspired by ant foraging behavior. When searching for food, ants walking to and from a food source deposit a substance called pheromone on the ground. Other ants can perceive the presence of the pheromone and tend to follow paths where pheromone concentration is higher. Through this mechanism, ants are able to transport food to their nest in a remarkably effective way [23].

Several successful applications of ACO to a wide range of different optimization problems, especially for the NP-hard problems, have been developed. When the best known algorithms that promise to obtain the optimal solutions become practically infeasible for solving large size problems, ACO based algorithms, on the other hand, can quickly find high quality solutions. As an example, the most notable application of ACO is the traveling salesmen problem (TSP), where a salesman has to travel through a list of cities in such a way that the expenses are minimized [73]. To solve this problem, ACO deploys a set of artificial ants to search for the shortest possible route that visits each city exactly once. Guided by the artificial pheromone values, the ants construct solutions simultaneously according to the transition probability:

$$P_{ij}^{k} = \begin{cases} \frac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{n \in N} [\tau_{in}]^{\alpha} \times [\eta_{in}]^{\beta}} & \text{if } (i,j) \text{ is a solution component} \\ 0 & \text{Otherwise} \end{cases} \tag{1.1}$$

where P_{ij}^{k} is the probability for ant k to move from the city i to the city j, $n \in N$ represents the set of adjacent cities, τ_{ij} is the pheromone intensity on path (i, j), η_{ij} is the visibility that is the inverse to the distance between the two cities, α and β are parameters that control the relative importance of pheromone intensity versus visibility. The value of τ is updated iteratively using the following formula:

$$\tau_{ij} \leftarrow \rho \times \tau_{ij} + \Delta\tau_{ij},$$

where $0 < \rho < 1$ is the pheromone persistence rate and $\Delta\tau_{ij}$ is the amount of pheromone to be added to path (i, j). The value of $\Delta\tau_{ij}$ is usually set to a small positive number to increase the pheromone amount on the path (i, j) if it is selected as a solution component. On the other hand, $\Delta\tau_{ij}$ is set to zero, indicating no pheromone increase for the non-solution component. Through this update mechanism, the artificial ants are directed towards better solutions by the iteratively updated pheromone values.

1.1.2 ACO Algorithms

There have been a number of ACO algorithms proposed in the literature. They share the same idea of ant foraging behaviors, and mainly differ in objective functions and how the pheromone is updated. Below, we present the Ant System, Ant Colony System, and Max-Min Ant System that have been used as the programming templates for developing customized ACO algorithms in this thesis.

Ant System (AS)

The original ACO algorithm is the Ant System [22], where the pheromone values are updated at every iteration by all the m ants that have built a solution. The pheromone τ_{ij}, associated with the path connecting cities i and j, is updated as follows:

$$\tau_{ij} \leftarrow \rho \times \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \tag{1.2}$$

where $\Delta\tau_{ij}^{k}$ is the increased pheromone amount added on the path (i, j) by ant k:

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{if edge } (i, j) \text{ is used,} \\ 0 & \text{otherwise,} \end{cases} \tag{1.3}$$

where Q is a parameter, L_k is the length of the route constructed by ant k. In the construction of a solution, the above transition probability formula is used to select edges that form the solution (solution components).

The experiments conducted by Dorigo et al [22] for testing AS showed that the algorithm was robust in solving TSPs, and the results also indicated that within a range of parameter value settings (i.e., for the parameters α and β), the algorithm always found very good solutions for all the tested problems.

Ant Colony System (ACS)

Ant Colony System is a variant of the Ant System. In addition to the pheromone update performed at the end of the solution construction in AS, ACS introduces a local pheromone update mechanism [21, 35, 67], which is performed by all the ants after each construction step and applied to only to the last edge traveled:

$$\tau_{ij} = (1 - \varphi) \times \tau_{ij} + \varphi \times \tau_0, \tag{1.4}$$

where $0 < \varphi \leq 1$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

The local pheromone update is used to diversify the subsequent searches by decreasing the pheromone concentration on the traveled routes. Other ants are given more chances to choose a different set of solution components. As a consequence, ants are unlikely to converge to a common path. This is a desirable property given that if ants explore different paths then there is a higher probability that one of them will find an improving solution than that in the case that they all converge to the same tour [20].

Moreover, another difference between ACS and AS is the transition probability used by ants during the solution construction process. In ACS, the probability for an ant to move from city i to city j depends on a random parameter q, which is uniformly distributed over $[0, 1]$, and a parameter q_0. If $q \leq q_0$, then

$$j = \arg\max_{n \in N} \{\tau_{in} \times \eta_{in}\} \tag{1.5}$$

where N is the set of the adjacent cities. Otherwise, formula 1.1 is used.

Max-Min Ant System (MMAS)

Max-Min Ant System is another variant of AS [79]. In MMAS, only the best ant (the ant that found the best solution) updates the pheromone and the value of the pheromone is bounded:

$$\tau_{ij} \leftarrow [(1-\rho) \times \tau_{ij} + \Delta\tau_{ij}^{best}]_{\tau_{min}}^{\tau_{max}}, \tag{1.6}$$

where τ_{max} and τ_{min} are the upper and lower bounds of the pheromone value, and the operator $[x]_b^a$ is defined as:

$$[x]_b^a = \begin{cases} a & \text{if } x > a, \\ b & \text{if } x < b, \\ x & \text{otherwise,} \end{cases} \tag{1.7}$$

and

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & \text{if } (i,j) \text{ belongs to the best tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{1.8}$$

where L_{best} is the length of either the best route found in the current iteration or the best solution found since the beginning of the algorithm or a combination of both [24].

The improvement of MMAS, compared to AS, is that setting limits for amount of pheromone helps avoid leading ants to a premature stagnation, which is the situation where all ants follow the same path and construct the same solution over and over again such that better solutions cannot be found anymore [22]. It has been proven in [79] that the maximum possible pheromone is asymptotically bound by

$$\frac{1}{1-\rho} \times \frac{1}{f(s^{opt})} \tag{1.9}$$

where $f(s^{opt})$ is the optimal solution value. The lower bound can be determined with a value for P_{best} that is the probability of choosing the best solution component at a choice point. However, the upper and lower bounds are typically obtained empirically and tuned for specific problems [74].

In this dissertation, the above described ant colony optimization algorithms have been used as testing algorithms or served as programming templates in Chapter 2, 3, 4, 5 and 6.

1.2 Graph Coarsening Heuristics

The graph coarsening plays an important role in developing the multilevel meta-heuristic framework. It can be interpreted as a process of aggregation of graph nodes to define the nodes of the next coarser graph. Specifically, given a graph $G_0 = (V_0, E_0)$, graph coarsening techniques construct a sequence of graphs $G_1, \ldots, G_l$, where $G_i = (V_i, E_i)$ is a coarse approximation of G_{i-1} such that a solution of a given problem for G_i can be "efficiently" extended to that for G_{i-1} and vice versa [83, 86]:

$$|G_0| > |G_1| > |G_2| > |G_3| > \ldots > |G_l|, \tag{1.10}$$

where the operator $|\cdot|$ denotes for the size of a graph, i.e., number of edges and nodes. Essentially, the process of producing a coarser level graph is mainly composed of selecting a set of edges from a finer level graph, then collapsing the selected edges (also known as edge contractions), and aggregating the end nodes.

1.2.1 Finding Matching Edges

The edges selected for contraction are the matching edges (formally called matching) of which no two edges are incident on the same node [52, 64]. There are many existing techniques for finding a maximum matching in a graph [52, 32, 12, 87]. In this thesis, we implement a matching algorithm (Algorithm 1), which is adopted and modified from a randomized algorithm proposed in [52] of which the complexity is proportional to the number of edges $O(|E|)$ [43].

In Algorithm 1, a maximum matching is found by visiting every node in a finer graph. When an unmatched node u is visited, the algorithm checks whether it has an adjacent node that has not been matched. If such a node v_i exists, edge(u, v_i) is included in the matching and nodes u and v_i are marked as matched. Otherwise, u remains unmatched and the algorithm continues to visit the next unvisited node.

If the node u is adjacent to more than one unmatched node, Algorithm 1 is designed to include the incident edge with the largest attribute weight into the matching. This design is particularly suitable for minimization problems, because edges with large attribute values are unlikely to be part of good solutions. In contrast, for the maximization problems, edges with small attribute weights should be considered for the contraction.

Algorithm 1 Finding Matching Edges Algorithm.

```
$Q :=$ all nodes except source and destination nodes;
$QE = \phi$;
begin
    while $Q$ has unvisited nodes do
        visit an unvisited node $u$ in $Q$
        if $u$ has been matched then
            continue to next while iteration
        end
        for adjcant unmatched nodes $v$ of $u$ do
            find $v_i \in v \mid$ wight$(u, v_i)$ is maximum
        end
        make vertices $u$ and $v_i$ as matched
        add edge$(u, v_i)$ to $QE$
    end
end
return($QE$)
```

1.2.2 Edge Contraction

After a matching is found from a finer graph, the selected edges are collapsed to generate a coarser graph (Algorithm 2). The end nodes of matching edges are aggregated to form coarser level nodes. Those nodes that are not incident on matching edges are directly copied to the coarser graph. Because two matching edges can not be incident to the same node, each node in the finer level graph is mapped to an unique node in the resulted coarser level graph.

Algorithm 2 Graph Coarsening Algorithm.

```
$QE$ := matching edges
begin
    for edge(u,v) $\in QE$ do
        $u\_v \Leftarrow$ aggregate $u$ and $v$
        if $u$, $v$ are both adjacent to a node $k$ then
            weight($u\_v$,$k$) := weight($u$,$k$) + weight($v$,$k$)
        end
    end
end
```

While matching edges are collapsed together in the coarsening process, edges that both connect to the aggregated nodes and are incident to the same node are jointed together to form coarser level edges. Naturally, the weights of two jointed edges are combined to give a new weight to the resulted coarser level edge. This approach is adopted from [52], in which adjacent nodes to a node u_1 in a graph is defined as

$$Adj(u_1) = (\{Map[x] | x \in Adj(v_1)\} \cup \{Map[x] | x \in Adj(v_2)\}) - \{u_1\}$$

where v_1 and v_2 are the finer level nodes aggregated to obtain the coarser level node u_1, and $Map[x]$ maps a finer level node to a coarser level node such that $Map[v_1] = Map[v_2] = u_1$. The combined attribute weight of an edge (u_1, u_2) is given by

$$w(u_1, u_2) = \sum_x \{w(v_1, x) | Map[x] = u_1\} + \sum_x \{w(v_2, x) | Map[x] = u_2\}.$$

An example of the edge contraction process is illustrated in Figure 1.1 where a finer graph G_i is coarsened to construct a coarser graph G_{i+1}. The matching edges (W_1, W_4, W_{10}) are encompassed with dashed ovals, and they are contracted to produce the coarser level graph. Nodes 1 and 4 are aggregated into the coarser node 1_4 and the weights W_2 and W_6 are combined to $W_2 + W_6$. In the same manner, nodes 3 and 2 are aggregated into the coarser node 3_2. The weight W_3 and the aggregated weight $W_2 + W_6$ are combined to $W_2 + W_6 + W_3$. Finally, nodes 5 and 7 are aggregated into the coarser node 5_7.

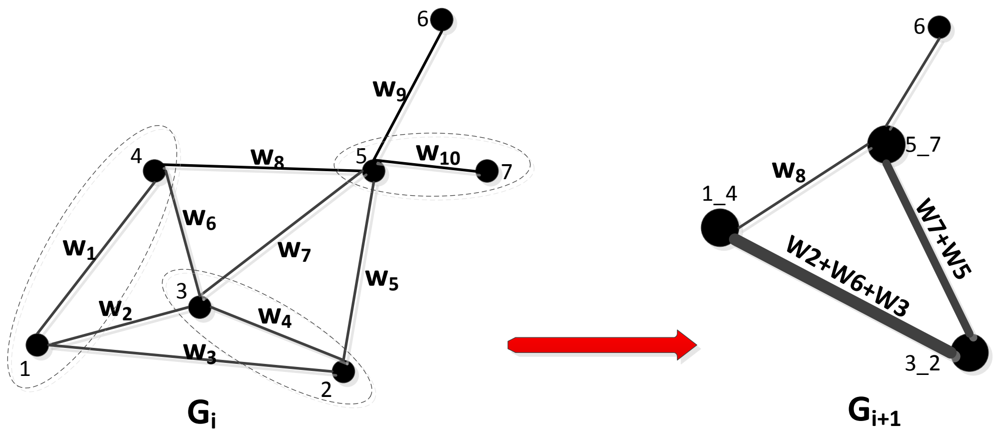


Figure 1.1: Edge contraction process.

1.3 Constrained Forest Transportation Planning Problem

The forest transportation planning problem (FTPP) is an economic conduct of forest harvesting operations that has long been the primary objective in the construction and maintenance of the forest transportation network [39]. Traditionally, the goal of FTPP has been to find the road network that minimizes both log hauling and road construction costs for timber management [16]. Modern FTPPs are driven not only by the financial aspects of timber management, but also by multiple uses of roads, such as recreation, and their social and environmental impacts on such things as water quality, wildlife, and fish habitats.

A great source of the environmental impacts comes from the forest road network due to the annually generated sediment [28, 30]. Research on the effects of forest roads has shown that traffic on forest roads can result in accelerated erosion and water quality impacts [48, 47, 49]. For example, the stream channels receive the highest amount of sediment during forest road construction activities due to removal of vegetation cover from road surface, cut-slope, fill-slope, and ditch areas. Sediment delivered from a road section to streams causes serious damages on water resources and aquatic life [3].

These environmental and social considerations and requirements introduce side constraint to the FTPP (CFTPP), which makes the problem more complex than the traditional cost minimization. In this study, we target a CFTPP that considers a fixed cost (i.e., road construction cost), a variable cost (log hauling) of timber transportation and a sediment constraint. The CFTPP is a special case of the fixed charge transportation problem (FCTP) and has been known as a NP-hard combinatorial optimization problem [58].

1.3.1 Objective Functions and Constraints of CFTPP

The CFTPP can be modeled as a network problem comprised of a set of nodes V and edges E representing road intersections and segments, respectively [17]. Each edge in the network is associated with three attributes: fixed cost (FC), variable cost (VC), and sediment amount (Sed). FC for an existing road segment is a one-time road construction cost and/or maintenance cost, VC includes hauling cost that is proportional to timber volume traffic, and Sed (tons/year) represents the amount of sediment expected to erode from the road segment caused by heavy traffic of log-trucks.

To formulate the CFTPP, let $S = \{s_1, \ldots, s_m\}$ be the set of timber sale locations and $D = \{d_1, \ldots, d_n\}$ the set of mill destinations, where $S, D \subset V$. Each timber sale $s_i \in S$ has a minimum volume of timber to be delivered at a given period to a designated mill $d_j \in D$ on a road network: $G = (S \cup D, \Omega)$, where $\Omega \subset E$ is the set of routes connecting all timber sale locations to the selected mills. For a route $R_{s_i,d_j} \in \Omega$ (the route from a timber sale s_i to a mill d_j), let $VC_{i,j}$ be the total variable cost, $FC_{i,j}$ the total fixed cost, and $Sed_{i,j}$ the total sediment amount. The objective function of the CFTPP is defined as:

$$Minimize: \sum_{R_{s_i,d_j} \in \Omega} VC_{i,j} \times Vol_{i,j} + FC_{i,j} \tag{1.11}$$

while the amount of sediment eroded from the entire road network is under a maximum allowable value Sed_{max}:

$$Constraint: \sum_{R_{s_i,d_j} \in \Omega} Sed_{i,j} \leq Sed_{max}. \tag{1.12}$$

1.3.2 Mixed Integer Programming Formulations for CFTPP

General approaches for solving FCTPs involve a mixed integer programming (MIP) formulation [17, 70, 76, 38]. The MIP is used in this work to measure performance of the proposed techniques and its formulations for the CFTPP is given as the follows:

$$Minimize: \sum_{ij \in E} [VC_{ij} \times (Vol_{ij} + Vol_{ji}) + FC_{ij} \times B_{ij} \tag{1.13}$$

Subject to:

$$\sum_{ij \in E} (Sed_{ij} \times B_{ij}) \leq allowable\_sed \tag{1.14}$$

$$Vol\_Sale_j + \sum_{ij \in L} Vol_{ij} - \sum_{ji \in L} Vol_{ji} = 0 \qquad \forall j \in S \tag{1.15}$$

$$\sum_{ij \in L} Vol_{ij} - \sum_{ji \in L} Vol_{ji} = 0 \qquad \forall j \in T \tag{1.16}$$

$$\sum_{ij \in L} Vol_{ij} - \sum_{j \in S} Vol\_Sale_j = 0 \qquad \forall j \in D \tag{1.17}$$

$$M \times B_{ij} - (Vol_{ij} + Vol_{ji}) \geq 0 \qquad \forall ij \in E \tag{1.18}$$

$$Vol_{ij}, Vol_{ij} \geq 0 \qquad \forall ij \in E \tag{1.19}$$

$$B_{ij} \in \{0, 1\} \qquad \forall ij \in E \tag{1.20}$$

Expression 1.13 specifies the objective function of the problem, where VC is the variable cost of wood volume transported over edge ij in either direction, Vol_{ij} is the wood volume transported over the edge from node i to node j, Vol_{ji} is the amount transported in the opposite direction (from node j to node i), FC is the fixed cost for edge ij in dollars, B_{ij} is a binary variable representing construction of edge ij (1 if the edge is built and 0 otherwise), and E indicates the total number of edges in the network. The first constraint (Eq. 1.14) is the sediment constraint that limits the maximum allowable sediment amount ($allowable\_sed$) in tons, where Sed_{ij} is amount of sediment from edge ij if the edge is built for traffic. The second, third, and fourth sets of constraints (Inequality. 1.15 to 1.17) ensure that all volume entering the network is channeled through the network to the destination nodes (mill location). The constraints in Eq. 1.15 apply to the set of origin nodes, S (timber sale locations), and ensure the sum of sale volume entering the network at node j, Vol_Sale_j, plus the sum of volume transported to j from other nodes, Vol_{ij}, equal the sum of volume transported from node j to connecting nodes, Vol_{ji}. L is the set of edges having node j as a from-or-to node. The constraints in Eq. 1.16 apply to the set of intermediate nodes, T (that are neither origins nor destinations), and ensure that the sum of volume entering node j, Vol_{ij}, equals the sum of volume leaving that node, Vol_{ji}. The constraints in Eq. 1.17 apply to the destination nodes, D (mill locations), and specify that the sum of the volume entering those nodes, $\sum_{ij\in L} Vol_{ij}$, equals the sum of the sale volume loaded onto the origin nodes, $\sum_{j\in S} Vol\_Sale_j$, thus ensuring all volume that enters the network is routed to the destination nodes. The fifth set of constraints (Eq. 1.18) represents the road-building constraint that makes sure that, if there is volume transported over edge (i, j) in either direction, the edge must be constructed and open for traffic; thus, the fixed cost and sediment amount are counted. M is a constant greater than or equal to the total amount of volume to be delivered to the mills. Lastly, the sixth and seventh sets of constraints (Eqs. 1.19 and 1.20) represent the non-negativity and binary value constraints, respectively.

1.4 Organization

The remaining Chapters of this dissertation are organized as follows:

- In Chapter 2, an ACO algorithm that is designed to search solutions in two stages and incorporates local search to refine the solution quality is presented to solve the CFTPP. An exhaustive search is conducted to look for the best parameter settings for the ACO in order to maximize its performance.

- In Chapter 3, a multilevel approach, namely MParamILS, is developed to automatically configure the ACO algorithm. The MParamILS combines a graph coarsening technique and the ParamILS to select high-quality parameter settings by refining a parameter combination domain from the coarsest level problem to the finest level problem.

- In Chapter 4, the idea of the multilevel approach is further studied and extended to develop a multilevel ACO for solving the CFTPP, which integrates the MParamILS to automatically configure the parameters and uses the information obtained from solving coarser level problems to help search for solutions for the finer level problems.

- In Chapter 5, a multi-objective ACO, that is designed by combining various algorithmic components in existing ACO algorithms, is developed to solve a bi-objective FTPP, in which the objectives are to minimize both transportation cost and negative environmental impact at the same time.

- In Chapter 6, a multilevel multi-objective ACO is developed to solve the bi-objective FTPP, which aims at improving the performance of the multi-objective ACO algorithm in terms of computing time and solution quality.

- In Chapter 7, I summarize contributions of this dissertation and outlook for possible future research work.

2 Applying Ant Colony Optimization for Solving Constrained Forest Transportation Planning Problems

2.1 Introduction

In this Chapter, we present an ACO algorithm to solve the CFTPP. It consists of three major routines: least-cost route finding process from all timber sales simultaneously; two stage search process developed to quickly find feasible (stage one) and high-quality (stage two) solutions; local search solution refinement to further improve solution quality. The ACO algorithm was applied to a medium scale, grid-shaped hypothetical FTPP with 500 road segments, 25 timber sale locations, and a single mill destination. Four cases with increasing levels of sediment constraints were considered, and an exhaustive parameter search process was conducted to select the best parameter combination for each case. To test the robustness of the ACO algorithm, we created 10 different problem instances with different timber sale locations and destination nodes for the same hypothetical network. Solutions were then compared with those obtained from the MIP formulations solved by CPLEX [18].

2.2 Proposed ACO Algorithm

2.2.1 Ant Traveling Routine

In the ACO algorithm, artificial ant colonies are placed at the entry nodes (timber sale locations). Ants from the affiliated colonies move sequentially through adjacent nodes until the destination node (mill) is reached. Let S denote the set of timber selling locations, D the set of mills, C the set of ant colonies. Then, at each algorithm iteration, each colony $C_i \in C$ sends out one ant $a_i \in C_i$ to search for the best route to the designated mill location $D_i \in D$. After all colonies have found the best routes connecting each timber sale to the selected destination node, one iteration is completed.

The ants in the colonies are searching routes in the order that is assigned to each colony randomly at the beginning of the algorithm iteration. This sequence order is randomly shuffled for every new iteration to ensure that every colony in the network has a fair chance to be the first one to construct solution.

During the route search process, if ant a_i from C_i moves to a node N_i that belongs to the route that was discovered by ant a_j of a colony that precedes C_i in the search sequence and shares the same destination mill, the ant a_i will stop the search and join the solution route found by a_j. Similarly, if an entry node has been visited and is part of the solution routes obtained by other colonies with the same destination, the colony at the entry will not initiate the search process.

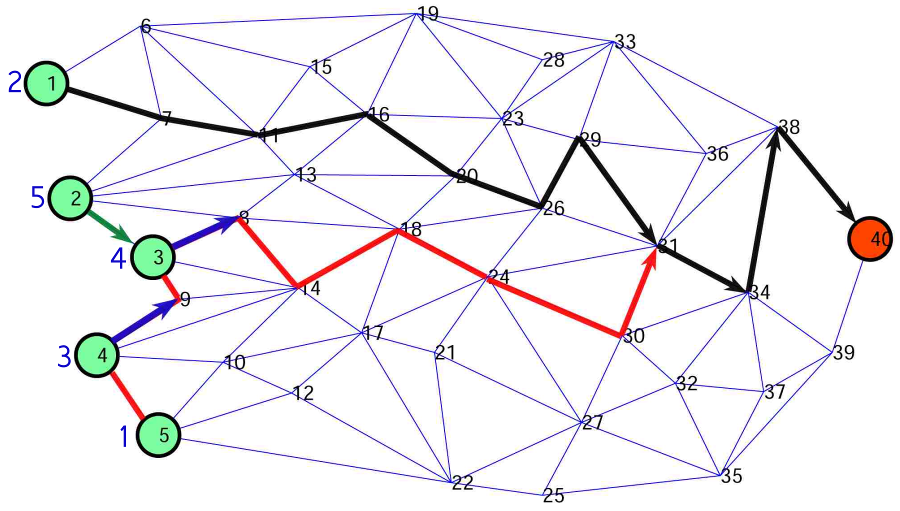


Figure 2.1: Ants traveling routine.

An example is illustrated in Figure.2.1 where the ant colony searching sequence is randomly determined at the beginning of the algorithm iteration in the order: $5 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2$ as indicated with the numbers beside the timber selling nodes. Ant colony C_5 is the first to construct solution: $N_5 \rightarrow N_4 \rightarrow N_9 \rightarrow N_3 \rightarrow N_8 \rightarrow N_{14} \rightarrow N_{18} \rightarrow N_{24} \rightarrow N_{30} \rightarrow N_{31} \rightarrow N_{34} \rightarrow N_{38} \rightarrow N_{40}$. The ant colony C_1 is the second and joins the route found by C_5 at the node N_{31}. Ant colonies C_4 and C_3 are next but do not search for solutions because the entry nodes are already part of the route of C_5. Colony C_2 is the last in the search sequence and again joins colony

C_5's route at the node N_3. Note that this search routine is designed specifically for the fixed charged cost transportation problems. Less fixed cost is incurred with more edges or routes shared by timber sellers, because they only have to pay the fixed cost once for a road segment. On the other hand, the variable cost is calculated with combined timber volumes from all timber selling nodes.

2.2.2 Edge Selection & Back Track Routine

When searching routes, circular paths, which are formed when ants move to a previously visited node, should be avoided. That is the edges leading to a circle should not be considered for solution components. This will cause ants moving back to the previous node if no available edges at front. The edges that the ant moves back from are marked as unavailable in order to prevent them from being selected again. The ant will continue moving back to a point where edges are available for selection again. If the ant moves back to its starting node where still no edges are available, ACO will discard the current iteration and start a new round of route search.

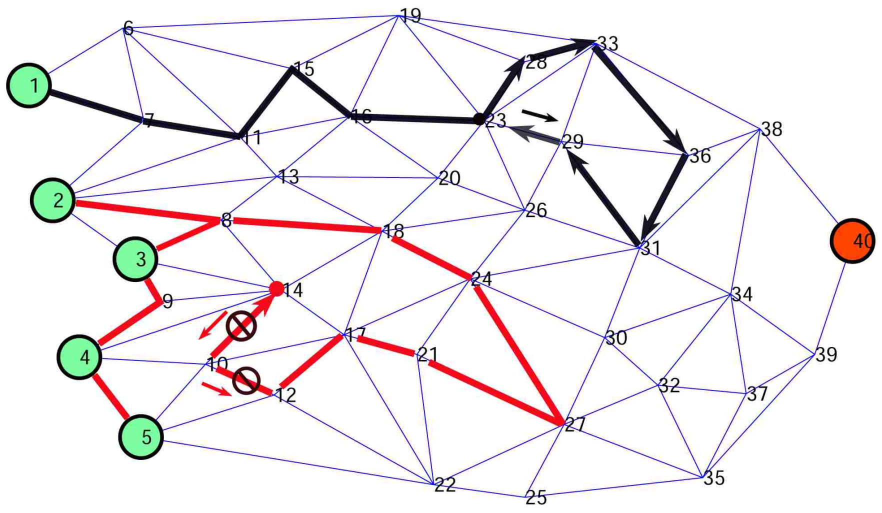


Figure 2.2: Ants back track routine.

An example is illustrated in Figure.2.2. The ant a_1 from the colony C_1 travels to N_{29}. The edge $E_{29,23}$ cannot be selected because a circle will be created as node N_{23}

is already visited. Similarly to the traveling route for the colony C_2, C_3, C_4 and C_5 (marked in red), there are no available edges to be selected at node N_{14}, so the ant has to move back to N_{10} and the edge $E_{10,14}$ is then marked as unavailable. At N_{10}, there are still no available edges to choose, so the ant continues to move back to N_{12} and the edge $E_{12,10}$ is marked as unavailable (indicated as a circle with an oblique line across it).

Pheromone update mechanism

The pheromone updating process is designed to have two stages. In the first stage, ants are set to find the feasible solutions only. In this stage, the algorithm only updates pheromone when there is a better solution found according to the total sediment amount. If a solution is found better than the best-so-far solution found from previous iterations, the new solution becomes the best solution. This will very quickly lead to a feasible solution if one exits, or stop the algorithm from searching further at the second stage if otherwise.

After the feasible solution is successfully found at the first stage, the algorithm will switch to the second stage of which the objective is to find the best feasible solution. The pheromone is updated when a better feasible solution is found, i.e., the solution has lower costs and its total sediment amount satisfies the constraint.

Local search refinement

Local refinement procedures have shown to improve solution quality for different ACO based algorithms [34, 77, 78]. In our ACO algorithm, a local search in the form of a 1-opt routine was implemented into our algorithm. Corresponding to the pheromone update process, the local search is based on sediment yield only during stage one and based on all three edge attributes during stage two.

After an iteration is completed, the local search procedure consists of looking at each node along the routes forming the solution and its adjacent nodes also forming the solutions. For a given node n_i forming a route $R(n_s \rightarrow \ldots \rightarrow n_i \rightarrow \ldots \rightarrow n_n)$, the local search procedure looks at adjacent nodes of n_i other than n_{i-1} and n_{i+1} along the route and evaluates the edges to these nodes to determine if there is a shortcut that eliminates either n_{i-1} or n_{i+1} from the route. Figure 2.3 shows an example of a selected route (red path in Figure 2.3 a) on which the local search refinement procedure is applied and resulting in eliminating n_7 and n_{11} from the route ((red path in Figure 2.3 b)).

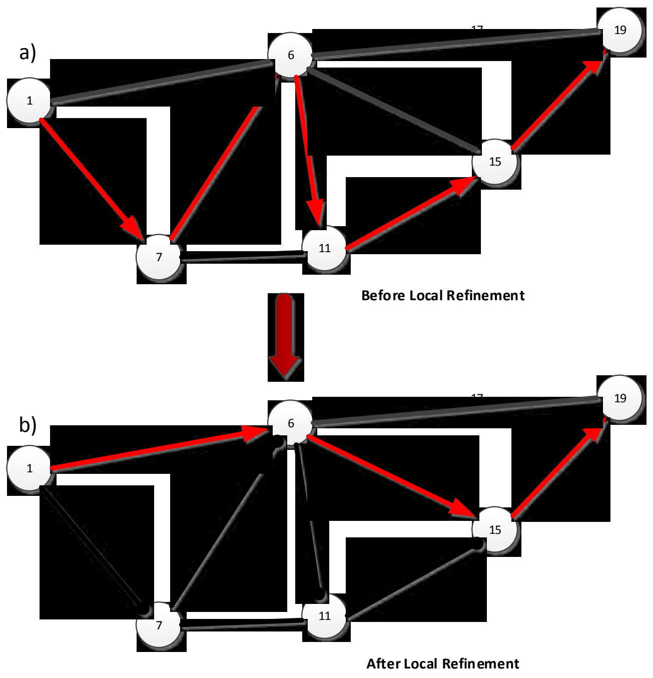


Figure 2.3: Diagram illustrating the 1-opt local search refinement procedure implemented into our ACO algorithm.

2.2.3 Stopping Criterion

Three stopping criteria are implemented into our ACO algorithm to address solution quality stagnation and solving time efficiency. During stage one the algorithm tracks the number of iterations evaluated, and if a user-defined maximum number of iterations is exceeded without finding a feasible solution, the algorithm stops and reports no feasible solution found. During stage two, the algorithm tracks the number of consecutive infeasible iterations, and if it exceeds a user-defined maximum number, then the algorithm stops and reports the best feasible solution found. Each time a feasible solution is found, the algorithm resets the associated counter to zero. Also during stage two the algorithm tracks the number of consecutive feasible solutions found of inferior quality than the best found so far, and if it exceeds a user-defined maximum number, the algorithm stops and the best feasible solution found so far is reported. In our ACO the maximum number of iterations is set to 10,000 for all cases.

2.2.4 Formulations of Transition Probability and Pheromone Update

Formulas for calculating transition probability and pheromone update are defined differently for the two stages, as a result from the different objectives. In the first stage, the objective is to find a feasible solution only. Therefore, the transition probability is defined as:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha[(Sedi_{ij})^{-1}]^\beta}{\sum_{j\in\phi_i}[\tau_{ij}(t)]^\alpha[(Sedi_{ij})^{-1}]^\beta} \tag{2.1}$$

where t is the iteration counter, ϕ_i denotes for a set of available neighboring edges adjacent to node i. The corresponding formula for pheromone update is defined as:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \times \rho + \Delta\tau_{ij}(Sedi) \tag{2.2}$$

where $\Delta\tau_{ij}(Sedi)$ is given as:

$$\Delta\tau_{ij}(Sedi) = \begin{cases} \dfrac{Q}{Sedi_{ij}} & \text{if edge } (i,j) \text{ is part of the route} \\ 0 & \text{Otherwise} \end{cases} \tag{2.3}$$

The edges associated with larger sediment values will receive less pheromone update and the edges with smaller sediment values will receive more pheromone update. In the first stage, only the sediment value on each edge is considered for the local refinement.

In the second stage, the objective is to find a feasible solution with minimum fixed cost and variable cost. Transition probability is calculated considering fixed cost, variable cost and sediment amount. Therefore, its formula is defined as:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \times [(\lambda(FC_{ij} + VC_{ij}) + (1-\lambda)Sedi_{ij})^{-1}]^{\beta}}{\sum_{j \in \phi_i} [\tau_{ij}(t)]^{\alpha} \times [(\lambda(FC_{ij} + VC_{ij}) + (1-\lambda)Sedi_{ij})^{-1}]^{\beta}} \tag{2.4}$$

The formula for pheromone update is defined as:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \times \rho + \Delta\tau_{ij}(t), \tag{2.5}$$

and

$$\Delta\tau_{ij}(t) = \begin{cases} \dfrac{Q}{\lambda[FC/\sum Vol_i + VC] + (1-\lambda)Sedi} \\ 0 \end{cases} \tag{2.6}$$

where $\sum Vol_i$ is the sum of all the timber volumes transported through the edge. From the formula 2.6 we can see that the amount of pheromone used for the pheromone update process is determined based on not only the costs and sediment but also the number of timber volumes being transported through the edge, because the more timber volumes transported via the edge the larger amount pheromone to be added to the solution components. For the local refinement, all three attributes are considered for comparison: $(FC + VC) \times \lambda + Sedi \times (1 - \lambda)$.

2.3 Experimental Studies

2.3.1 Experiment Setup

The ACO algorithm was applied to a 500-edge network presented in Contreras at. el. [17] (Figure 2.4). This hypothetical problem allows traffic in both directions (thus 1000 edges); it considers 25 timber sale locations with a total volume of 36,500 m^3 to be delivered to one mill destination in a single period. Variable cost, fixed cost, and sediment yield per edge ranged from \$0.01/m^3 to \$10/m^3, from \$0.1 to \$23,000 for road construction and maintenance, and from 0.4 to 200 tons, respectively.

We also considered four cases with increasing level of sediment constraint. Case I is a cost minimization problem without a sediment constraint, cases II and III are cost-minimization problems subject to increasing levels of upper-bound sediment constraints, 2,000 and 1,500 tons respectively, and case IV is a sediment-minimization problem without a cost constraint.

Table 2.1: Range of parameter values considered.

Parameter	**Interval**	**Steps**
α	[0, 1]	0.05
β	[0, 1]	0.05
ρ	[0, 1]	0.05
δ	0.00001	
Q	0.001, 0.0001, 0.00001	
λ	[0, 1]	0.1

Table 2.2: Best parameter combination found for each case.

Case	α	β	ρ	λ
I	0.5	0.4	0.55	1
II	0.5	0.9	0.6	0.7
III	0.5	0.7	0.65	0.7
IV	0.45	1	0.15	0

ACO parameters have been shown to have a significant effect on solution quality. Thus we conducted an exhaustive parameter search to find the best value for the four parameters in our ACO algorithm namely: α, β, ρ, and λ. Table 2.1 shows the range of values considered when searching for the best parameter values, totaling 240,000 parameter combinations. Each parameter combination was applied ten times and the combination providing the best solution on average was selected as the best

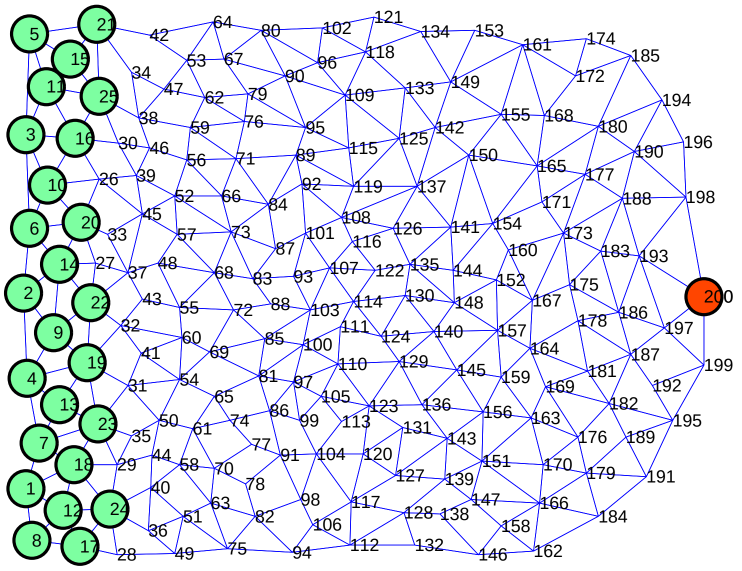


Figure 2.4: The hypothetic FTPP problem instance.

parameter combination. This exhaustive parameter search was conducted for all four cases and resulted in a different best parameter combination for each case. From Table 2.2 we can see that although pheromone importance (α) was relatively similar for all cases, the importance of the edge attributes (β), sediment yield and costs, increased as sediment constraint level became more limiting. Pheromone persistence (ρ) increased with increasing level of sediment constraint with the exception of case IV. As expected the importance of the sediment yield ($1-\lambda$) increased as the sediment restriction level increased.

2.3.2 Experiment Results and Discussions

The four test cases were solved to optimality by the MIP solver, providing a benchmark for comparing solutions found by the ACO algorithm. The results (Table 2.3) show that our ACO algorithm was able to find the optimal solution for cases I and IV. We used these two unconstrained test cases (cases I and IV) as a reference to obtain a meaningful sediment constraint range and set two increasing levels for the constrained cases (cases II and III). As the ACO algorithm presented in this study was designed to address constrained FTPP, it was able to reach high quality solutions for all cases and instances. The best found ACO solutions for cases II and III, achieved optimality levels of 96.7% and 96.1%, respectively.

Table 2.3: Objective function value comparisons between MIP and ACO for the Hypothetical FTPP.

Case	**ACO (Objective Value - $)**	**Sediment Constraint Value (tons)**	**MIP Objective Value - $**	**Percent Different**
I	1,496,562	N/A	1,496,562	0.00%
II	1,637,860	2,000	1,585,393	3.31%
III	2,086,280	1,500	2,008,344	3.88%
IV	948.6	N/A	948.6	0.00%

To test the overall robustness of the ACO algorithm and its ability to consistently find high quality solutions on different problems of similar size using the same parameter values found for the original FTPP, we created a set of 10 different problem instances on the same transportation network (Figure 2.5). These ten problem instances were created by randomly assigning timber sale locations and the mill destination to different nodes. Timber volume at each sale location as well as the three edge attributes (fixed and variable transportation costs and sediment yield) on each edge remained the same for all problem instances. As with the original FTPP, we also considered the four problem cases with increasing level of sediment constraint. In order to compare ACO solution quality, we used the MIP solver to obtain optimal solutions. The total sediment amount associated with the optimal solution for case I (cost minimization) and the objective function of the optimal solution for case IV (sediment minimization) were used as the upper and lower limits of the sediment constraint in cases II and III. For all ten instances, one third and two thirds of the difference between the upper and lower limits were subtracted from the upper limit to determine the sediment constraint values for cases II and III, respectively.

The ACO algorithm was able to successfully solve all ten problem instances (Table

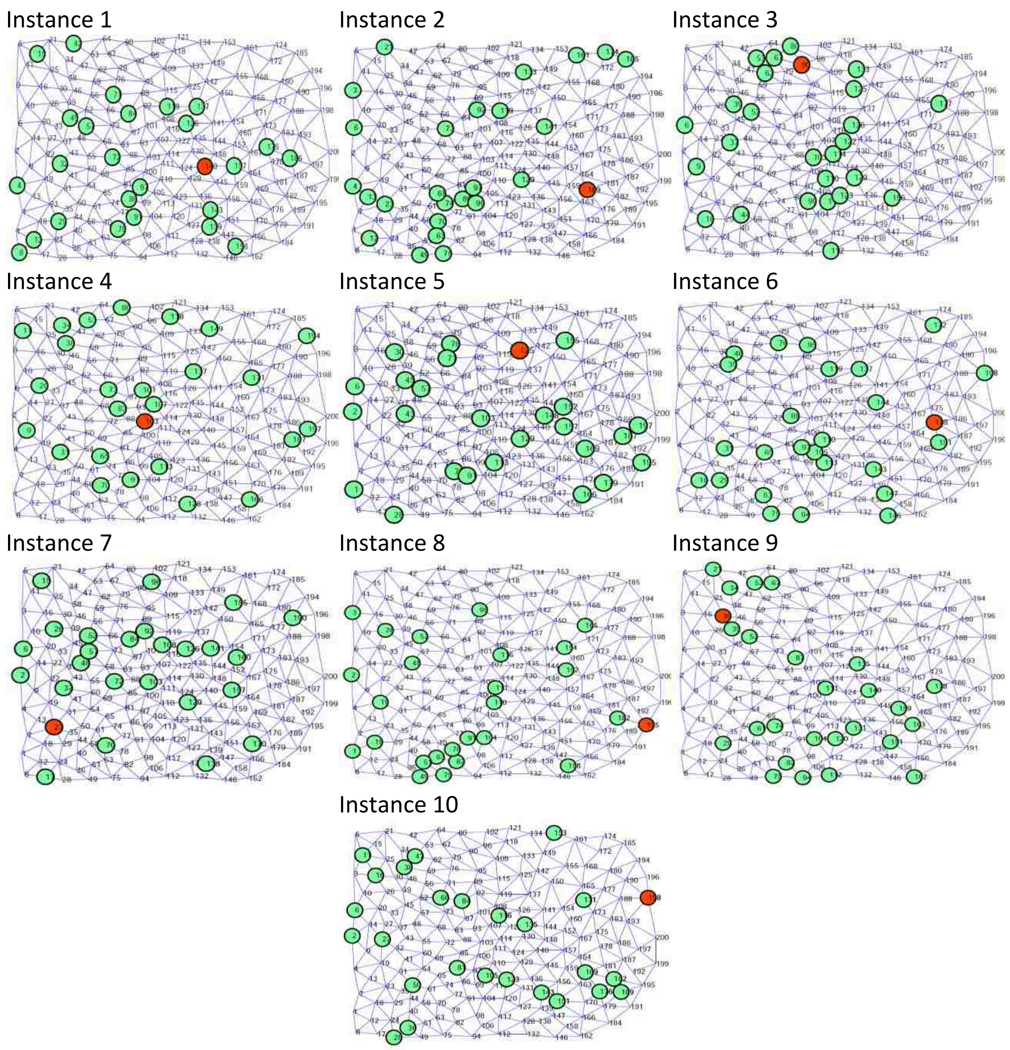


Figure 2.5: Ten problem instances created on the hypothetical FTPP.

2.4), four cases for each of the ten instances, obtaining high quality solutions and in many cases matching optimal solutions. For case I, the ACO algorithm was able to find optimal solutions for seven of the ten instances, and on average for all ten instances ACO solutions were 99.8% optimal. For case IV, the ACO algorithm found optimal solutions for all problem instances but number five, which solutions quality was 99.6%. ACO solutions for the constrained FTPP (cases II and III), for which the ACO algorithm was designed, also provided near-optimal solutions. For case II, ACO solution quality ranged from 97.7% to 99.9% with an average solution quality of 98.9%. As problem complexity increased in case III due to the more strict constraint, ACO solution quality averaged 97.8% with a range between 95.3% and 100%. Consistently for all problem instances, solution quality for case II was slightly better than for case III, mainly because of the fewer feasible solutions evaluated per unit of time.

In general, the ACO algorithm was able to produce near-optimal solutions for all tested problem cases. Although MIP was able to find optimal solution, the ACO algorithm only required a fraction of time requested by MIP (Table 2.5). On average, the best solution found by the ACO algorithm required about 25% (18 vs. 79 sec) and 1% (24 vs. 1678 sec) of the computing time required by the MIP solver to find the optimal solution for cases I and IV. For case II, the ACO was relatively similar between each instance, taking between 190 and 960 sec with an average of 544 sec. On the other hand, computing time required by the MIP solver was about 18 times larger (9,949 sec). Due to the complexity of the problem, ACO and MIP solution times for case III were on average much larger and more variable than those for case II. ACO solution times varied from 360 to 29,000 sec with an average of 5,370 sec, which is only about 7.3% of the average time required by the MIP solver.

2.4 Concluding Remarks

In this study, a customized ACO algorithm was developed to solve CFTPP considering fixed and variable costs as well as a sediment constraint representing the negative environmental impact of timber transport. The ACO metaheuristic, as most approximation algorithms, is highly dependent on problem specific fine tuning of parameters to ensure high quality solutions. Consequently, after conducting an exhaustive parameter search process on the hypothetical network considered in this study, the ACO algorithm was able to find optimal and near-optimal solutions. The best parameter combination found for each case of the original hypothetical network was applied to ten different problem instances. Resulting ACO solutions for the constrained problems (cases II and III) were on average 98.4% optimal, which indicated consistent

Table 2.4: Objective function comparisons between MIP and ACO solutions for cases I, II, III and IV of the ten different problem instances.

Instance	**Case**	**ACO (Objective Value - $)**	**Sediment Constraint Value (tons)**	**MIP Objective Value - $**	**Percent Different**
1	**I**	866254	N/A	866254	0.00%
	II	887,719	2,159	878,749	1.02 %
	III	1,027,550	1,727	981,203	4.72 %
	IV	1294	N/A	1294	0.00%
2	**I**	1361070	N/A	1360965	0.01%
	II	1,416,090	2,490	1,415,960	0.01 %
	III	1,619,740	1,860	1,563,669	3.59 %
	IV	1230	N/A	1230	0.00%
3	**I**	1016500	N/A	1016500	0.00%
	II	1,055,330	2,254	1,048,768	0.63 %
	III	1,174,630	1,746	1,170,956	0.31 %
	IV	1236	N/A	1236	0.00%
4	**I**	897677	N/A	897677	0.00%
	II	914,972	2,449	910,152	0.53 %
	III	1,043,763	1,778	1,043,763	0.00 %
	IV	1106	N/A	1106	0.00%
5	**I**	1171130	N/A	1171130	0.00%
	II	1,203,500	2,445	1,181,284	1.88 %
	III	1,301,920	1,945	1,260,541	3.28 %
	IV	1389	N/A	1384	0.37%
6	**I**	1173387	N/A	1173387	0.00%
	II	1,212,620	2,354	1,208,610	0.33 %
	III	1,398,950	1,760	1,355,860	3.18 %
	IV	1163	N/A	1163	0.00%
7	**I**	1069100	N/A	1050671	1.75%
	II	1,089,140	2,672	1,066,148	2.16 %
	III	1,164,660	1,978	1,164,368	0.03 %
	IV	1283	N/A	1283	0.00%
8	**I**	1214400	N/A	1211218	0.26%
	II	1,241,760	2,660	1,229,392	1.01 %
	III	1,418,220	1,971	1,361,841	4.14 %
	IV	1343	N/A	1343	0.00%
9	**I**	1322930	N/A	1322930	0.00%
	II	1,410,850	2,262	1,378,432	2.35 %
	III	1,679,540	1,734	1,636,147	2.65 %
	IV	1205	N/A	1205	0.00%
10	**I**	1328930	N/A	1328930	0.00%
	II	1,403,150	2,342	1,394,355	0.63 %
	III	1,634,760	1,750	1,628,223	0.4 %
	IV	1132	N/A	1132	0.00%

Table 2.5: Comparison of computing times (sec) for a single run required by the ACO algorithm and the MIP solver for the four cases of each of the ten instances.

	ACO				MIP			
Instance	**Case I**	**Case II**	**Case III**	**Case IV**	**Case I**	**Case II**	**Case III**	**Case IV**
1	17	434	363	23	41	3,254	31,722	2,790
2	14	263	2,732	25	134	62,314	90,973	500
3	16	790	428	25	65	3,532	21,893	2,147
4	21	708	396	23	95	4,732	36,385	1,275
5	18	190	29,051	19	64	3,110	149,585	54
6	21	304	8,885	24	69	4,344	97,582	2,922
7	21	905	371	28	76	1,333	31,540	2,933
8	20	371	10,722	31	68	8,458	55,516	1,786
9	14	509	332	20	84	1,182	152,629	1,157
10	20	962	419	26	98	7,225	59,597	1,212
Average	18	544	5,370	24	79	9,949	72,742	1,678

results and overall robustness of the algorithm.

The algorithm developed in this study has a great application potential to ensure the economic efficiency of timber transport operations, which is the largest cost component of timber harvesting. However, the ACO algorithm needs improvement to ensure solution quality and time efficiency for larger and more complex, real-world FTPP. Future work should be focused on time efficient technique to configure parameter values without the need to conduct an exhaustive parameter search. The current version of the algorithm coded in a sequential fashion, thus incorporating parallelization is likely to reduce solution time and allow addressing large-scale problems.

3 Automatically Configuring ACO Using Multilevel ParamILS to Solve Transportation Problems

3.1 Introduction

In this Chapter, we present the design, implementation and testing of a multilevel online parameter configuration framework (MParamILS) for ACO, which combines a graph coarsening heuristic and the ParamILS framework. The proposed multilevel framework configures ACO algorithms to solve the original problem, which might be computationally expensive, using a set of increasingly coarser level problems of which the computational cost is cheaper. The graph coarsening technique (described in Chapter 1) is recursively applied to the original problem, from which the general structure and certain edge properties are inherited by the resulted coarser level problems. The ParamILS is a state of the art parameter configuration algorithm.

MParamILS is robust for it does not rely on any specific problem types or require predetermined good parameter value sets. Moreover, the framework developed in this work can be potentially generalized to other problem domains and optimization algorithms. For example, instead of using ParamILS, other parameter configuration methods can be used to solve different kinds of problems (such as those presented in [9, 41, 15]). The following sections provide preliminary knowledge and detailed descriptions of MParamILS.

3.2 Preliminary Knowledge

3.2.1 Challenge for Setting Parameters for ACO-based Algorithms

As aforementioned, the performance of ACO-based algorithms are highly dependent on the values of their parameters. Without proper parameter settings, ACO-based algorithms can either converge very slowly or stagnate in local optimal solutions [27, 71, 33]. However, setting parameters for ACO algorithms is a difficult task because users have to balance between diversification and intensification [29, 24].

On one hand, when choosing parameter values that emphasize diversification, the final solution quality is often better, but more computing time is required. On the other hand, when choosing parameter values that emphasize intensification, ACO algorithms converge quickly but often to sub-optimal solutions.

Traditionally, ACO parameters were configured by manually changing one parameter at a time while keeping other parameters constant [33], which was similar to the Coordinate Search. Such a parameter tuning method was time consuming and prone to human errors [81]. In addition, the performance of the Coordinate Search is highly affected by the initial search point and step size. Without a proper initial setting, the Coordinate Search may converge very slowly and often obtain local optimal solutions [84].

Automatic parameter configuration techniques have been developed to improve the performance of the manual parameter configurations [72]. Broadly, the automatic configuration techniques are categorized into "online", which modifies an algorithm's parameter values while solving a problem instance, and "offline", which adjusts parameter values before the target algorithm is actually deployed [71, 31, 14]. For example, Khichane et al. proposed two online frameworks (named GPL and DPL) that automatically adapted parameters for ACO algorithms at runtime [55]. The idea behind the frameworks resembled the pheromone update mechanism of ACO and the experiments showed positive results. However, discrete sets of parameters must be known a priori, and they were assumed to contain good values, which should allow ACO to find good results. This requirement is often difficult to meet for parameters with large value ranges. Birattari et al. [10, 6, 11] presented the iterated F-Race (I/F-Race) method for offline algorithm configurations. This method consisted of sampling parameter values from a probability distribution, selecting the best parameter settings according to the results of F-Race, and updating the probability distribution to bias the sampling towards good parameter values. Although the method was used to automatically tune parameters, I/F-Race required a well defined probability distribution that is usually difficult to determine if a limited set of instances are available [10], and the adoption of a full factorial decision was impractical and computationally prohibitive for a large number of parameter configurations [6]. Manuel et al. [66] proposed two offline parameter variation strategies called "delta" and "switch", which were used for changing parameter values in the I/F-Race. In the delta strategy, parameter values first increased by a certain amount at each algorithm iteration, and then decreased when the values exceed a maximum allowable range. In the switch strategy, parameter values were either randomly selected from a value range or kept constant at each algorithm iteration. The experimental results showed that the performance of the automatic configuration method was able to match that obtained by the parameter variation strategies designed by a human expert.

3.2.2 Algorithm Configuration Problem

Generally speaking, setting parameters for ACO is an algorithm configuration problem that is comprised of a set of input data, a given target algorithm A, and a parameter domain $\Theta = \mathit{\Theta}_1 \times \ldots \times \mathit{\Theta}_k$, where $\mathit{\Theta}_i$ represents a parameter value set. Let $A(\vec{\theta})$ be an instantiation of the target algorithm with a parameter combination $\vec{\theta} \in \Theta$, $O_{A(\vec{\theta})}$ the objective function that measures the observed cost for running $A(\vec{\theta})$, and $\mathbb{E}$ a statistical measurement such as the expectation, median, or norm. The objective of the algorithm configuration problem is to find the parameter combinations $\Theta^* \subseteq \Theta$ such that the target algorithm achieves the best possible performance on any input data that minimizes $\mathbb{E}(O_{A(\vec{\theta})})$, $\forall \vec{\theta} \in \Theta^*$. Mathematically, the objective of the algorithm configuration problem is described as

$$\Theta^* = \vec{\theta} : \arg\min \mathbb{E}(O_{A(\vec{\theta})}), \vec{\theta} \in \Theta \tag{3.1}$$

Accordingly, algorithms that solve the algorithm configuration problem are called configurators, and algorithms to be configured are called target algorithms.

3.2.3 ParamILS Framework

ParamILS framework is a state of the art procedure to solve the algorithm configuration problem [46]. To configure parameters, ParamILS makes a sequence of algorithm runs for each parameter combination. It then, selects the best one based on the statistical information calculated from obtained objective values.

In order to properly compare between two parameter combinations, a sufficient number of algorithm runs are required for calculating the necessary statistical information. Due to the stochastic nature of ACO algorithms, a high-quality or the optimal solution may be obtained by a low-quality parameter setting purely by chance. Therefore, the expected objective function value is used to assess qualities of parameter combinations.

As the expected values are calculated with the number of algorithm runs, two parameter combinations are comparable only when they have been evaluated the same number of times. Therefore, we formally state in Definition 1 that a parameter combination $\vec{\theta}_1$ dominates $\vec{\theta}_2$ if the expected objective value of $\vec{\theta}_1$ is better than that of $\vec{\theta}_2$ for the same number of algorithm runs:

Definition 1. *(Domination). $\vec{\theta}_1$ dominates $\vec{\theta}_2$ if $N(\vec{\theta}_1) \geq N(\vec{\theta}_2)$ and $\mathbb{E}(\vec{\theta}_1) < \mathbb{E}(\vec{\theta}_2)$, where $N(\vec{\theta}_1)$ = length($R_{\vec{\theta}_1}$) and $R_{\vec{\theta}_1}$ denotes for algorithm runs on the parameter combination $\vec{\theta}_1$.*

Definition 1 is used to modify the original ParamILS framework for configuring ACO algorithms.

3.3 Multilevel ParamILS

3.3.1 Multilevel Parameter Configuration Framework

The proposed multilevel framework (Figure 3.1) consists of a target algorithm for parameter configuration, a problem that is coarsened into a set of increasingly coarser level problems, a parameter domain that contains possible parameter combinations, and a configurator that runs the target algorithm to evaluate the qualities of parameter combinations in the domain [63].

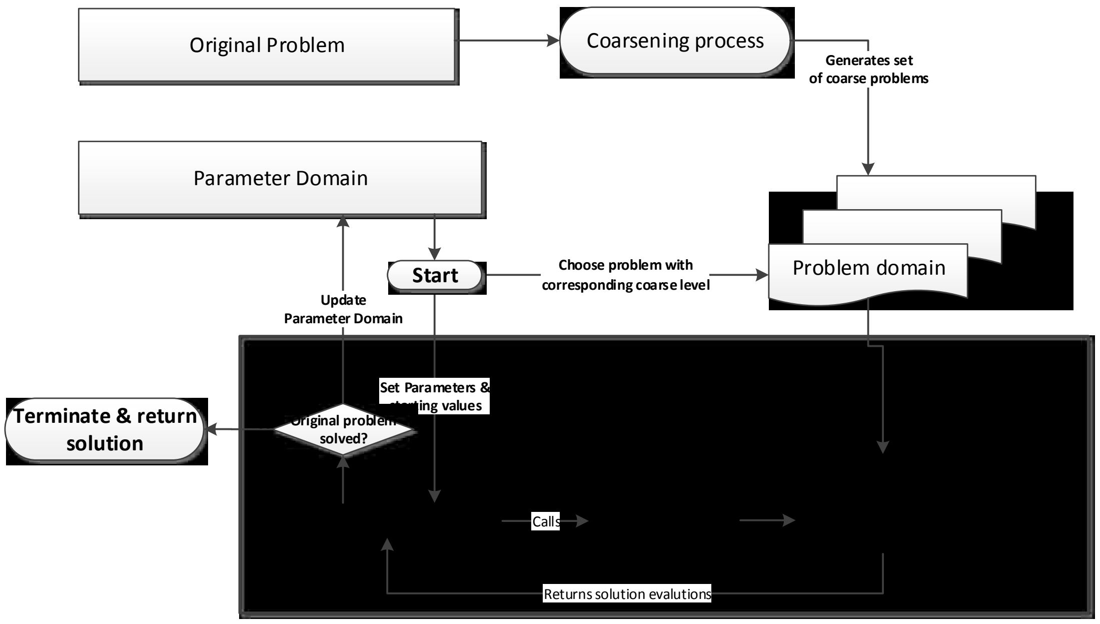

Original Problem
Coarsening process
Generates set of coarse problems
Parameter Domain
Start
Choose problem with corresponding coarse level
Problem domain
Update Parameter Domain
Set Parameters & starting values
Terminate & return solution
Original problem solved?
Calls
Returns solution evalutions

Figure 3.1: Multilevel parameter configuration framework.

In this study, the ACO-based algorithms developed for transportation problems with underlying weighted networks are the target algorithms, and the ParamILS is used as the configurator. The qualities of parameter combinations in the parameter domain are evaluated by running the target algorithm. Based on the obtained objective function values, low-quality parameter combinations can be identified and removed from the domain. This evaluation process is applied first to the coarsest level problem, and then applied subsequently to the next coarser level problem, then to the finest level (original) problem. Because of the smaller sizes of coarser level problems, evaluating parameter combinations on a coarser level problem is expected to finish quicker than that on the original problem. The size of the domain decreases as the low-quality parameter combinations are constantly being eliminated. As a result, the required computing time is reduced due to fewer number of parameter combinations having to be evaluated at finer level problems. After the parameter evaluation process on the coarser level problems, the remaining parameter combinations in the domain at the finest level are considered high-quality and used to solve the original problem.

3.3.2 Modifying ParamILS

A modified ParamILS framework (Algorithms 3 to 5) is implemented according to Definition 1 to configure ACO, where procedure *Objective* (Algorithm 3) evaluates a parameter combination and returns the expected objective value. For the evaluated parameter combinations, a global cache is used to archive their evaluation information, which contains the number of times the parameter combination has been evaluated (*runtime*), the best objective value obtained, and the sum of all the obtained objective values (*sum*). *Objective* procedure takes an integer value (N), which is the number of algorithm runs required for the evaluation, and a threshold *bound*, which is used to stop the evaluation process for low-quality parameter combinations. Before evaluating a parameter combination $\vec{\theta}$, its *runtime* is extracted from the global cache to compare with N. If $\vec{\theta}$ has already been evaluated N times (i.e., $runtime \geq N$), it will not be evaluated again and *Objective* simply returns the expected objective value calculated based on the evaluation information. Otherwise, $\vec{\theta}$ is evaluated until the number of evaluations equals N or the parameter combination is considered as low-quality. This is determined when the expected objective value exceeds the *bound*. Then, *Objective* stops evaluating the parameter combination further and returns a worst possible objective value (i.e., a big number for minimization problems) as an indication of bad quality.

Note that a proper *bound* value can save computing time by avoiding evaluating

Algorithm 3 Objective($\vec{\theta}$, N, bound). $R[i]$ denotes i^{th} ACO run.

```
Input(Parameter Set $\vec{\theta}$, number of runs: $N$, bound)
Output(the expected objective value $E(\vec{\theta})$)

begin
    $runtime \leftarrow$ number of algorithm runs for $\vec{\theta}$
    $sum \leftarrow$ sum of objective values in $R_{\vec{\theta}}[1], \ldots, R_{\vec{\theta}}[runtime]$
    if $runtime \geq N$ then
        return($sum/runtime$)
    end
    foreach $i = runtime$ to $N$ do
        $O_i \leftarrow$ objective from a newly executed run of $A(\vec{\theta})$
        $sum \leftarrow sum + O_i$
        $runtime \leftarrow runtime + 1$
        $R_{\vec{\theta}}[i] \leftarrow O_i$
        // terminate early for bad parameter set
        if $(sum/N) > bound$ then
            return($WorstPossibleObjective$)
        end
    end
    return($sum/runtime$)
end
```

low-quality parameter combinations. An improper *bound* value (such as too small or too large), however, can either cause a good parameter combination being falsely labelled as low-quality or a bad parameter combination not being identified. In this study, based on the preliminary experimental results, *bound* is set to be twice as big as the best expected objective value found. However, we emphasize that this setting is by no means optimal or universal. For a different set of testing problems, a new *bound* setting may be more appropriate to achieve good results.

Procedure *better* (Algorithm 4) compares two parameter combinations $\vec{\theta}_1$, $\vec{\theta}_2$. It returns true if $\vec{\theta}_1$ dominates $\vec{\theta}_2$ and false otherwise. The *Objective* procedure and an auxiliary procedure *Dominate* are used to make the comparison. To determine which parameter combination is superior, $\vec{\theta}_1$ and $\vec{\theta}_2$ have to be evaluated for the same number of times. If both parameter combinations produce the same expected objective value,

Algorithm 4 better($\vec{\theta}_1$, $\vec{\theta}_2$).

```
Input(Parameter Set $\vec{\theta}_1$, Parameter Set $\vec{\theta}_2$)
Output(true if $\vec{\theta}_1$ dominates $\vec{\theta}_2$, false otherwise)
begin
    if $\vec{\theta}_1 == \vec{\theta}_2$ then
        return(true)
    end
    if $N(\vec{\theta}_1) \leq N(\vec{\theta}_2)$ then
        $N \leftarrow N(\vec{\theta}_1)$
    else
        $N \leftarrow N(\vec{\theta}_2)$
    end
    while true do
        $N \leftarrow N + 1$
        Objective($\vec{\theta}_1$, N, bound)
        Objective($\vec{\theta}_2$, N, bound)
        if Dominate ($\vec{\theta}_1$,$\vec{\theta}_2$) then
            return true
        end
        if Dominate ($\vec{\theta}_2$,$\vec{\theta}_1$) then
            return(false)
        end
    end
end
Procedure: Dominate($\vec{\theta}_1$,$\vec{\theta}_2$)
begin
    if $N(\vec{\theta}_1) < N(\vec{\theta}_2)$ then
        return(false)
    else
        return(
        Objective($\vec{\theta}_1$, $N(\vec{\theta}_2)$, bound) $\leq$
        Objective($\vec{\theta}_2$, $N(\vec{\theta}_2)$, bound))
    end
end
```

better will continue the evaluation process until one of them dominates the other.

The main procedure of ParamILS (Algorithm 5) uses *better*, and the other two auxiliary procedures (*Nbh* and *IterativeLocalImprovement*) to select high-quality parameter combinations. *Nbh* takes a parameter combination and randomly selects one of its neighboring parameter combinations that differ in one parameter value. *Itera-*

Algorithm 5 ParamILS(Θ, $\vec{\theta}_{start}$, r, s); *Nbh*($\vec{\theta}$): returns a neighboring parameter combination; $R_{\vec{\theta}}$: denotes for the evaluation information of $\vec{\theta}$; *getbest*($R_{\vec{\theta}}$): returns the best objective value obtained; *IterativeLocalImprovement*: iteratively compares the neighboring parameter combination with the current best one.

```
Output(Best parameter seting, parameter combination set)
begin
    $\vec{\theta}_{best} = \vec{\theta}_{start}$
    foreach i=1 to r do
        $\vec{\theta} \leftarrow$ random $\vec{\theta} \in \Theta$
        if better($\vec{\theta}$, $\vec{\theta}_{best}$) then $\vec{\theta}_{best} \leftarrow \vec{\theta}$
    end
    $\vec{\theta}_{best} \leftarrow$ IterativeLocalImprovement($\vec{\theta}_{best}$)
    while NotTerminationCriterion do
        $\vec{\theta} \leftarrow \vec{\theta}_{best}$
        // parameter local perturbation
        foreach $j = 1$ to $s$ do
            $\vec{\theta} \leftarrow$ random $\vec{\theta}' \in$ Nbh($\vec{\theta}$)
        end
        $\vec{\theta} \leftarrow$ IterativeLocalImprovement($\vec{\theta}$)
        // parameter acceptance
        if better($\vec{\theta}$, $\vec{\theta}_{best}$) then $\vec{\theta}_{best} \leftarrow \vec{\theta}$
    end
    $O_{\vec{\theta}_{best}} \leftarrow$ getbest($R_{\vec{\theta}_{best}}$)
    $E_{\vec{\theta}_{best}} \leftarrow$ E($R_{\vec{\theta}_{best}}$) // expected objective value for $\vec{\theta}_{best}$
    forall the evaluted $\vec{\theta} \in \Theta$ do
        if getbest($R_{\vec{\theta}}$) $\leq O_{\vec{\theta}_{best}}$ and $E(R_{\vec{\theta}}) < BF \times E_{\vec{\theta}_{best}}$ then
            add $\vec{\theta}$ to $\Theta_{selected}$
        end
    end
    return($\Theta_{selected}$, $\vec{\theta}_{best}$)
end
Procedure: IterativeLocalImprovement($\vec{\theta}$)
begin
    while $\vec{\theta}' \neq \vec{\theta}$ do
        $\vec{\theta}' \leftarrow \vec{\theta}$
        foreach $\vec{\theta}'' \in$ Nbh($\vec{\theta}'$) in randomized order do
            if better($\vec{\theta}''$, $\vec{\theta}'$) then $\vec{\theta} \leftarrow \vec{\theta}''$
            break
        end
    end
    return($\vec{\theta}$)
end
```

```
Procedure: Nbh($\vec{\theta}$)
begin
    visit each $\vec{\theta}' \in \vec{\theta}$ in randomized order
    if $\vec{\theta}'$ differs from $\vec{\theta}$ in one parameter then return($\vec{\theta}'$)
end
```

tiveLocalImprovement uses *Nbh* to compare the current best parameter combination $\vec{\theta}_{best}$ with the neighboring parameter combinations. If a neighbouring parameter combination is found to be better, it replaces $\vec{\theta}_{best}$ and becomes the new best parameter combination. *IterativeLocalImprovement* continually refines the best parameter combination until a dominating one, where no better neighbors can be found, is obtained.

ParamILS takes two parameters: r, which is the number of attempts to find a better parameter combination than $\vec{\theta}_{start}$ at the beginning, and s, which is the number of parameter neighboring searches (*IterativeLocalImprovement*). The values of the two parameters affect the overall computing time and final solution quality. In the experiments, r was set to 10, s was set to the number of parameters being configured (α, β, ρ for ACO), and $\vec{\theta}_{start}$ was set to $\alpha = 0.5, \beta = 0.5, \rho = 0.5$. ParamILS iteratively evaluates parameter combinations. Meanwhile, the evaluation information is archived and updated. In the end, a set of good parameter combinations are selected from evaluated parameter combinations according to the evaluation information. A parameter combination is considered good if its best objective value obtained is no worse than the one obtained by $\vec{\theta}_{best}$ and their expected objective value is less than $\mathbb{E}_{\vec{\theta}_{best}}$ times a boundary factor (BF):

Definition 2. *(Selecting good parameter combinations). A parameter combination $\vec{\theta}$ is considered good if $getbest(R_{\vec{\theta}}) \leq getbest(R_{\vec{\theta}_{best}})$ and $\mathbb{E}(R_{\vec{\theta}}) < BF \times \mathbb{E}(R_{\vec{\theta}_{best}})$, where BF is an integer scaler and $getbest(R_{\vec{\theta}})$ is a function that returns the best objective value obtained from evaluating $\vec{\theta}$.*

Similar to the *bound* setting in Algorithm 4, based on the preliminary experimental results, the boundary factor was set to 2 in the implementation. Time complexity of ParamILS depends on the number of parameter combinations to be evaluated. The convergence of ParamILS is proven in [46], where the probability of finding the optimal parameter configuration $\vec{\theta}^*$ approaches one as a number of parameter combination searches goes to infinity.

3.3.3 Multilevel ParamILS Framework

The Multilevel ParamILS framework (Algorithm 6) combines the graph coarsening procedure (Chapter 1) and ParamILS. the graph coarsening procedure coarsens a problem G into a set of increasingly coarser level problems $G_0, G_1, \ldots, G_n$. ParamILS is first applied to the coarsest problem (G_n) with a start parameter combination. It selects high-quality parameter combinations from a parameter combination domain Θ and the identified low-quality parameter combinations are discarded. Next, ParamILS is applied to the second coarsest level problem (G_{n-1}) and uses the best parameter combination selected from the coarsest level problem as the start parameter combination. It again selects high-quality parameter combinations, removes low-quality ones from the Θ, and possibly obtains a new best parameter combination. Subsequently, this process continues to the next level coarser problem until ParamILS is applied to the finest level problem G_0 (original problem), resulting in a significant reduction in size of the Θ at the finest level. Consequently, computing time is greatly saved in that 1) identifying and eliminating parameter combinations on coarser level problems is faster than that on the original problem, 2) the target algorithm (ACO) is expected to converge faster using selected high-quality parameter combinations.

3.4 Experimental Studies

3.4.1 Experiment Setup

Algorithms and procedures were implemented using C++ and uploaded to the Lipscomb High Performance Computing Cluster (HPC) supported and maintained by the University of Kentucky Center for Computational Science. All programs were

Algorithm 6 Multilevel ParamILS framework(Θ, $\vec{\theta}_{start}$, G).

```
Input($\Theta$, $\vec{\theta}_{start}$, Problem $G$)
Output(good parameter combinations set, best solution)
begin
    $(G_n, \ldots, G_0 \Leftrightarrow G) \leftarrow$ obtain coarse problems from $G$
    foreach $i = n$ to 0 do
        // update domain $\Theta$, obtain $\vec{\theta}_{best}$
        $(\Theta, \vec{\theta}_{best}) \leftarrow$ ParamILS($\Theta, \vec{\theta}_{start} G_i$)
        $\vec{\theta}_{start} \leftarrow \vec{\theta}_{best}$
    end
    return($\Theta$, best solution)
end
```

executed on the computing nodes of HPC: Dual Intel E5-2670 of 8 Cores at 2.6 GHz with 64 GB of 1600 MHz RAM and Linux Red Hat OS. To test for performance, MParamILS was compared with an Exhaustive method and the ParamILS (Table 3.1). The Max-Min Ant System (MMAS) [80] was used as the target algorithm and the objective was to configure its three main parameters: α that controls relative importance of pheromone information, β that controls relative importance of heuristic information, and ρ which is the pheromone evaporation rate used to prevent ACO from converging to local optimal or reaching premature stagnation. The value ranges of the three parameters were confined to [0,1] with a step of 0.05 for α, β and 0.1 for ρ, which makes a total 4,000 parameter combination domain. The MMAS was implemented according to the previous study (Chapter 2).

The MMAS was applied to five test cases (Table 3.2). Case I was a fixed charge transportation problem [44] where fixed and variable costs were involved. Case II and Case III were constrained fixed charge transportation problems where constraints of different strictness levels were imposed. Case IV was a minimization problem where the objective was to obtain a set of routes from sources to destinations with minimum total weights. Case V was the Hamiltonian cycle problem, which is a special case of the travelling salesmen problem (TSP) obtained by setting the edge attributes to one

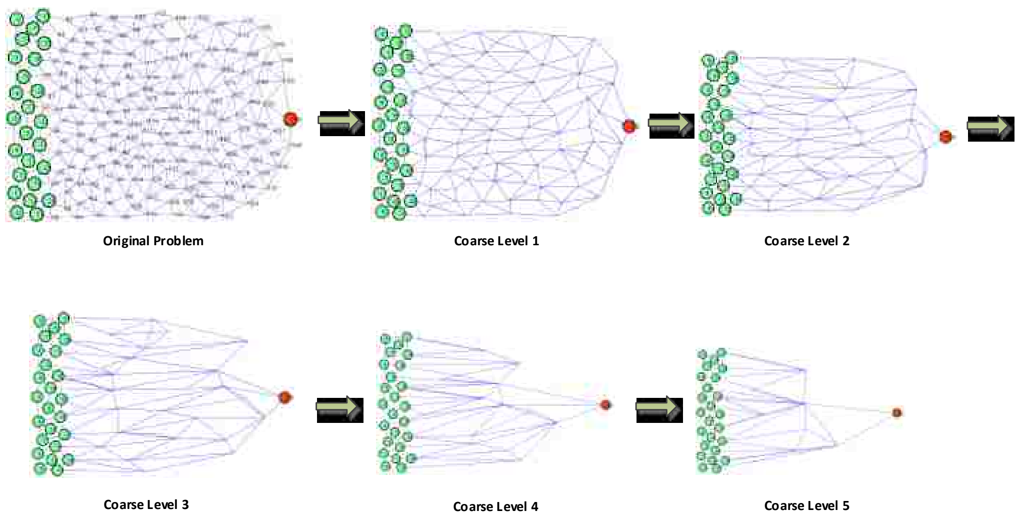


Figure 3.2: Original problem and its subsequent coarser level problems.

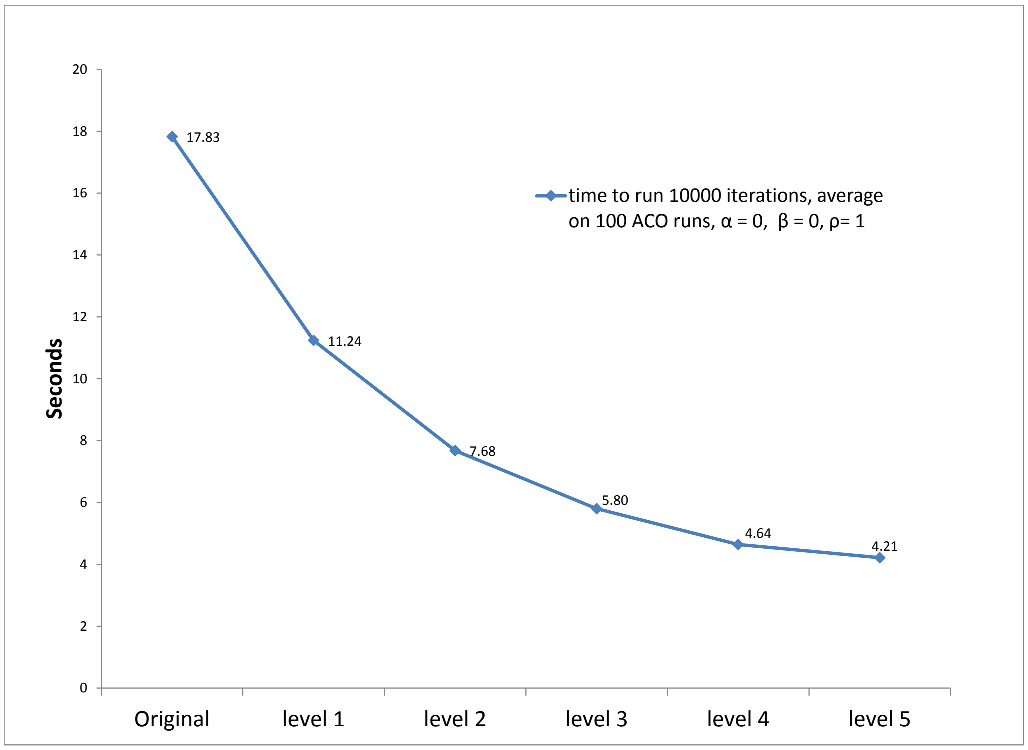


Figure 3.3: Average computing time of running MMAS 100 times on the original problem and its coarse level problems for Case I (ACO stops search for solution after 10000 iterations, $\alpha = 0, \beta = 0, \rho = 1$).

[5].

The test cases were designed on the network that was used in Chapter 2 that consists of 500 road segments and 200 intersection nodes with 25 sources and one

Table 3.1: Tested methods, parameter settings, and running environment.

Method	Parameter Settings	Implementation	OS
Exhaustive	$\alpha \in [0, 0.05, 0.1, \ldots, 0.95, 1]$, $\beta \in [0, 0.05, 0.1, \ldots, 0.95, 1]$, $\rho \in [0, 0.1, 0.2, \ldots, 0.9, 1]$, $\lvert\Theta\rvert = 4000$, $Max\ iteration = 10000$, Max pheromone = 0.01, Min pheromone = 0.00001.	Divide Θ into 10 partitions and use 10 processors to run GuidedACO with each parameter combination partition simultaneously.	C++, Linux.
ParamILS	Same ACO settings as Exhaustive, Number of iterations: 100, Local search $r = 10$, Perturbation $s = 3$, Initial best parameter Setting: ($\alpha = 0.5, \beta = 0.5, \rho = 0.5$).	Parameter settings in Θ is sequentially configured and ParamILS stops after 100 iterations. Best solution is obtained during parameter configuration.	C++, Linux.
MParamILS	Same ACO settings as Exhaustive, Four level problems: Level 0 (original problem), Level 1 (first coarser), Level 2 (second coarser), Level 3 (coarsest).		C++, Linux.

Table 3.2: Test cases, objective functions, and constraints.

Test Case	Name	Objective Function	Constraints	Description
Case I	Fixed charge problem (FCP)	$\min \sum_{i=1}^{n} \sum_{\substack{j\neq i,\\ j\neq 0}}^{n} VC_{i,j} \times E_{i,j} + FC_{i,j} \times E_{i,j}$ where $E_{i,j} = \{0,1\}$		VC: variable cost, FC: fixed cost, Route: multiple sources to one or more destinations.

Case II & Case III	Constrained FCP (CFCP)	$\min \sum_{i=1}^{n} \sum_{\substack{j\neq i,\\ j\neq 0}}^{n} VC_{i,j} \times E_{i,j} + FC_{i,j} \times E_{i,j}$
		where $E_{i,j} = \{0,1\}$
			$\sum_{i=1}^{n} \sum_{\substack{j\neq i,\\ j\neq 0}}^{n} Sed_{i,j} \times E_{i,j} \leq SedRct$	$Sed_{i,j}$: sediment,
				SedRct = 2000 for Case II,
				SedRct = 1500 for Case III,
				Route: multiple sources to one or more destinations.

Case IV	Minimization transportation problem	$\min \sum_{i=1}^{n} \sum_{\substack{j\neq i,\\ j\neq 0}}^{n} Sed_{i,j} \times E_{i,j}$ where $E_{i,j} = \{0,1\}$		Route: multiple sources to one or more destinations.
Case V	Hamiltonian cycle problem	$\sum_{i=1}^{n} \sum_{\substack{j\neq i,\\ j\neq 0}}^{n} E_{i,j}$ where $E_{i,j} = \{0,1\}$	$\sum_{\substack{i=0,\\ i\neq j}}^{n} E_{i,j} = 1$, $\sum_{\substack{j=0,\\ j\neq i}}^{n} E_{i,j} = 1$	Route: single source, every node has to be visited exactly once.

destination. The graph coarsening procedure was applied to the network to produce coarser level problems. Special nodes, such as source and destination nodes, were

Table 3.3: Number of nodes and edges between the original network and coarser level networks. Percentage of nodes and edges reduced from finner level networks to coarser level networks.

	Original	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
# of Nodes	200	116	74	53	41	35
# of Edges	500	300	190	134	102	86
Nodes Reduction Rate	N/A	42.00%	36.21%	28.37%	22.64%	14.63%
Edges Reduction Rate	N/A	40.00%	36.66%	29.47%	23.88%	15.68%

reserved from the coarsening process and copied directly from a finer level to a coarser level problem. Five coarser level problems (Level 1, 2, 3, 4, 5 in Figure 3.2) with increasingly reduced sizes were obtained (Table 3.3). The problem size was reduced considerably from the original problem to the coarser level problems (about 40% from the original problem to Level 1 problem). However, the size reduction rate gradually decreased as the coarser level increased (i.e., only 15% from Level 4 to Level 5). This can be explained by the fact that a coarser level problem contains fewer number of matching edges than a finer level problem. Therefore, fewer number of nodes and edges are aggregated and collapsed when the coarsening process is applied, causing a smaller size reduction rate.

Because the size reduced from a coarser level to the next coarser level problem can be inadequate, not all five coarser level problems might be needed in MParamILS. To select appropriate coarser level problems, we ran MMAS 100 times on each of the coarser level problems (including original problem) to solve for Case I and compared their average running times (Figure 3.3). MMAS was set to stop after 10,000 iterations and the three parameters were set: $\alpha = 0$, $\beta = 0$, $\rho = 1$, which turned off the functionality of the pheromone information and made MMAS a randomized search algorithm. The average computing time reduction was highest from the original problem to Level 1 problem, and lowest from Level 4 problem to Level 5 problem (Figure 3.3). In general, the differences in computing time between coarser level problems became less significant as the problem size differences became smaller. This was especially evident for Level 3, 4 and 5 problems, where their computing times only differed in one or two seconds. Due to the small time difference, we only considered Level 1, 2 and 3 problems in our experiments for the MParamILS.

3.4.2 Experiment Results and Discussions

The Exhaustive method was applied to each test case once and used to compare solution quality. It ran MMAS on each parameter combination 10 times, and the best solutions were selected according to the objective functions. Since sequential exhaustive methods can take a long time, we divided the parameter combination domain into 10 intervals and applied the Exhaustive method to each of them simultaneously. The computing time of the Exhaustive method was calculated by adding the time spent for each interval. ParamILS and MParamILS were run independently 10 times for each test case, and average computing time and objective function value were used for comparisons in the experimental results. Because MParamILS was set to use a different number of coarser level problems, we use $\text{MParamILS}_\text{i}$ to denote the different settings, where i indicates the maximum number of coarsening levels used. For example, MParamILS_4 indicates that Level 0-3 problems were used. Additionally, we use MParamILSs as a collective name referring to MParamILS with all the tested coarsening level settings as a whole.

The performances were first compared between the Exhaustive method and the two heuristic methods. The purpose was to investigate whether the tested heuristics can produce the same solution qualities. No significant differences among the tested methods in terms of solution quality can be observed, as the results obtained for all test cases are reduced to a single point due to the extremely small variances (Figure 3.4). This indicates that ParamILS and MParamILSs were able to match the performance of the Exhaustive method, and high-quality solutions were consistently produced by both heuristic methods. In terms of computing time in average for all

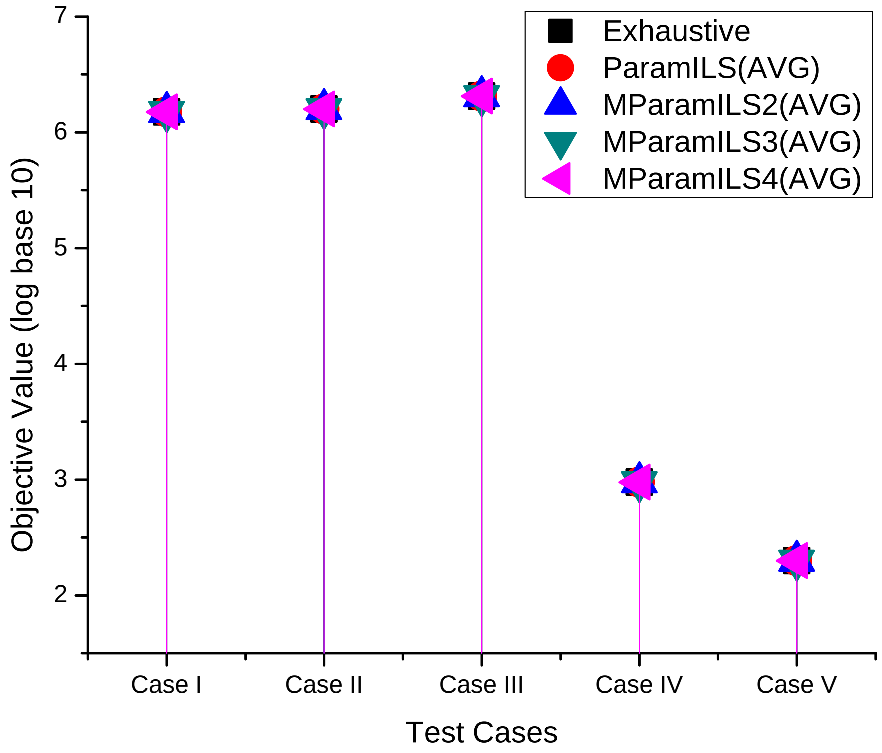


Figure 3.4: Best objective values obtained by the Exhaustive, average objective values obtained by ParamILS and ParamILS_s.

test cases, the Exhaustive method spent a substantially longer time (about 75% more) compared to ParamILS of which the required computing times for all the test cases were the second longest (Table 3.4).

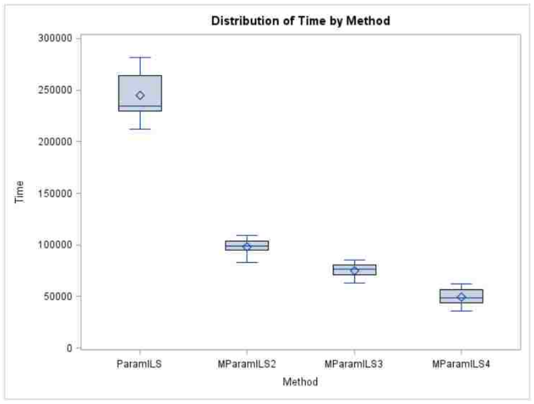
Table 3.4: Results of statistical significance tests (ANOVA test) on computing time of methods for each test case and the corresponding least-square means of computing time. The difference in computing time is significant if the p value is less than 0.05.

$P < .05$	**Exhaustive**	**ParamILS**	**ParamILS_2**	**ParamILS_3**	**ParamILS_4**	P-$Value$
Case I	951413.80	244822.90	98266.35	75390.81	49693.95	$< .0001$
Case II	921843.70	246613.80	99363.67	69789.65	54150.39	$< .0001$
Case III	866465.40	277902.80	104019.05	73155.85	56313.06	$< .0001$
Case IV	849819.00	191257.50	82751.51	73309.86	63709.44	$< .0001$
Case V	1018748.20	172384.10	53592.25	35869.23	34975.68	$< .0001$
Average	921658.02	226596.22	87598.57	65503.08	51768.50	
Reduced		75%	61%	25%	21%	

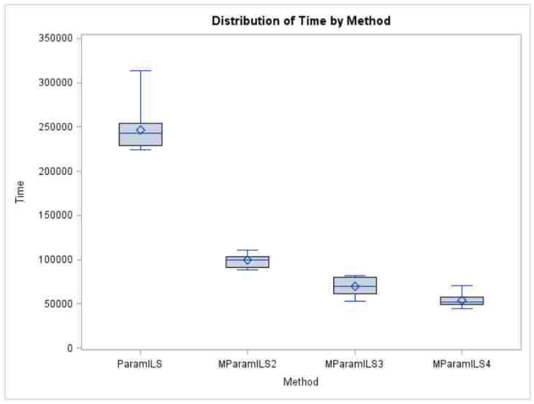
Second, comparisons of performances between ParamILS and MParamILSs were conducted to investigate how much MParamILSs can save time compared to ParamILS. According to the statistical results, the computing times were significantly different among the tested methods for all the test cases, and MParamILSs was always faster than ParamILS (Table 3.4). On average when compared to ParamILS, MParamILS2 reduced more than 60% of the computing time, MParamILS3 and MParamILS4 reduced more than 70% of the computing time. This demonstrates, for all tested problem cases, that MParamILSs required significantly shorter time than ParamILS. It can also be observed that MParamILS$_2$ required longer time than MParamILS$_3$, followed by MParamILS$_4$, which indicates that larger time savings were obtained with MParamILS using more coarser level problems.

Next, comparisons in terms of computing time variation were made among MParamILSs with ParamILS as the baseline method, in order to investigate the performance of MParamILS using a different number of coarser level problems. Figure 3.5 displays five box plots of computing time distributions of 10 runs of the methods on each test case. Results show that ParamILS had the largest computing time variations compared to MParamILSs for all the test cases. Moreover, time variations among MParamILSs were similar, and varied for different test cases. For example in Case III, MParamILS$_3$ had larger variation than MParamILS$_4$, whereas an opposite result can be observed in Case IV. This shows that 1) MParamILSs was more stable (less variation) than ParamILS in terms of computing time, and 2) time variations among MParamILSs were small and not related to the number of coarser level problems used.

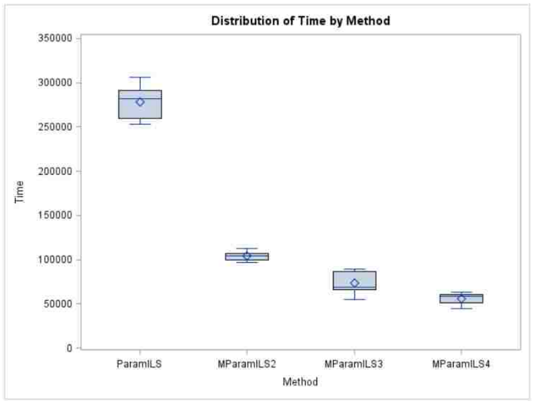
Finally, we conducted significance statistical analyses on both objective values and computing times considering all test cases (Diffograms in Figures 3.6 - 3.7). The vertical and horizontal axes in the Diffograms were marked with the least-squares means. Colored lines indicate if there is a significant difference between the two methods. When a colored line crosses the reference line (the diagonal dashed line), the least-squares means associated with the center point (joint point) of the line are



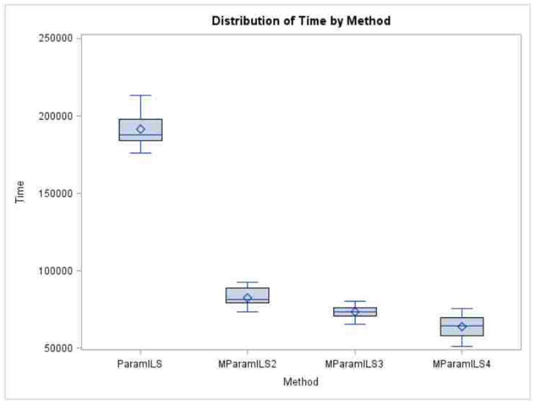
(a) Computing time distribution of Case I



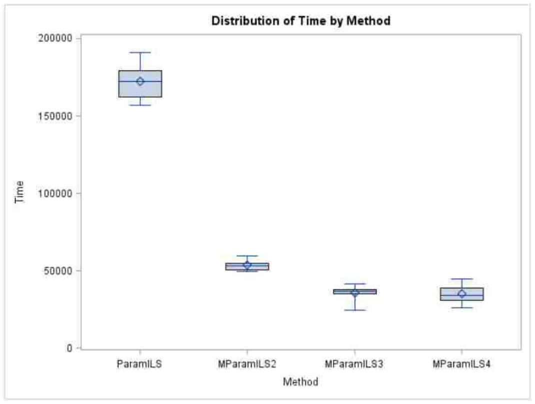
(b) Computing time distribution of Case II



(c) Computing time distribution of Case III



(d) Computing time distribution of Case IV



(e) Computing time distribution of Case V

Figure 3.5: Distribution of computing times of ParamILS and MParamILSs over 10 runs for each test case.

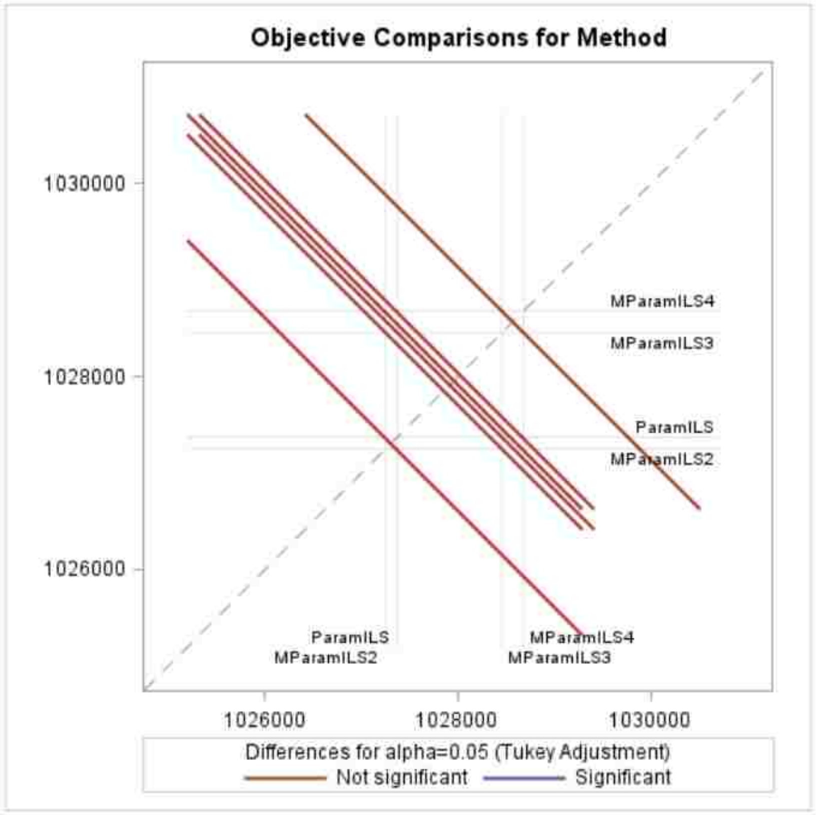


Figure 3.6: Diffograms illustrating the signaficance comparison on objective value among methods.

not significantly different. The results were consistent between any combination of methods that there were no significant differences in objective value (Figure 3.6). This indicates MParamILSs was able to match ParamILS in solution quality for all the test cases. On the other hand, results show that computing times were significantly different among ParamILS and MParamILSs (Figure 3.7). This difference was largest between ParamILS and each MParamILS and relatively small among MParamILSs. This corresponds with the results in Table 3.4 that MParamILSs required substantially less computing time than ParamILS for all the test cases.

3.5 Concluding Remarks

In this chapter, we present a multilevel parameter configuration scheme (MParamILS) to configure ACO-based algorithms for solving transportation problems modeled with

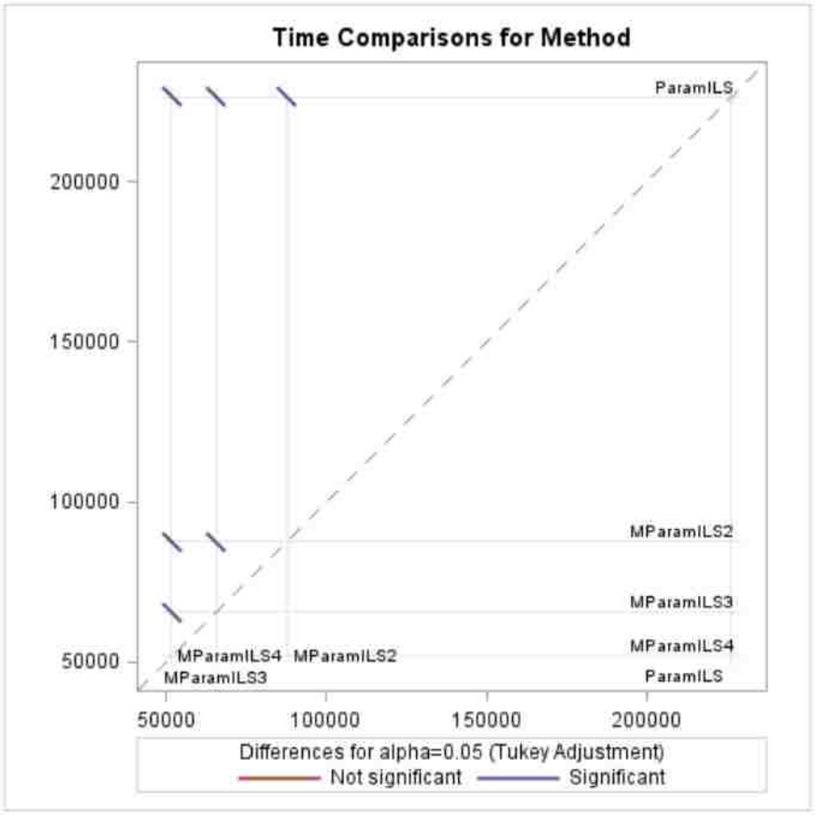


Figure 3.7: Diffograms illustrating the signaficance comparison on computing time among methods.

underlying weighted networks. It combines the graph coarsening procedure and a modified ParamILS procedure. MParamILS coarsens a large problem into smaller problems and uses the ParamILS to select high-quality parameter combinations from the smaller problems. The selected parameter combinations are then used to solve the original problem. As a result, the computing time is significantly reduced by evaluating a larger number of parameter combinations on less complex (smaller) problems and evaluating fewer number of parameter combinations on more complex (larger) problems. MParamILS was applied to five test cases and compared with ParamILS and the Exhaustive method in terms of solution quality and computing time. Results show that MParamILS was able to match solution qualities obtained by the other methods and reduce computing times substantially for all test cases.

Moreover, savings in computing time increased with the number of coarser level problems used in MParamILS. Consideration should be given to select an appropriate number of increasingly coarser problems used in MParamILS. A large number of coarser problems might result in low-quality parameter combinations as the structure of the coarsest problem might differ too much from the original problem. On the other hand, a small number of coarser problems can result in MParamILS requiring a longer computing time. In addition, the graph coarsening procedure used in the experiments produces coarser level graphs by contracting finer level edges with large attribute weights. This is an important property for solving minimization problems. However, depending on different applications, weights on contracted edges can be calculated differently. Analogously, parameter configurators other than ParamILM in the proposed multilevel scheme can be considered. Lastly, although we applied MParamILS to configure ACO algorithms, it can be extended to any parameterized algorithms.

4 A Multilevel ACO Approach for Solving Constrained Forest Transportation Planning Problems

4.1 Introduction

As demonstrated in the last chapter, a multilevel approach was used successfully for the parameter configuration problem. In fact, as a general solution strategy, multilevel schemes have been used for many years and applied to several problem areas [42, 54, 60, 69, 51] where solution quality can benefit from having a relatively high-quality initial solution that can be computed inexpensively on a lower level scale. These schemes have proven to be efficient when solving discrete NP-hard problems with a finite but exponential number of problem component combinations [86, 57, 85].

In this chapter, we extend the study from the previous work and develop a multilevel ACO approach (MLACO) to solve the CFTPP. The MLACO not only considers automatically configuring the parameter values to achieve best algorithm performance, it also attempts to reduce computing time by using the multilevel computation. Specifically, on one hand, MLACO integrates the multilevel ParamILS technique to automatically configure the ACO parameters; on another hand, the best solution obtained for a coarser level problem is used to help search for best solution for a finer level problem quickly. The idea is to reduce the computational cost, which is expensive for using exact methods to solve the original problem of large size, by solving a set of smaller problems hierarchically, starting from the smallest one to the largest one. In particular, three objectives to be achieved in this chapter with the MLACO are to demonstrate that, for the problem instances on which it is tested, the MLACO can either accelerate the solution convergence rate or improve the solution qualities; explain the underlying process and why the MLACO can enhance performance compared to other methods; and extract generic properties of the MLACO such as its advantages and limitations and suggest how it might be applied to other optimization problems. Ultimately, the MLACO presented in this study can serve as a framework that can be easily expanded to develop tools that incorporate other factors into forest transportation planning and provide forest managers with environment-friendly road network alternatives to help them make informed decisions.

4.2 Multilevel Scheme

Typically, multilevel schemes solve a large problem using a set of increasingly smaller problems through a series of solution refinements. These smaller problems are obtained by successively applying a coarsening process to the original problem. The coarsening process creates a hierarchy of problem instances where a given coarser level problem is always smaller than the previous finer level problem. The solution is computed for a given coarser level problem and then transformed into a solution for the next finer level problem until the original problem is solved.

An illustration of this process with ACO algorithm is presented in Figure 4.1 where a finer problem is coarsened into a coarser problem. The coarsening process is believed to have preserved certain properties of the finer level problem in a sense that a coarser level solution can be mapped into a set of finer level problem components, and vice versa. An ACO algorithm is applied to the coarser level problem of which the computational time is observably lower than that for the finer level problem. The obtained solution, then, is interpolated into a set of components of the finer level problem, which can be used to aid the ACO algorithm to find good solutions for the finer level problem faster. Based on this idea, the MLACO is developed in this chapter. To avoid confusion and assure the clarity, we define the following terminologies that are used through out this thesis.

coarser/finer level problems: for a set of increasingly coarser level problems $\Pi = \Pi_0, \Pi_1, \ldots, \Pi_N$, where Π_0 is the original problem and Π_N is the coarsest level problem. If a problem $\Pi_i \in \Pi$ is referred to as a coarser level problem, then the problem $\Pi_{i-1} \in \Pi$ is referred to as its finer level problem and vice versa.

interpolated components: for a coarser level problem Π_i and its finer level problem Π_{i-1}, the interpolated components refer to the finer level problem components obtained by interpolating the solution found for the coarser level problem.

mapping process: for a coarser level problem Π_i and its finer level problem Π_{i-1}, the mapping process interpolates the coarser level solution to obtain the interpolated components.

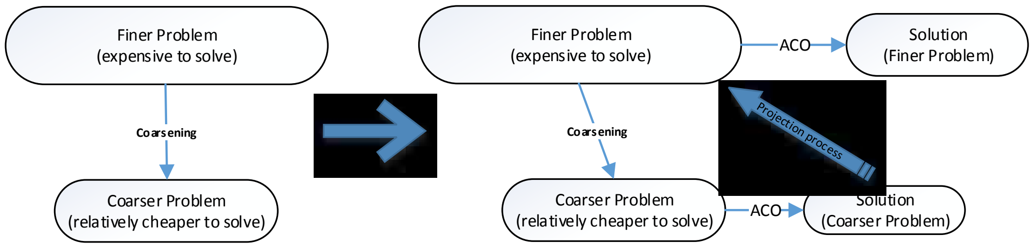


Figure 4.1: Diagram illustrating a multilevel scheme where a finer level problem is coarsened to a coarser level problem (Left hand side picture). An ACO algorithm solves the coarser level problem first and the obtained solution is used to help find high-quality solutions for the finer level problem (right hand side picture).

4.3 Multilevel Approach

4.3.1 Obtaining Coarser Level Problems

As the networks of the CFTPP are weighted graphs, graph coarsening techniques can be naturally applied to produce coarser level problems. Similar to Chapter 3, the same graph coarsening algorithm used for the Multilevel ParamILS is used for the MLACO. The coarsening algorithm generates coarser level problems by finding a set of matching edges (matching) from the finer level problem, in which no two edges are incident to the same node, and then collapsing the matching edges and aggregating the incident nodes (Figure 4.2). This process is applied to each subsequently obtained coarser level problem, resulting in a set of increasingly smaller problems.

For the sake of clarity, in the following section, the nodes on the collapsed matching edges are referred to as "matching nodes", the coarser level nodes obtained by aggregating the matching nodes are referred to as “aggregated nodes”, and the coarser level nodes that are the same as its finer level are referred to as “unaggregated nodes”.

4.3.2 Underlying ACO

The underlying ACO algorithm (Algorithm 7), referred to as “GuidedACO” in this study, is designed to search for solutions for a finer level problem guided by the interpolated solution components that are mapped from the computed solution of the coarser level problem. In the implementation, the GuidedACO is used for all coarser level problems and the original problem with or without the guidance from the interpolated solution components. The edge case here is when applying the GuidedACO to the coarsest level problem where no solution components are available.

The GuidedACO is subsequently applied to the coarser level problems, from the smallest to the largest, to the original problem (as shown in Figure 4.2). The best solutions found for the coarser level problems are mapped to the interpolated components, which are used for the pheromone update to help the underlying ACO algorithm find high-quality solutions for the finer level problems.

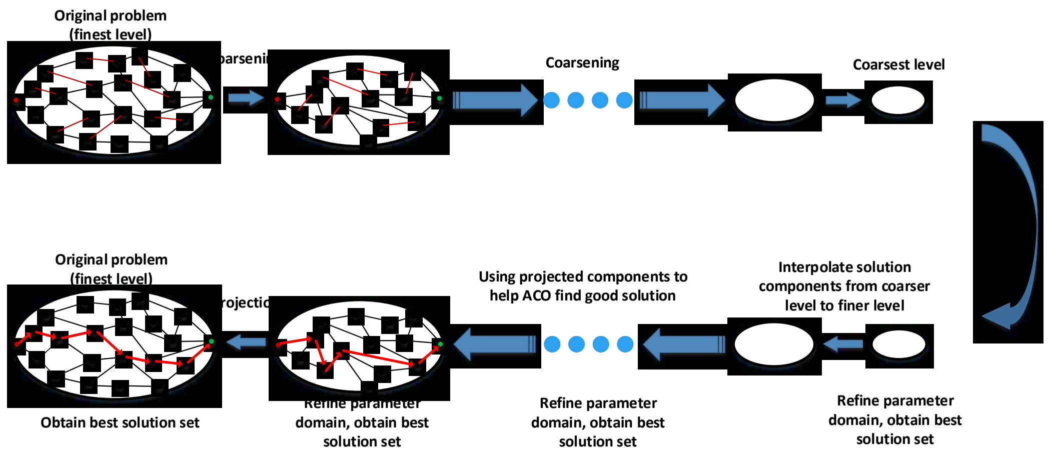

Original problem (finest level)
oarseni
Coarsening
Coarsest level
Original problem (finest level)
rojectio
Using projected components to help ACO find good solution
Interpolate solution components from coarser level to finer level
Obtain best solution set
Refine parameter domain, obtain best solution set
Refine parameter domain, obtain best solution set
Refine parameter domain, obtain best solution set

Figure 4.2: The upper left corner of the diagram shows the original problem network where the objective is to find a route from an origin (red node) to a destination (green node). The matching edges (red edges) are contracted to produce coarser level problems. The original problem is coarsened into a set of increasingly coarser problems.

Algorithm 7 GuidedACO(Π, $S_{component}$, $\vec{\theta}$)

```
Initialization(HasFeasible = false; BestSolus ⟸ $\phi$.)
Output(BestSolus)// equivalent best solution set
begin
    Set_Phero($\Pi$,$S_{component}$)
    while stop conditions are not satisfied do
        CurSolu ⟸ Ants obtain solution for $\Pi$ with $\vec{\theta}$
        /* Searching best feasible solutions                          */
        if CurSolu is feasible AND HasFeasible then
            if CurSolu is better than CurBestSolu then
                Set_Phero($\Pi$,$S_{component}$)
                CurBestSolu ⟸ CurSolu
                BestSolus ⟸ $\phi$ /* remove all previous best solutions  */
                BestSolus ⟸ CurBestSolu
            end
            else if CurSolu is equivalent to CurBestSolu then
                BestSolus ⟸ BestSolus + CurSolu
            end
            Phero_Update($\Pi$,CurBestSolu)
        end
        /* Searching feasible solutions                               */
        if $\neg$ HasFeasible then
            if CurSolu is feasible then
                CurBestSolu ⟸ CurSolu
                HasFeasible ⟸ true
                continue

            end
            if CurSolu is better than CurBestSolu then
                CurBestSolu ⟸ CurSolu
                Phero_Update($\Pi$,CurBestSolu)

            end
        end
    end
end
```

Similarly to the ACO algorithm described in Chapter 2, the GuidedACO is also designed specifically for the CFTPP. It proceeds with first stage where existence of feasible solution is determined, and then with second stage where the solution quality is iteratively improved. Compared to the ACO in Chapter 2, in guided ACO, the interpolated components are initialized with a larger amount of pheromone. This is to increase their chances of being selected for constructing solutions because they are expected to form the best solution with high probability due to the fact that these problem components are mapped using the best solutions found for coarser level problems.

Algorithm 8 Set_Phero(Π, $S_{component}$)

```
begin
    foreach $C_i \in \Pi$ do
        $\tau_{C_i} \Longleftarrow$ Initial_Pheromone
        if $C_i \in S_{component}$ then
            $\tau_{C_i} \Longleftarrow (\frac{\eta_{C_i}}{\max(\eta)})^{-1} \times$ Update_Factor $\times \tau_{C_i}$
        end
    end
end
```

In trial experiments, we discovered that, the GuidedACO usually can find several solutions that have the same objective value. This observation shows that the possibility exists that optimal solution may not be unique, which is actually intuitive for constrained optimization problems where two solutions can be equivalent in the following two situations: 1) two feasible solutions with the same objective function value but different constraint values; 2) two feasible solutions with the same objective function and constraint values but differ in one or more solution component combinations. In practice, the former case is expected to occur more frequently than the latter case because seldom two solutions with different constructions have identical objective and constraint values.

In this study, it is preferable that all the optimal solutions can be obtained, mapped into interpolated components and be used for the pheromone update and solution construction guidance routines (as implemented in Algorithm 7). To this end, we give a definition for solutions that are equivalent:

Definition 3. *For a constraint optimization problem, two feasible solutions S_i and S_j are equivalent, denoted as $S_i \parallel S_j$, if their objective function values are equal:*

$Obj(S_i) = Obj(S_j)$, and there is at least one solution component C such that $C \in S_i \wedge C \notin S_j$ or vise versa.

Therefore, pheromone values are updated in both stages using the current best solution set, where all the best-found-so-far solutions are used. The GuidedACO increases pheromone values for the current best solution components by a small amount and decreases pheromone values for other problem components by a small fraction. If a solution is found to be better than the current best solution, pheromone values are first reset according to the interpolated components (Algorithm 8) and then updated with the new current best solution.

The reason for resetting the pheromone values is to prevent the premature convergence to a local optima. Different from the ordinary problem components on which pheromone values are equally reset with an initial pheromone amount, pheromone values for the interpolated components are set to be the inverse of the normalized heuristic value ($\eta_{C_i} / \max \eta$) multiplied by an updating factor (Update_Factor - discussed later) (shown in (Algorithm 8)). This formulation ensures that the interpolated components with smaller heuristic values (such as costs) receive a larger amount of pheromone, whereas interpolated components with larger heuristic values receive less pheromone. The GuidedACO stops searching for better solutions when the set of equivalent best solutions reach stagnation.

4.3.3 Multilevel ACO

The MLACO is shown in Algorithm 9 that applies the GuidedACO to solve the CFTPP through a process of solution, parameter search and refinement on the coarser level problems. The first step is to generate a set of increasingly coarser level problems of which the coarsening information including contracted edges and aggregated weights is tracked and recored. This information is used when mapping the coarser level solutions to the finer level.

More concretely, the coarsening information is defined as follows. Let a problem Π_0 be coarsened into coarser level problems $\Pi_1, \Pi_2, \ldots, \Pi_N$ and $k = k_0, k_1, \ldots, k_{N-1}$ be the coarsening information for all coarser level problems, then $k_i \in k$ is defined as a set of triplets:

$$k_i = (k_i | (v_{a_1}^{\Pi_{i-1}}, v_{b_1}^{\Pi_{i-1}}, v_{c_1}^{\Pi_i}), (v_{a_2}^{\Pi_{i-1}}, v_{b_2}^{\Pi_{i-1}}, v_{c_2}^{\Pi_i}), \ldots, (v_{a_d}^{\Pi_{i-1}}, v_{b_d}^{\Pi_{i-1}}, v_{c_d}^{\Pi_i})) \tag{4.1}$$

where d is the number of the aggregated nodes and each triplet contains two finer level matching nodes v_a, v_b and the resulted coarser level aggregated node v_c.

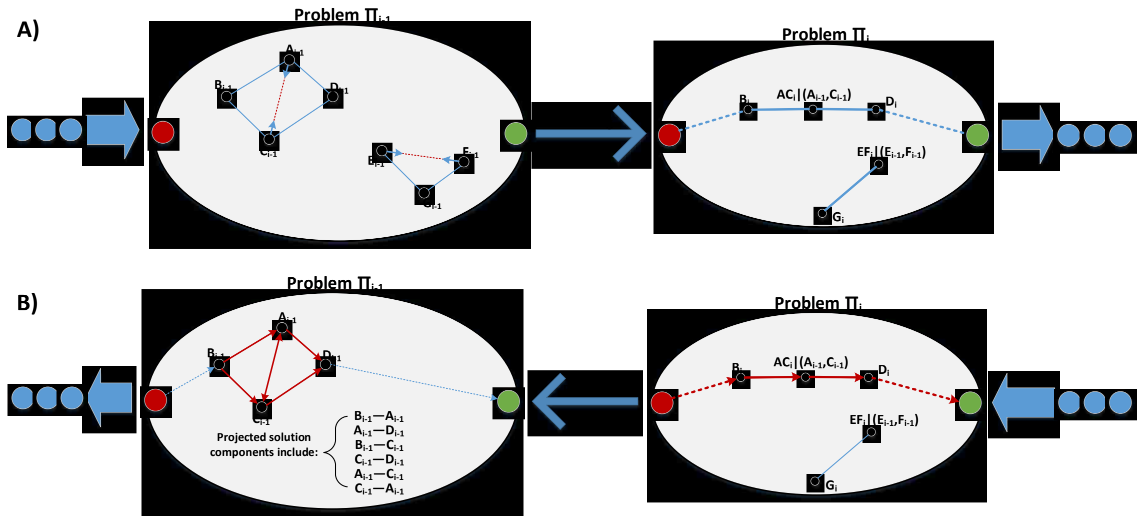


Figure 4.3: Diagram illustrating the problem coarsening phase (A), and the mapping process in the solution refinement phase (B).

In general, the interpolated components are the finer level problem components contracted and aggregated to produce the coarser level solution components. With the coarsening information, the best found solutions for a coarser level problem can be easily mapped to produce the interpolated components, which is described in Mapping procedure in Algorithm 9. The GuidedACO uses the interpolated components to construct solutions for the finer level problem and iteratively refines and improves the solution quality. The obtained best finer level problem solutions, in turn, are used again with the coarsening information to map a new set of interpolated components that are used in the GuidedACO to solve the next finer level problem. The MLACO

Algorithm 9 MLACO

```
Input(Coarser problems $D \Leftrightarrow \Pi_0, \Pi_1, \ldots, \Pi_N$ where $\Pi_0$ is the original problem;
Coarse information $k = k_0, k_1, \ldots, k_{N-1}$; Parameter combination domain $\Theta$.)
begin
    $S \Longleftarrow \phi$ // $S$ is an empty solution set
    Best_Solu $\Longleftarrow$ Evaluate($\Theta$, $\Pi_N$, $S$)
    for $i = N - 1$ to 0 do
        $S \Longleftarrow$ Mapping($\Pi_i, \Pi_{i-1}, k_i,$ Best_Solu)
        Best_Solu $\Longleftarrow$ Evaluate($\Theta$, $\Pi_i$, $S$)
    end
    Return(Best_Solu)
end

/* Obtain solution with given problems                                  */
Procedure: Evaluate($\Theta$, $\Pi$, $S$)
begin
    foreach $\vec{\theta} \in \Theta$ do
        LocalBestSolu $\Longleftarrow$ running GuidedACO($\Pi$, $S$, $\vec{\theta}$)
        if solutions in LocalBestSolu better than that of GlobalBestSolu then
            GlobalBestSolu $\Longleftarrow \phi$
            GlobalBestSolu $\Longleftarrow$ LocalBestSolu
        end
        else if solutions in LocalBestSolu equivalent to that of GlobalBestSolu
        then
            GlobalBestSolu $\Longleftarrow$ GlobalBestSolu + LocalBestSolu
        end
        if solutions in LocalBestSolu have very low qualities then
            remove $\vec{\theta}$ from $\Theta$
        end
    end
    Return(GlobalBestSolu)
end

/* Obtain the interpolated components                                   */
Procedure: Mapping($\Pi_i, \Pi_{i-1}, k_i,$ Best_Solu$_{\Pi_i}$)
begin
    Interpolated_Comps $\Longleftarrow \phi$
    foreach solution $S_{\Pi_i} \in$ Best_Solu$_{\Pi_i}$ do
        foreach component $C_{\Pi_i} \in S_{\Pi_i}$ do
            Interpolated_Comps $\Longleftarrow$ interpolated components $SC_{\Pi_{i-1}} \subset \Pi_{i-1}$ using $k_i$
            foreach component $C_{\Pi_{i-1}} \in SC_{\Pi_{i-1}}$ do
                Calculate its Update_Factor
                add $C_{\Pi_{i-1}}$ to Interpolated_Comps
            end
        end
    end
    Return(Interpolated_Comps)
end
```

repeats the same steps subsequently from coarser to finer level until the finest level problem (original) is solved.

An example of the coarsening and solution mapping processes is illustrated in Figure 4.3, where problem Π_{i-1} is coarsened into Π_i in the coarsening phase and the solution found for Π_i is mapped to Π_{i-1} for the interpolated problem components. In the problem coarsening phase, the problem Π_{i-1} is coarsened to obtain coarser level problem Π_i. In the solution refinement phase, after mapping the solution components e_{B_i,AC_i} and e_{AC_i,D_i} in Π_i, the possible solution combinations that connect from node B_{i-1} to node D_{i-1} are $(B_{i-1} \rightarrow A_{i-1} \rightarrow D_{i-1})$, $(B_{i-1} \rightarrow C_{i-1} \rightarrow D_{i-1})$, $(B_{i-1} \rightarrow A_{i-1} \rightarrow C_{i-1} \rightarrow D_{i-1})$, $(B_{i-1} \rightarrow C_{i-1} \rightarrow A_{i-1} \rightarrow D_{i-1})$. The interpolated components are edges appear in all the possible solution combinations: $e_{B_{i-1},A_{i-1}}$, $e_{A_{i-1},D_{i-1}}$, $e_{B_{i-1},C_{i-1}}$, $e_{C_{i-1},D_{i-1}}$, $e_{A_{i-1},C_{i-1}}$, $e_{C_{i-1},A_{i-1}}$.

Furthermore, while the interpolated components are used as initial solution and refined iteratively to obtain better quality solutions, the MLACO also attempts to achieve maximum performance by automatically selecting high-quality parameter combinations (Evaluate procedure in Algorithm 9) using the coarser level problems. When a new solution is obtained, it is compared to the solutions in the best solution set obtained from previous iterations. If the new solution is equivalent or better, it is included in the best solution set and the associated parameter combination is considered high-quality. Otherwise, it is discarded along with the parameter combination used. By identifying and discarding low-quality parameters, the overall computing time of the MLACO is reduced because of the better ACO performance due to the selected high-quality parameter combinations for the final level problem. This approach is similar to the refinement process for a multilevel graph partitioning algorithm developed in [52] where solutions are refined from coarser to finer levels.

4.3.4 Calculating Update Factor

When setting pheromone values in a finer level problem, different interpolated components are given different pheromone amounts. Each interpolated component is associated with an update factor that reflects its importance in terms of forming a good quality solution based on the interpolated coarser level solutions. Depending on the values of the update factors, the amount of pheromone is assigned by giving a larger amount of pheromone to interpolated components associated with larger update factors and vice versa (Algorithm 8).

For a coarser level problem, after the GuidedACO is applied, a number of equivalent solutions are found. Each of the equivalent solutions is then mapped to obtain a set of interpolated components. As one solution component can exist in several equivalent coarser level solutions, an interpolated component might also be obtained more than once. The number of times that an interpolated component is obtained by the mapping process indicates how frequently the corresponding coarser level component is used as the solution component and can be considered as a metric to calculate the associated update factor.

An example of the calculation of the update factor is illustrated in Figure 4.4 where the solution set of a coarser level problem contains two routes that pass through edges $e_{a,b}$, $e_{b,c}$, and $e_{b,d}$ (the 1^{st} and 2^{nd} routes in the left picture in Figure 4.4.A). Each edge is associated with a number indicating the number of times the edge is used in the solution set (i.e., the number for the edge $e_{a,b}$ is two since it is used twice). After the mapping process, the nodes a and c stay the same and node b is replaced with finer level nodes b_1 and b_2 to produce the interpolated components because b is an aggregated node (right picture in Figure 4.4.A). If an interpolated component is incident to one or two unaggregated nodes (such as e_{a,b_1} and $e_{b_1,c}$), it is assigned the number associated with the coarser level solution component incident to the same nodes. Otherwise, zero is assigned to the interpolated component incident only to

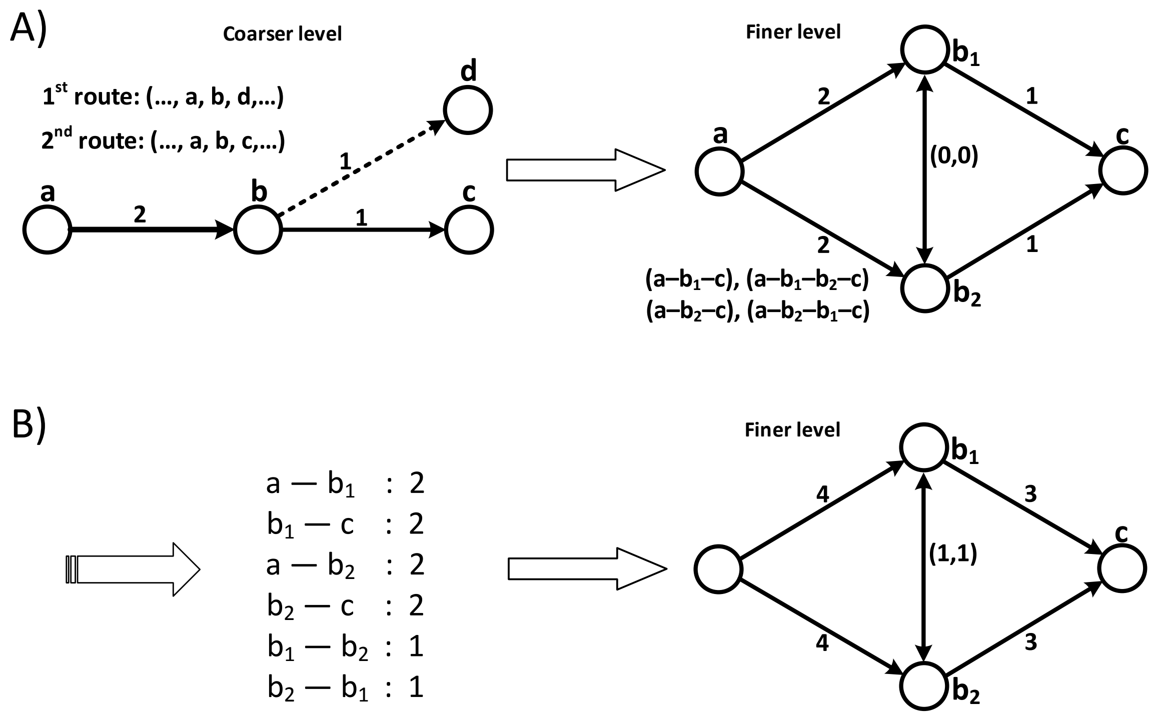


Figure 4.4: Calculating update factor values for interpolated solution components.

the finer level nodes (i.e., e_{b_1,b_2} and e_{b_2,b_1}). As the coarser level solution route goes from nodes a to c, the interpolated components are expected to also connect these two nodes, which results in four possible paths: $(a \to b_1 \to c)$, $(a \to b_2 \to c)$, $(a \to b_1 \to b_2 \to c)$, $(a \to b_2 \to b_1 \to c)$ (left picture in Figure 4.4.B)). Counting the solution component occurrences in the four paths, e_{a,b_1}, $e_{b_1,c}$, e_{a,b_2}, $e_{b_2,c}$ appear twice and e_{b_1,b_2}, e_{b_2,b_1} appear once. Then the update factor values are calculated by adding these occurrences to the existing assigned numbers (right picture in Figure 4.4.B).

4.4 Experimental Studies

4.4.1 Experiment Setup

The algorithms and procedures presented in this study were implemented using C++ and Java. The computer nodes of Lipscomb High Performance Computing Cluster (HPC) were used to execute the programs. We compared the performance of the MLACO with other three methods: an Exhaustive method, a ParamILS method

[46], and the MIP.

The configured parameters included α, β, ρ and their values were confined to [0,1] with a pace of 0.05. The Exhaustive method ran the GuidedACO for each parameter combination in the parameter combination domain 10 times and selected the feasible solutions with the best objective value. The parameter combination domain was divided into 20 partitions and the Exhaustive method was applied to each of the partitions simultaneously. The computing time of the Exhaustive method was calculated by adding up times required for all partitions (which is the same setting as in the Chapter 3). The ParamILS method configured parameter settings for the GuidedACO based on solution quality. It iteratively permuted parameter values and ran the GuidedACO until it could not find a better quality solution. The MIP formulated the CFTPP using the formulations presented in [61] and was implemented using Java and the CPLEX 12.5 Callable Library (ILOG Inc. 2007) with the default parameter setting [18]. Because MIP can take impractically long computing time, we set its maximum running time to 864,000 seconds (10 days). CPLEX was forced to stop and report the best solution found when the computing time exceeded the maximum running time.

The MLACO approach was set to use a set of three increasingly coarser level problems: Level-1, Level-2 and Level-3. After initial test runs, three coarsening levels were selected to balance solution quality and computation time as well as preserving the properties of the original problem. This resulted in the coarsest level problem to be about one eighth of the size of the original problem (problem size was reduced by about half from a given level to its coarser level).

For the experimental data, we used 10 network instances containing 20 CFTPPs used in Lin, et al., [61]. These problems were designed as medium-scale, grid-shaped hypothetical networks. The hypothetical grid-shaped network was used because it resembles real-world FTPPs, providing a good test case for algorithm performance.

In addition, the medium-scale size allows solving the instances a large number of times within reasonable time to conduct the parameter search.

4.4.2 Experimental Results

The experiments were conducted to test performance in terms of solution quality and computing time. All methods were able to find feasible solutions for all problems (obtained sediment amounts below the constraint values as shown in Figure 4.5). As MIP can solve CFTPPs optimally, its solutions were used to benchmark solution qualities achieved by the Exhaustive, ParamILS and MLACO methods (Figure 4.6). The three approximation methods were able to match MIP solutions for most of the problems, except for problems 3, 16, 19 and 20 where solution qualities were slightly worse. In the worst case (problem 16), the MLACO approach was able to outperform the ParamILS method (closer to the MIP solution) and was slightly worse than the Exhaustive method. In addition, univariate statistical tests (Table 4.1, where two methods are significantly different if Sig. < 0.05 (0.95 confidence interval)) show that there were no significant differences between MIP and any other tested methods for solution quality, indicating that on average the Exhaustive, ParamILS and MLACO are expected to obtain near-optimal solutions for the tested problems. This also indicates that the MLACO approach was able to self-configure properly and obtained competitive high-quality solutions compared to other methods.

In terms of computing time, results show significant differences among the tested methods (Table 4.1). As expected, on average the Exhaustive method required the longest (184.35 days) compared to other methods (Figure 4.7). The ParamILS required only 12.27 days (about 6.6%), MIP 6 days (3.2%), and the MLACO only required an average time of 1.28 days (Table 4.2). For the worst case scenario, the Exhaustive method required 227 days, ParamILS 19 days, MIP 10 days and MLACO 4 days, while for the best case scenario, the Exhaustive method required 137 days, ParamILS 6 days, and both MIP and MLACO less than one day. Although MIP

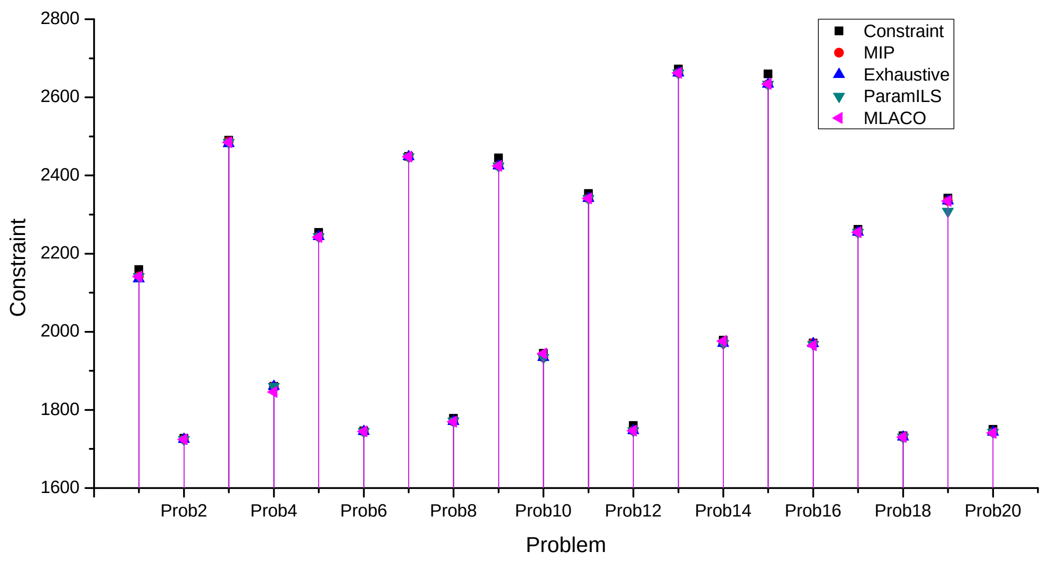


Figure 4.5: Sediment values obtained by each method for the 20 problems.

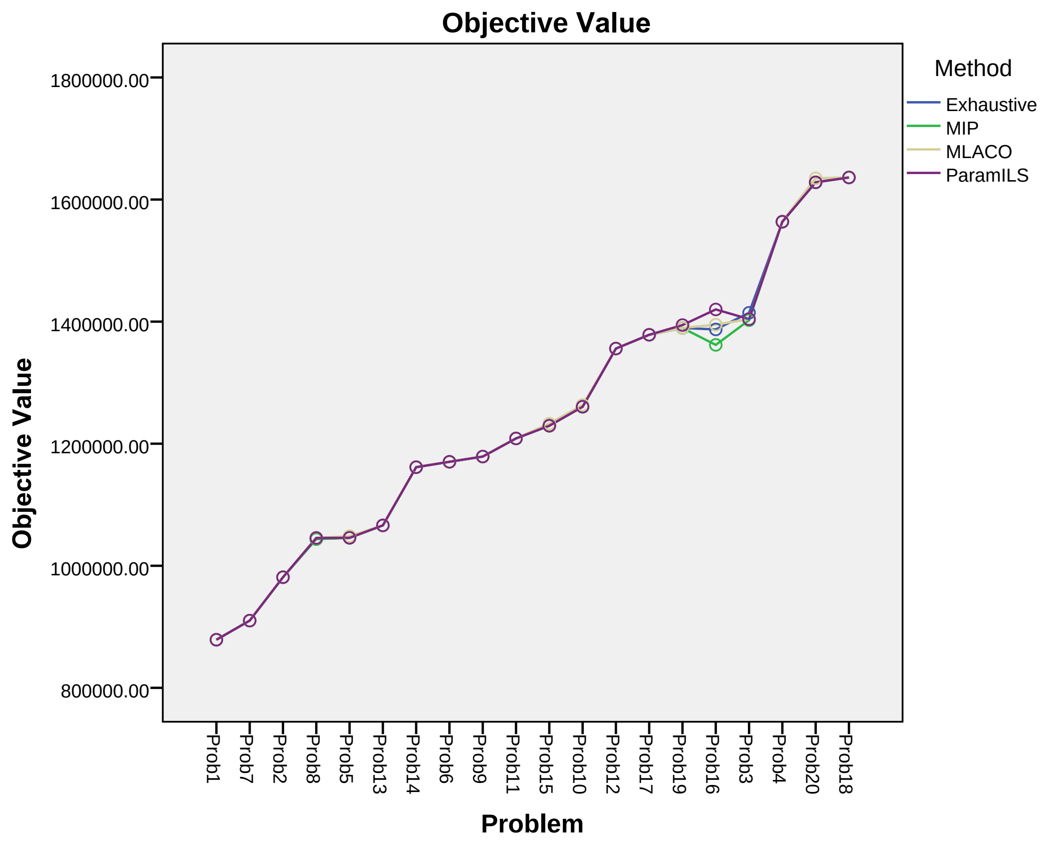


Figure 4.6: Solution quality comparisons between Exhaustive, ParamILS, and MLACO with respect to MIP.

solvers typically require relatively long computing time, on average the MIP spent less time to solve all problems than the Exhaustive and ParamILS methods. This

Table 4.1: Statistical test (Univariate Test with Tukey Post Hoc) of objective function values and computing time for the tested methods. The Exhaustive method was not included in the statistical test for computing time as the differences between it and other methods were evidently significant shown in Figure 4.7 and Table 4.2

Objective function value			**Time**		
(I) Method	(J) Method	Sig.	(I) Method	(J) Method	Sig.
Exhaustive	MIP	0.707	MIP	MLACO	< 0.0001
	MLACO	0.969		ParamILS	< 0.0001
	ParamILS	0.845			
MIP	Exhaustive	0.707	MLACO	MIP	< 0.0001
	MLACO	0.43		ParamILS	< 0.0001
	ParamILS	0.243			
MLACO	Exhaustive	0.969	ParamILS	MIP	< 0.0001
	MIP	0.43		MLACO	< 0.0001
	ParamILS	0.983			
ParamILS	Exhaustive	0.845			
	MIP	0.243			
	MLACO	0.983			

resulted from MIP being able to finish quickly for some problems (Figure 4.8), which reduced overall computing time. Variation in computing time among problems was largest for the Exhaustive search (Std Dev: 26.7 days) followed by MIP (Std Dev: 4.8 days) and ParamILS (Std Dev: 3.2 days). In contrast, the MLACO approach showed the most stable computing times (Std Dev: 1.05 days). Also, computing time for MIP was highly skewed by some problems (1,3,5,7,11,13,15,17) that were solved very quickly. For other problems, MIP spent exactly 10 days indicating it was not able to find the optimal solutions and was forced to stop after the maximum running time. As suggested on the CPLEX reference manual [45], MIP performance is affected by its parameter setting. However, it is impractical to configure all 135 MIP parameters driving the search process.

We also analyzed the performance of the MLACO approach by examining the parameter domain size and computing time for each level of the coarser problems. The minimum computing time required for all problems at each level was related to

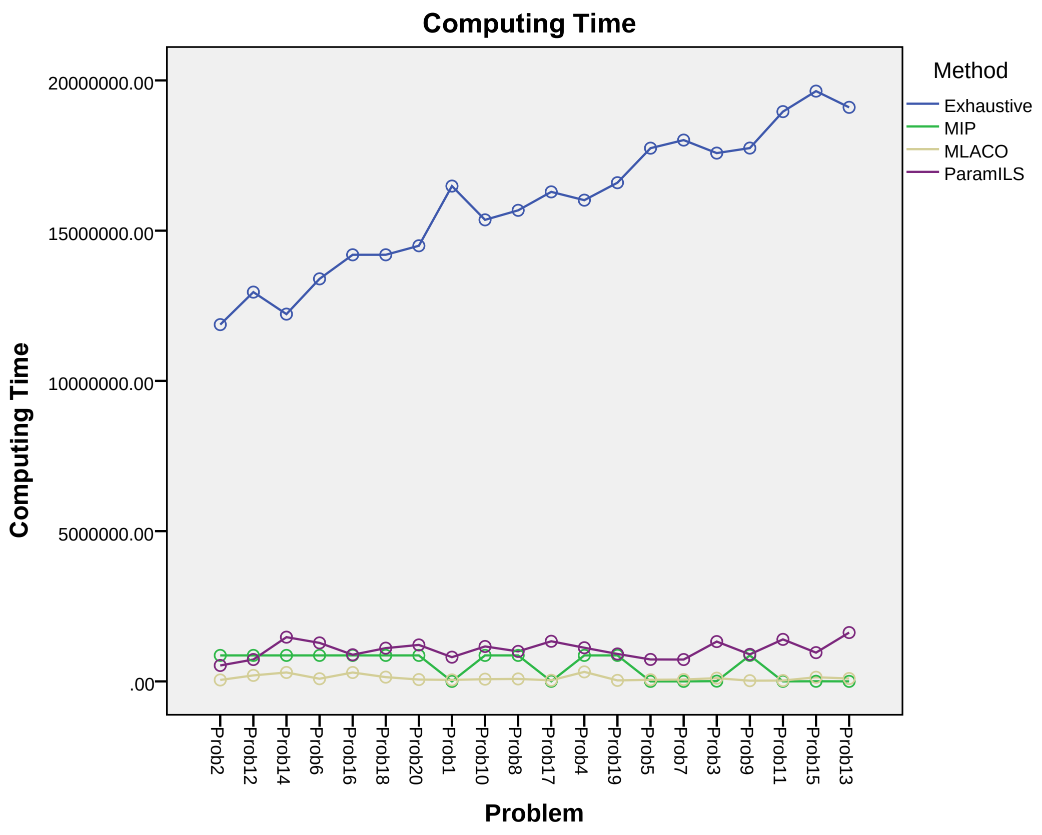


Figure 4.7: Computing time comparisons between Exhaustive, MIP, ParamILS, and MLACO.

Table 4.2: Summary of computing time required to solve all problems for the Exhaustive, ParamILS, MIP, and MLACO.

(Seconds)	**Exhaustive**	**ParamILS**	**MIP**	**MLACO**
Min(days)	137	6	0	0
Max(days)	227	19	10	4
Median(days)	183	11	10	1
Average(days)	184.352	12.274	6.009	1.288
Stand. Dev(days)	26.079	3.280	4.888	1.053

the number of parameter combinations in the domain, which was largest for Level-3 and smallest for Level-0, 8,000 and 6 respectively (Table 4.3). On the other hand, the maximum computing time was largest for Level-0 (280,291 seconds) with the least number of parameter combinations to evaluate (176 parameter combinations for Problem 4) and was relatively small for Level-3 (18,796 seconds) and Level-2

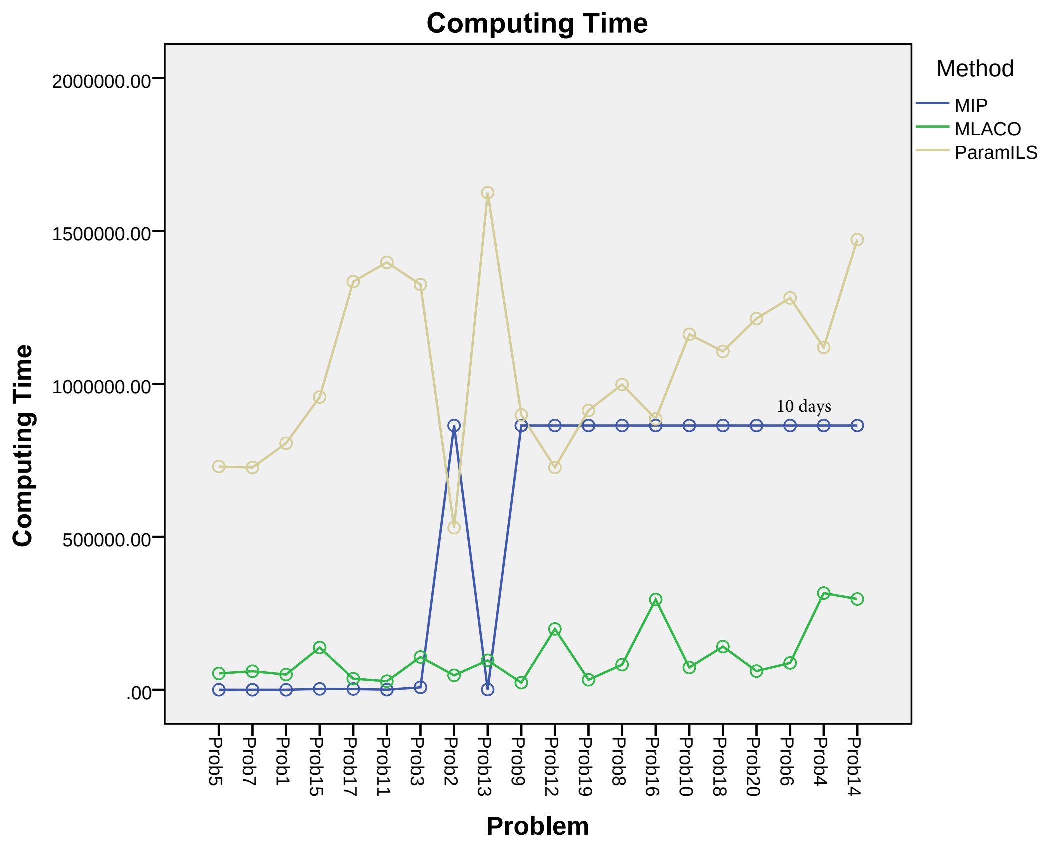


Figure 4.8: Computing time comparisons between MIP, ParamILS, and MLACO.

Table 4.3: Time spent and parameter combination domain size changes for each coarse level problem used in MLACO. Level-0 denotes for the original problem.

	Time(Seconds)				Size of parameter combination domain			
	Level-3	Level-2	Level-1	Level-0	Level-3	Level-2	Level-1	Level-0
Min	7,978	3,436	3,143	512	8,000	889	63	6
Max	18,796	13,725	46,478	280,291	8,000	2,162	564	176
Median	9,310	7,301	9,284	54,112	8,000	1,239	235	37
Average	10,076	7,609	11,243	82,367	8,000	1,275	270	58
Stand. Dev	2,337.848	2,339.627	9,542.240	86,383.311	0	275.652	142.449	53.721

(13,725 seconds) for which a larger number of parameter combinations were evaluated (8,000 and 2,162 parameter combinations). In general, computing time increased more quickly for Level-0 problem even though the number of parameter combinations was smaller compared to other levels (Figure 4.9). This might indicate that computing time was more susceptible to problem complexity than the number of the parameter

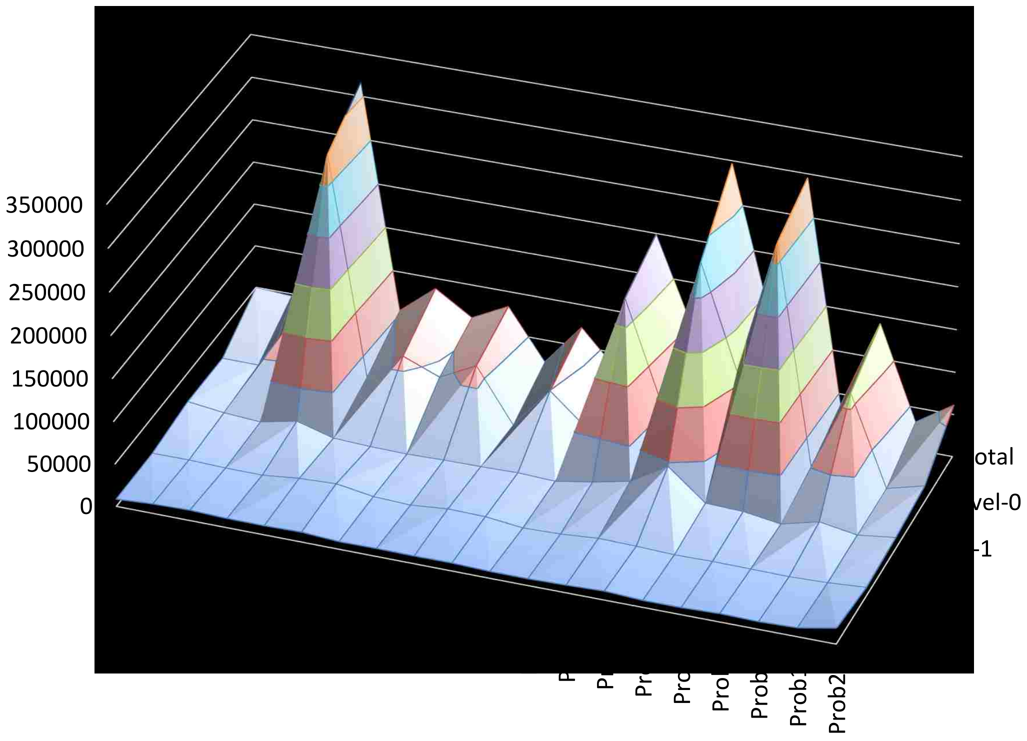


Figure 4.9: Time spent on each level of problems by MLACO.

combinations to evaluate. Lastly, the size of parameter combination domain decreased significantly (Figure 4.10) from Level-3 (8,000 on average) to Level-2 (1,275) and continued to decrease to Level-1 (270) and to Level-0 (58). This result shows that as problem complexity increased, the number of high-quality parameter combinations for the problem was reduced, which might indicate that the performance of the MLACO approach was less sensitive to its parameter setting for a coarser level problem than that for a finer level problem.

4.5 Concluding Remarks

In this chapter, a multilevel ACO approach (MLACO) is developed to solve the CFTPP. The approach coarsens a problem into a set of increasingly coarser level problems and uses an underlying ACO algorithm (GuidedACO) to solve the problem from the coarsest level to the finest level. The best solutions found for coarser level

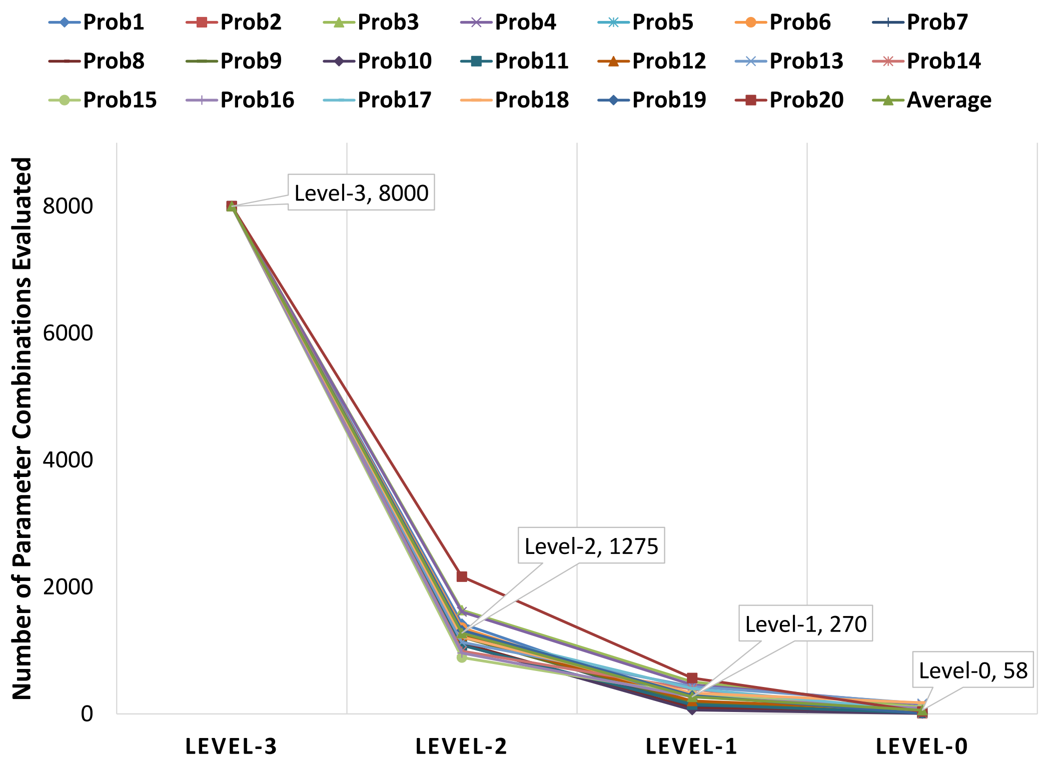


Figure 4.10: Parameter combination domain size changes at each level of problems for MLACO.

problems are used to obtain the interpolated components to help the GuidedACO improve solution quality and reduce computing time. In addition, the MLACO approach configures its parameter setting by consecutively eliminating low-quality parameter combinations from a predefined parameter combination domain from coarser level to finer level problems. The performance of the MLACO approach was tested on 20 problem instances with similar topology to real world problems and solutions were compared with those obtained from the Exhaustive, ParamILS, and MIP methods. Experimental results demonstrate that MLACO approach was able to obtain high-quality solutions for all tested problems with significant reduction in computing time.

The design of the underlying ACO algorithm includes two stages to: first find feasible solution and then improve solution quality. Although only one constraint was considered, the algorithm design can be easily modified to incorporate any number of constraints. The solution refinement method implemented in our algorithm offers a good potential for applications to other large-scale and complex optimization problems with underlaying networks. The parameter configuration technique that eliminates low-quality parameter combinations at coarser levels and reduces parameter combination domain for finer levels can also be used to configure any parameterized algorithms whose performance is greatly affected by the parameter settings. Finally, we conclude that the MLACO approach can be a powerful tool in the solution of the CFTPP and its multilevel design can be applied to other constrained combinatorial optimization applications.

5 Applying Pareto Ant Colony Optimization to Solve Bi-objective Forest Transportation Planning Problems

5.1 Introduction

In this chapter, we consider a bi-objective forest transportation planning problem (BOFTPP) with two objectives, namely, minimizing the transportation cost and sediment, which is the negative environmental impact. The difference from the previous studies is that, one of the goals is to minimize the sediment amount, instead of keeping it under an allowable level. This presents a new challenge because the best solution is determined by balancing between two objective values as opposed to only one for the CFTPP studied in the previous chapters. The practical use of solving the BOFTPP is to provide alternative solutions to help forest road planners make informed decisions that balance cost and environmental concerns.

To efficiently solve the BOFTPP, we develop a multi-objective ACO (MOACO) algorithm by combining many design choices (such as pheromone matrices update and transition probability formulation) proposed in recent literature [4, 36, 37, 65]. Each of the them is grabbed from ACO algorithms devised for tackling different multi-object combinatorial optimization problems, and has shown good merits in solving the tested problems in the experimental results. By integrating these design choices, the proposed MOACO offers possibilities that were not previously considered.

In addition, the developed MOACO incorporates the flexibility from the borrowed algorithmic components, which has reduced its dependency on parameter configurations (the number of parameters needed to be configured is reduced). Most importantly, the specific algorithm design in this study provides a powerful tool to solve the BOFTPP.

5.2 Preliminary Knowledge

5.2.1 Multi-objective Optimization

Multi-objective optimization problems are characterized by considering several objectives simultaneously [36, 4, 7]. In practice, there is no single best solution for the problem but a set of solutions that are superior when all objectives are considered. The solution set is known as Pareto set (named after the economist Vilfredo Pareto) or non-dominated solution set [1] (sometimes approximation set [89]). Mathematically, a multi-objective optimization problem with K objectives can be stated as:

$$minimize: \quad f(x) = \{f_1(x), \ldots, f_K(x)\}, \quad x \in \Omega, \tag{5.1}$$

where x is a feasible solution, Ω represents the feasible area in the solution search space, and $f(x)$ is a vector of objective functions that produces a vector of objective values R^K. A solution vector $a \in \Omega$ dominates another solution vector $b \in \Omega$ ($a \succ b$) if and only if:

$$\forall i \in 1, 2, \ldots, K | f_i(a) \leq f_i(b) \wedge \exists j \in 1, 2, \ldots, K | f_j(a) < f_j(b). \tag{5.2}$$

Similarly, The solutions a and b are incomparable ($a \parallel b$) if $a \nsucc b$ and $b \nsucc a$. In this thesis, we use the term "approximation set" for the solution set, which consists of incomparable solutions.

5.2.2 Bi-objective Forest Transportation Planning Problem (BOFTPP)

As aforementioned, the BOFTPP considers two objectives that need to be minimized: costs and sediments. The former objective is to reduce transportation costs from a set of timber selling locations to a set of designated mill locations. The latter objective is to reduce the total sediments eroding from the entire road network. Similarly to the CFTPP, the underlying transportation network in the BOFTPP is given as a directed graph $G = (V, E)$ with a set of $n = |V|$ nodes $\{v_1, \ldots, v_n\}$ representing road intersections and a set of edges $e_{i,j} \in E, \forall i, j \in [1, \ldots, n]$ representing road segments that connect adjacent road intersections. With the three attributes associated with each edge in the network: fixed cost (F_cost), variable cost (V_Cost), and sediment amount (Sed), the objective functions of costs and sediments are defined as follows:

$$\begin{aligned} \min(f_{cost}) &= \sum_{e_{i,j} \in E} (F\_Cost_{i,j} + V\_Cost_{i,j} \times Vl_{i,j}) \times b_{i,j}, \\ \min(f_{sedi}) &= \sum_{e_{i,j} \in E} Sed_{i,j} \times b_{i,j}, \end{aligned} \tag{5.3}$$

where $b_{i,j}$ is a binary variable that equals one if the edge $e_{i,j}$ is in use and equals zero otherwise. The objective of BOFTPP is to find the set of trade off (incomparable) solution routes connecting timber sales to mill locations that minimize, in the sense of Pareto optimality, the vector of objective functions $\overrightarrow{f} = (f_{cost}, f_{sedi})$.

5.3 Multi-objective ACO for BOFTPP

5.3.1 Construction of Non-dominated Solutions

The MOACO algorithm is presented in Algorithm 10. First, an equal amount of pheromone value is initialized on all problem components. Then, the algorithm begins to iteratively search for incomparable solutions and archive them in an approximation set. At each algorithm iteration, a new solution is obtained and compared with all the archived incomparable solutions that are obtained from the previous algorithm iterations. The new solution is determined to be incomparable and added to the approximation set if it does not dominate or is not dominated by any previously archived solutions. Otherwise, either the new solution is dominated by the previously obtained solutions, in which case the new solution is discarded or it dominates the previously obtained solutions, in which case the algorithm archives the new solution and removes those dominated by it. At the end of each algorithm iteration, the pheromone values are updated according to a selected solution (the select method is presented in later section) from the approximation set. The algorithm stops when a user defined condition (enough number of incomparable solutions) is satisfied.

Algorithm 10 MOACO for BOFPTT

```
Input(FTPP: D; A set of N real numbers: $\lambda_i \in \lambda, \lambda \subset [0, 1]$)
begin
    Approx_Set ⇐ $\phi$
    /* Initialize pheromone                                              */
    Initialize(D)
    while Inadequate incomparable solutions do
        foreach $\lambda_i \in \lambda$, $\forall i \in [1, N]$ do
            Iteration ⇐ 0
            while Iteration < Threshold do
                New_Solu ⇐ construct a solution with $\lambda_i$
                if New_Solu is incomparable then
                    add New_Solu to Approx_Set
                end
                else if New_Solu is dominated then
                    discard New_Solu
                end
                else
                    foreach solution S ∈ Approx_Set do
                        if New_Solu dominates S then
                            remove S from Approx_Set
                        end
                    end
                    add New_Solu to Approx_Set
                end
                Iteration ⇐ Iteration + 1
                Update_Phe(D, Approx_Set, $\lambda_i$)
            end
        end
    end
    Return(Approx_Set)
end
```

Note that the stop condition forces the MOACO to produce a certain number of incomparable solutions. Without this enforcement, MOACO may produce very few incomparable solutions because whenever a new solution is added to the archived approximation set, some previously found incomparable solutions may be removed. Although an important criterion to evaluate quality of an approximation set is its closeness to the Pareto frontier, the number of incomparable solutions obtained is another important quality assessment criterion. Ideally, we want to obtain all the Pareto solutions. In the literature, many heuristic algorithms are designed to stop after a maximum number of iterations. However, it is hard to set such a number appropriately for ACO based algorithms. On one hand, setting the iteration number too low will likely result in a low-quality solution set. On the other hand, if the iteration number is set too large, unnecessary amount of computing time may result.

Another important criterion to determine the quality of an approximation set is the solution distribution. A high-quality approximation set should contain incomparable solutions that cover wide range of the Pareto frontier or, ideally, the entire frontier. However, very often the obtained approximation set only concentrates on the best solutions for a single objective, or sometimes centers extremely towards trade off solutions [65]. In order to obtain non-dominated solutions that are more spread out and distributed evenly across the Pareto frontier, we use a set of λ values that divide an interval [0,1] into several equal-length subintervals with each subinterval representing a region of the Pareto frontier. The MOACO starts from the first Pareto region to the last Pareto region. At each region, MOACO iteratively searches for incomparable solutions. The solution search is guided by the corresponding λ values for the region. One cycle of the incomparable solution search process finishes after all Pareto regions are searched. If the stop condition is not satisfied, MOACO will start a new cycle of the incomparable solution search process. The previously found incomparable solutions are passed to the new cycle to provide a good initial solution set for updating pheromone values. Preliminary results suggest that because pheromone values are important in the ACO solution construction process, the quality of the resulted approximation set is expected to improve in the new round of incomparable solution search process for an equal number of iterations.

5.3.2 Decision Rule

The solution at each iteration (Algorithm 10) is constructed by making a sequence of decisions to select the next solution component. The decision is based on a transition probability that is calculated based on both heuristic (costs and sediment) and pheromone values (5.4).

$$P_{i,j}^k = \frac{(\frac{\text{F\_Cost}_{i,j}}{\text{Vl}_{i,j}} + \text{V\_Cost}_{i,j})^{(1-\lambda)} \times (\text{Sed}_{i,j})^{\lambda} \times (\tau_{cost})_{i,j}^{(1-\lambda)} \times (\tau_{sed})_{i,j}^{\lambda}}{\sum\limits_{l \in N_i^k} (\frac{\text{F\_Cost}_{i,l}}{\text{Vl}_{i,l}} + \text{V\_Cost}_{i,l})^{(1-\lambda)} \times (\text{Sed}_{i,l})^{\lambda} \times (\tau_{cost})_{i,l}^{(1-\lambda)} \times (\tau_{sedi})_{i,l}^{\lambda}} \tag{5.4}$$

where τ_{cost} and τ_{sedi} denote pheromone values for costs and sediment respectively. Because the BOFTPP considers two different objectives, we use two pheromone matrices for the transition probability, one for each objective. The parameter λ is used to specify different weights given to cost and sediment in different regions of the Pareto frontier. The values of λ are defined in the interval $[0, 1]$ with pace of 0.2 such that $\lambda = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$, which divides the Pareto frontier into 6 regions. The MOACO iteratively constructs solutions with the value of λ changing from 0 to 1 for each region. At the beginning, when $\lambda = 0$, the transition probability only considers cost, the algorithm focuses on finding incomparable solutions towards the cost minimization objective. As the value of λ increases, the weight gradually shifts from cost to sediment when calculating the transition probability. As a result, the incomparable solutions obtained are expected to reflect corresponding cost and sediment proportions according to their weights. Eventually, the transition probability only considers sediment when $\lambda = 1$. Note that, in the transition probability (Equation 5.4), we only use λ to adjust the relative importance between heuristic and pheromone values, whereas common ACO algorithms usually use two parameters, α and β. The reason is two-fold. First, since we want the obtained incomparable solution set to be more spread out and wider coverage of the Pareto frontier, there is no need to find a set of specific values of α and β that greatly affect performance of a single objective optimization problem. Second, this design reduces the number of parameters to be configured, thus making the MOACO algorithm more robust (less dependent on parameter setting).

5.3.3 Pheromone Update

Pheromone values are updated for both cost and sediment pheromone matrices using the Update_Phe procedure (Algorithm 11) where the current approximation set and the λ values are used. Solutions in the approximation set are evaluated according to the λ value in the corresponding Pareto region in order to select a solution for updating pheromone. The method to select the solution is described in Figure 5.1, where the Pareto frontier is divided into six regions according to the six different λ values. The MOACO starts with the first Pareto region where $\lambda = 0$ and archives all the obtained-so-far incomparable solutions. For all available solutions (blue squares surrounded by a red oval in Figure 5.1), a value according to a formula, $cost^{(1-\lambda)} \times sediment^{\lambda}$, is calculated. A solution (square with red edge in Figure 5.1) with the smallest value is selected to update pheromone. Note that when $\lambda = 0$, the selected solution will have the smallest total cost value. Next, for the second region, new incomparable solutions are obtained (blue circles in Figure 5.1) and archived together with previous obtained solutions (blue circles and squares surrounded by a green oval in Figure 5.1). From these solutions, a solution for pheromone update in the second Pareto region (circle with red edge in Figure 5.1) is selected using the same formula, but with a different λ value ($\lambda = 0.2$). Note that this process selects a solution not only from incomparable solutions obtained for the current region, but from incomparable solutions obtained for all the previously searched regions. This is because the best solution to update pheromone values for the current Pareto region may be obtained from other Pareto regions. As the solution construction process of MOACO is stochastic, incomparable solutions obtained from a region may belong to other Pareto regions. Therefore, all the obtained incomparable solutions are considered in our method to determine the most suitable solution for updating the pheromone. This idea is similar to the "update by origin" method used in [50]. Subsequently for the rest of the Pareto regions, the same solution selection process is used.

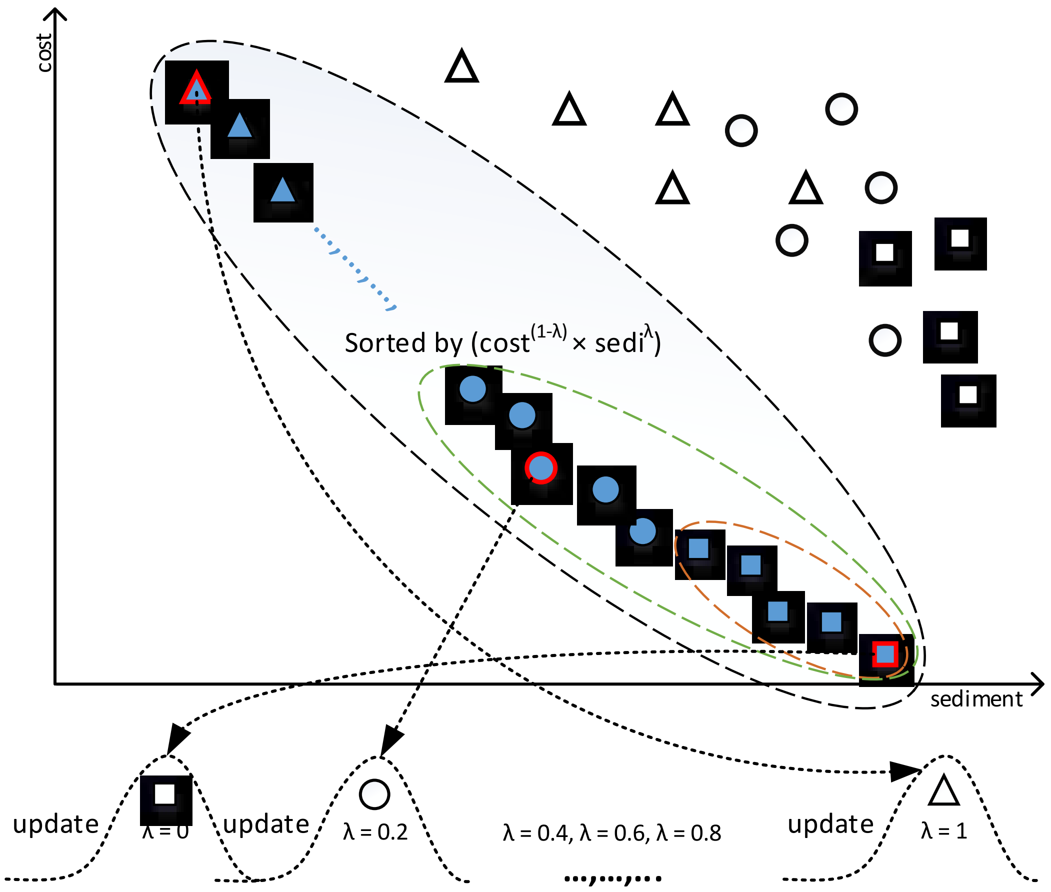


Figure 5.1: An incomparable solution from the approximation set is selected using the corresponding λ value to update pheromone values for a Pareto region. Incomparable solutions are shown as shapes filled with blue color (different shapes for different Pareto frontier region), the dashed ovals encompass the approximation set available for different regions where a solution (circle highlighted in red) is selected for pheromone update.

Although the λ value changes for different regions to adjust weights of the two heuristic values, the cost values are usually much larger than the sediment values which causes unbalanced comparisons. To cope with this, the cost and sediment values are normalized based on the largest objective value of solutions in the approximation set. This will restrict the value of cost and sediment within an interval [0,1] in the formula $cost^{(1-\lambda)} \times sediment^{\lambda}$ and help the algorithm determine an appropriate solution to update regional pheromone.

Algorithm 11 Update_Phe

```
Input(FTPP: D; incomparable solution set: Approx_Set; λ)
begin
    /* Find the appropriate solution */
    $Best\_Value \Leftarrow$ max possible number
    $Max\_Cost \Leftarrow$ max cost in $Approx\_Set$
    $Max\_Sedi \Leftarrow$ max sediment in $Approx\_Set$

    foreach solution $S \in Approx\_Set$ do
        if $Best\_Value > (\frac{S.cost}{Max\_Cost})^{(1-\lambda)} \times (\frac{S.sediment}{Max\_Sedi})^{\lambda}$ then
            $Best\_Value \Leftarrow (\frac{S.cost}{Max\_Cost})^{(1-\lambda)} \times (\frac{S.sediment}{Max\_Sedi})^{\lambda}$   $Best\_Solu \Leftarrow S$
        end
    end
    /* Update pheromone values using Best_Solu */

    foreach solution component $C \in D$ do
        if $C \in Best\_Solu$ then
            /* Update cost pheromone matrix */
            $\tau_{C\_cost} \Leftarrow \tau_{C\_cost} + \Delta\tau \times (1 - \lambda)$
            /* Update sediment pheromone matrix */
            $\tau_{C\_sedi} \Leftarrow \tau_{C\_sedi} + \Delta\tau \times (\lambda)$
        end
        else
            /* Evaporate pheromone with $0 < \rho < 1$ */
            $\tau_{C\_cost} \Leftarrow \tau_{C\_cost} \times \rho$
            $\tau_{C\_sedi} \Leftarrow \tau_{C\_sedi} \times \rho$

        end
    end
end
```

When updating pheromone values, the λ value is used again to adjust the amount of pheromone to be added to each solution component. The selected solution is used to update pheromone values for both cost and sediment pheromone matrices. For the edges used in the solution, pheromone values are increased by a small amount, $\Delta\tau$, multiplied by either $1 - \lambda$ for cost matrix or λ for sediment matrix. When the weight favors cost, this setting allows more pheromone update for cost than for sediment and vice versa. For the edges not used in the solution, their pheromone values are decreased by a factor ρ, which is the pheromone persistence rate.

5.4 Experimental Studies

5.4.1 Experiment Setup

The MOACO algorithm presented in this study was implemented using C++. Programs were executed on the computing nodes of Lipscomb High Performance Computing Cluster. To test for MOACO performance, we applied it to the ten FTPP problem instances used in [61]. For parameter setting, we set $\rho = 0.95$ to allow for slow pheromone evaporation. As the pheromone values were updated at each iteration, setting ρ too small would rapidly reduce the pheromone amounts on unused problem components, while only pheromone values on selected solution components would be increased, which would likely cause solution stagnation and provide a low-quality approximation set.

5.4.2 Approximation Set Evaluation

An important issue in multi-objective optimization is the evaluation of the quality of approximation sets. In single-objective optimization, solution quality is evaluated by the objective function. In multi-objective optimization, two solutions may be incomparable (i.e., neither dominates the other). Comparing two approximation sets is even more complex because it is difficult to define appropriate quality measures that take into consideration several factors such as proximity to the true Pareto frontier, and coverage of a wide range of diverse solutions [89]. Studies have addressed the problem of comparing approximation sets of the Pareto optimization quantitatively. Examples include the hypervolume measure [88], which considers volume of the objective space dominated by an approximation set, and the chi-square-like deviation measure, which tries to capture diversity of an approximation set [75]. A comprehensive analysis and review of multi-objective optimization performance assessment methods can be found in [89].

In this study, we used a unary ϵ-Indicator, (suggested in [89]), which is defined by a binary ϵ-Indicator to measure BOFTPP solution quality. For any two given approximation sets A and B, the binary ϵ-Indicator is defined as:

$$I_\epsilon(A, B) = \inf_{\epsilon \in R}\{\forall z^2 \in B, \exists z^1 \in A : z^1 \succeq_\epsilon z^2\} \tag{5.5}$$

where $z = (z_1, z_2, \ldots, z_n)$ represents a vector of an optimization problem with n objectives, and $z^1 \succeq_\epsilon z^2$ if and only if

$$\forall 1 \leq i \leq n : z_i^1 \leq \epsilon \times z_i^2 \tag{5.6}$$

for a given $\epsilon > 0$. The unary ϵ-Indicator is defined: $I_{\epsilon 1} = I_\epsilon(A, P)$ where A is an approximation set and P is a set of reference points. Instead of making pair-wised comparisons, all approximation sets are compared to a reference set which is assumed to be the same or close to the Pareto frontier so that the approximation set with the smallest ϵ-Indicator value with respect to the reference set is considered the best solution set. Because it is not possible to obtain the Pareto frontier for a multi-objective optimization problem beforehand, in the experiments, we used a reference point $\{$cost$= 700,000$, sediment $= 500\}$ to calculate the unary ϵ-Indicator based on experimental results presented in [61]. There are two reasons for using this reference point. First, due to the nature of the unary ϵ-Indicator, either using one reference point or a set of reference points will provide the same effect because the purpose is to quantify the closeness of the solution approximation set to the Pareto frontier. Second, in order to make the comparison straightforward, the calculated ϵ-Indicator values are expected to be always greater than zero so that approximation set with smaller ϵ value is consider better (closer to Pareto frontier).

Although other values for the reference point can be used (i.e., cost $= 0$, sediment $= 0$) that in theory could produce same results, large ϵ numbers can be produced that are less comparable than small numbers if the difference is too big and cause

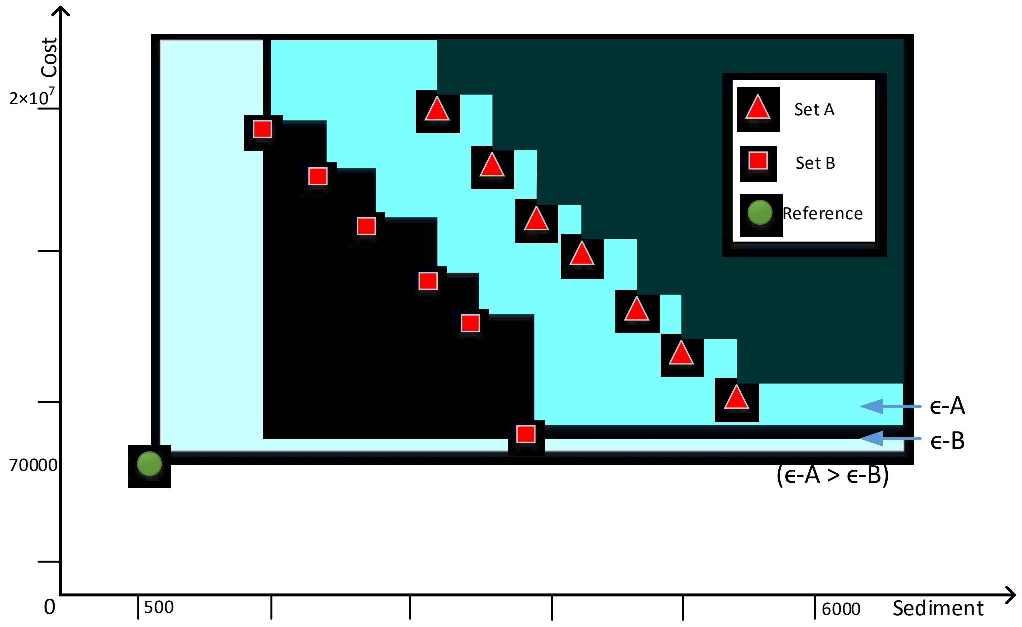


Figure 5.2: SetA (red triangle) and SetB (red square) are two approximation sets. To compare their quality, an ϵ-A value is calculated for setA and an ϵ-B value is calculated for SetB with respect to a reference point (greed circle). Since SetA is further away from the reference point than SetB, the value of ϵ-A will be greater than value of ϵ-B, indicating that SetB is a better approximation set than SetA.

difficulty when visualizing the results side by side. An example is illustrated in Figure 5.2 where two approximation sets SetA and SetB are compared based on their ϵ values (ϵ-A, ϵ-B) calculated with respect to the reference point.

5.4.3 Experimental Results

The MOACO was applied to solve each of the ten problem instances ten times with three stop conditions (obtain at least 40, 50 and 60 incomparable solutions) designed to test its performance on increasingly harder problem complexity. There were 300 approximation sets obtained and an ϵ-Indicator value was calculated for each of them. Figure 5.3 shows the mean ϵ-Indicator values of all problem instances over the 10 MOACO runs for the three different stop conditions. On average with respect to the ϵ-Indicator value, MOACO obtained best quality approximation set for the 60 stop condition, and the solution quality was reduced for 50 and 40 stop conditions. In general, the average ϵ-Indicator value increased as the number of required incomparable solutions decreased. This indicates the MOACO tends to obtain approximation set closer to Pareto frontier when the required number of incomparable solutions was set higher. Also, the ϵ-Indicator value gap is different among the three stop conditions for the same problem instance (Figure 5.3). In some cases, such as problem instances 2 and 6, the average ϵ-Indicator values for 40 and 50 stop conditions were almost the same, while the average ϵ-Indicator value for 60 stop condition was relatively smaller. In other cases (problem instances 3 and 9), MOACO obtained very close average ϵ-Indicator values for 50 and 60 stop conditions and the average ϵ-Indicator value for 40 stop condition was relatively larger. This is because of the stochastic nature of ACO algorithms where solution quality often varies for each algorithm run. However, this variation is quite small for the largest average ϵ-Indicator value difference is less than 0.5. This indicates that the solution quality is steady despite the stochastic nature of heuristic algorithms that the same solution can not be guaranteed at each run.

Computing time of MOACO for all problem instances is illustrated in Figure 5.4, which shows that the time spent is largest for 60 stop condition, followed by 50 and 40 conditions. This is because of the increasing difficulty to satisfy the stop condition. As the required number of incomparable solutions for the stop condition increases, more computational time was needed for MOACO. Computing time varied among different stop conditions. This variation was less evident for the 40 stop condition and became stronger for 50 and 60 stop conditions. Particularly, the computing time variation was larger for problem instances 1, 4, 9 and 10, but less obvious for problem instances 2, 6, 7 and 8. This indicates that, even with the same stop condition, the level of imposed complexity was different for the different problem instances. One of the factors that can determine the complexity of a BOFTPP is the size of its associated Pareto frontier and solution space. Problems with small Pareto frontier required longer computing time because the Pareto solutions were harder to identify.

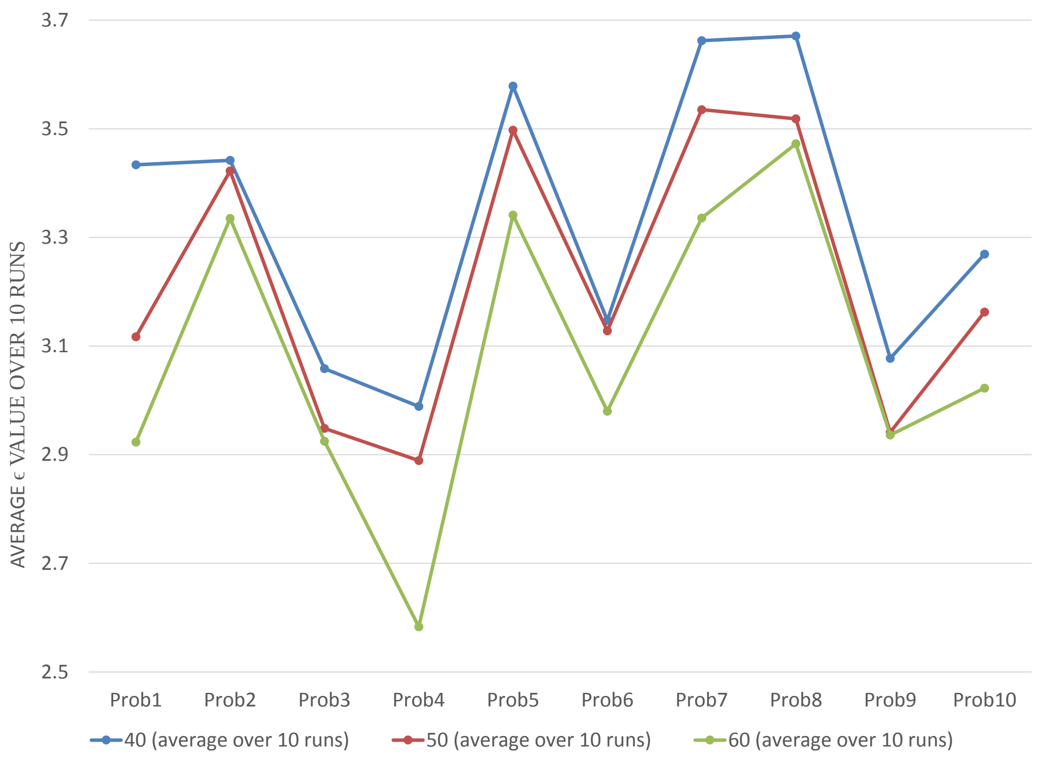


Figure 5.3: Average ϵ-Indicator values for all problem instances over three difference stop conditions.

Another observation is that MOACO spent much larger computing time for 60 stop condition compared to 50 and 40 stop conditions, which in general corresponds to the ϵ-Indicator values shown in Figure 5.3. This shows that at least for some problem instances, the performance of the MOACO was affected by the computing time spent. Better results seem to be obtained with longer computing time, which also justifies setting the parameter $\rho = 0.95$.

Figure 5.5 illustrates the distribution of ϵ-Indicator values over the ten MOACO runs on the ten problem instances for the three stop conditions. The ϵ-Indicator value trend appeared to decrease slightly as the number of required incomparable solutions increased. For the ϵ-Indicator value variations, there was no significant differences among different stop conditions. All the boxes are of similar length, although there are a few exceptions, such as problem instance 10 whose box length is much shorter for 60 stop condition than 40 and 50 stop conditions. Overall, the variation for ϵ-Indicator

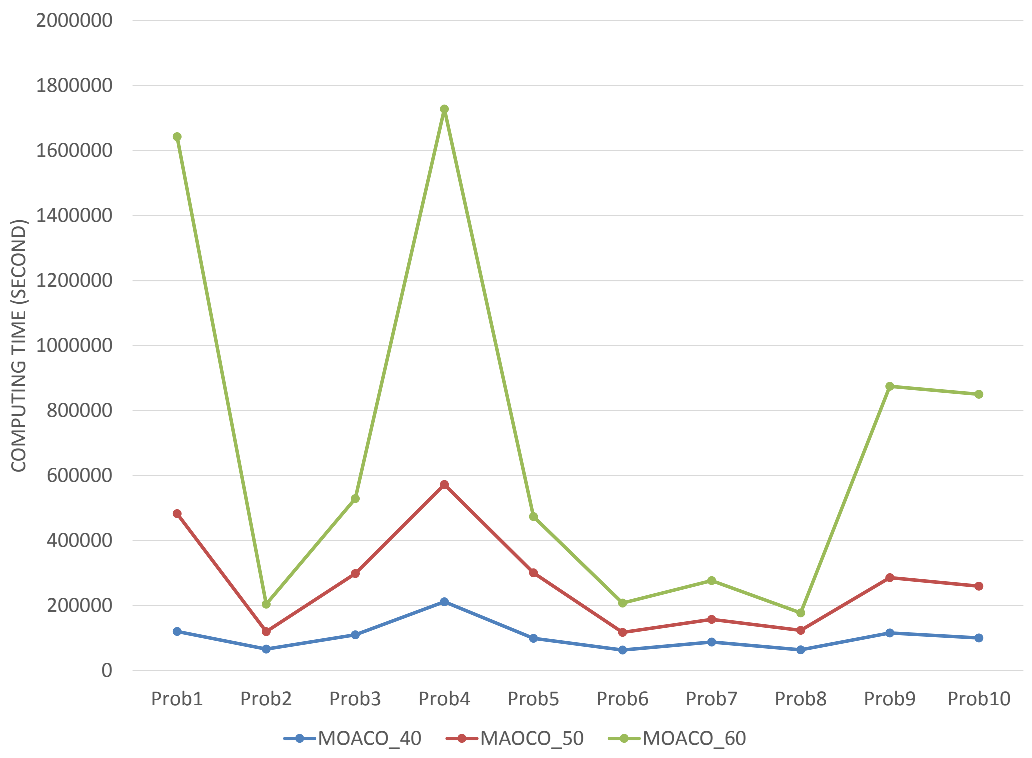


Figure 5.4: Computing times for all the problem instances over 10 MOACO runs.

values was small (less than 0.5), which corresponded to results in Figure 5.3 and indicated that MOACO was able to produce relatively consistent results regardless of stop condition settings.

Figures 5.6a-5.10b graphically present the results for the ten tested problem instances with three stop conditions. All approximation sets obtained by MOACO for the ten runs and three stop conditions were combined into one single approximation set by removing the dominated solutions from the joined set. For almost all problem instances, the approximation sets found for the 60 stop condition were better than those found for 40 and 50 stop conditions in terms of closeness to the optimal approximation set (closer to the reference point) and coverage of the Pareto frontier. This is more obvious in problem instance 2 where the approximation set obtained for 60 stop condition is much closer to Pareto frontier than the other stop conditions. For problem instances 3, 6, 7 and 8, the approximation set obtained for all three

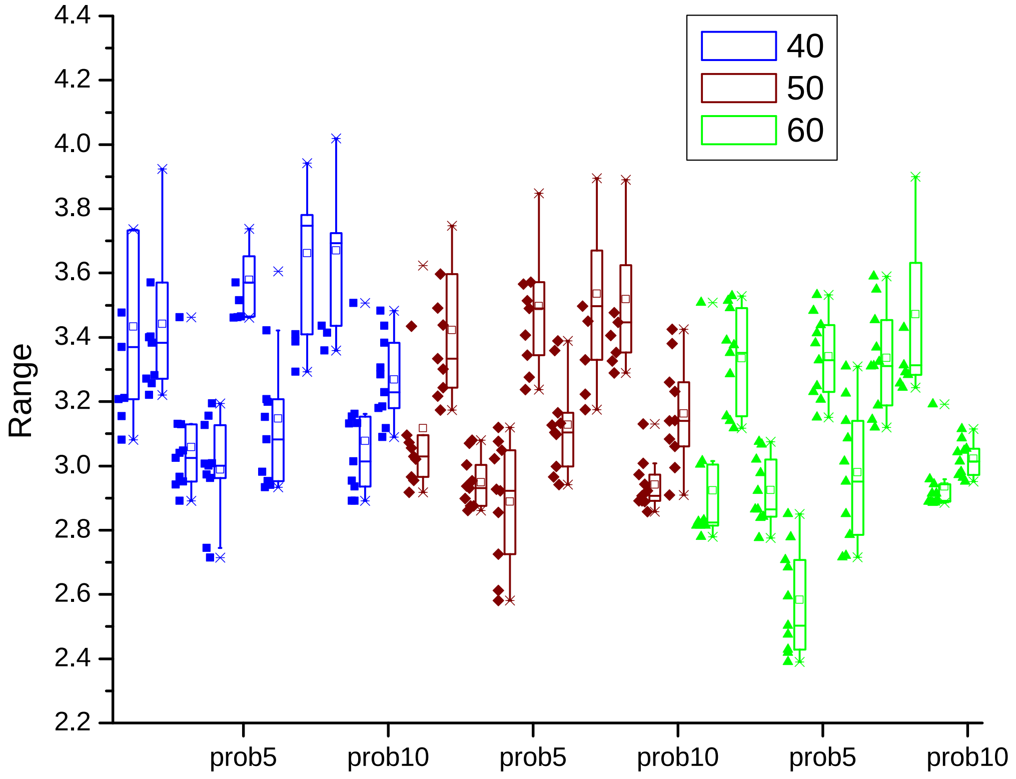
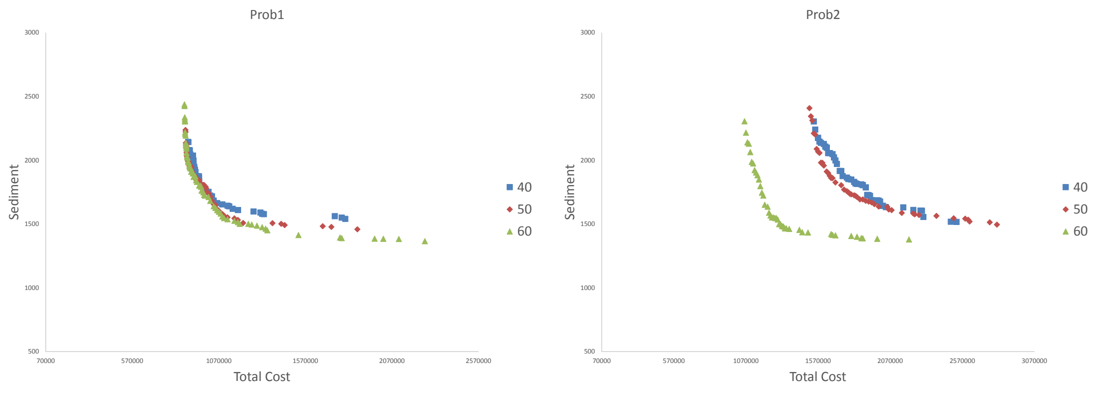


Figure 5.5: ϵ-Indicator value distribution of all 10 problem instances for 40, 50, and 60 stop conditions in form of box plot and scatter plot. Box plots show the Max, Min, Mean, 25^{th} and 75^{th} percentiles of ϵ-Indicator values, dots to the left of each box show scatter plots of ϵ-Indicator values.

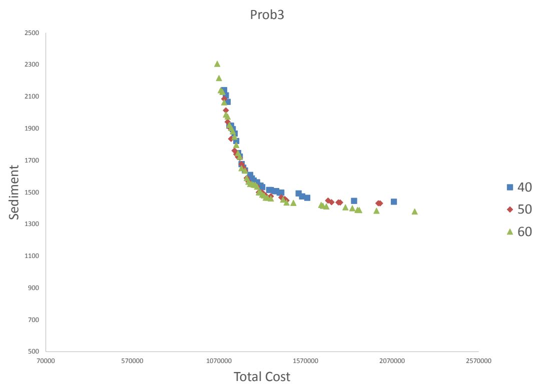


(a) Solution of Problem 1.

(b) Solution of Problem 2.

Figure 5.6: Distribution of average objective values for test cases 1 and 2.

stop conditions overlap each other, except that approximations for 60 stop condition usually cover a wider area of the Pareto frontier. For test cases 1, 4, 5, 9, 10, the


(a) Solution of Problem 3.

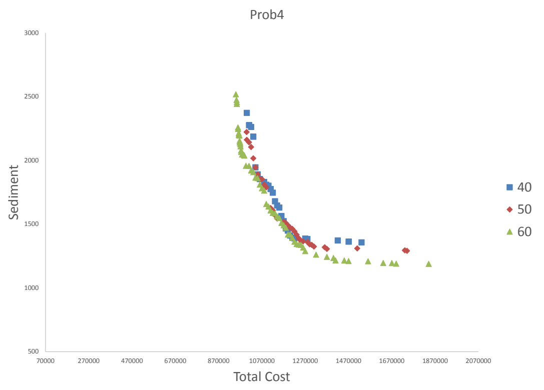
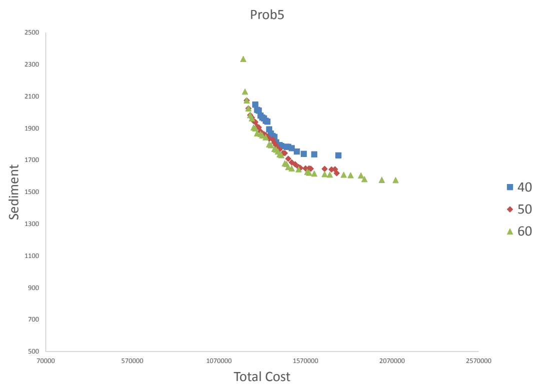

(b) Solution of Problem 4.

Figure 5.7: Distribution of average objective values for test cases 3 and 4.


(a) Solution of Problem 5.

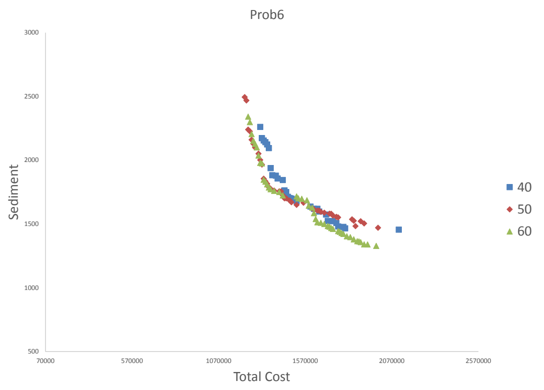
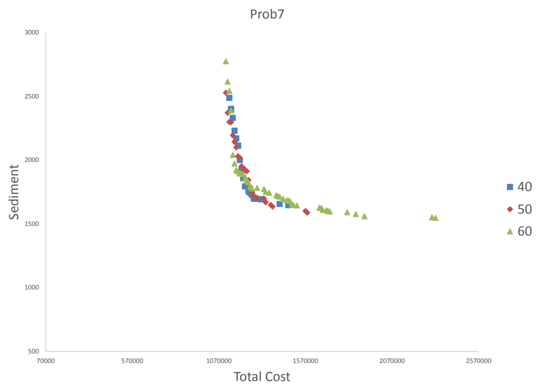

(b) Solution of Problem 6.

Figure 5.8: Distribution of average objective values for test cases 5 and 6.


(a) Solution of Problem 7.

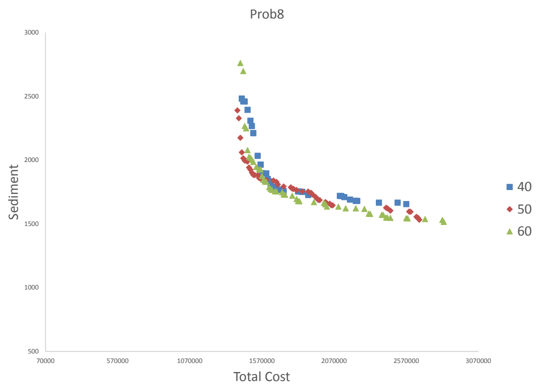
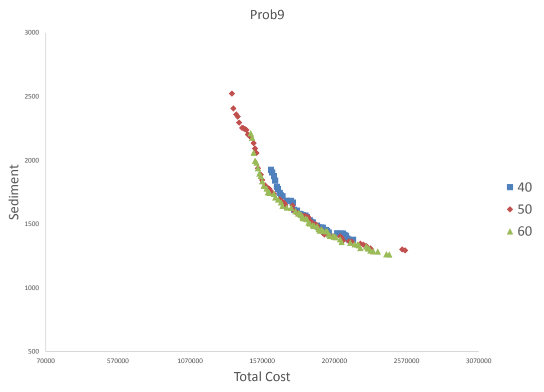
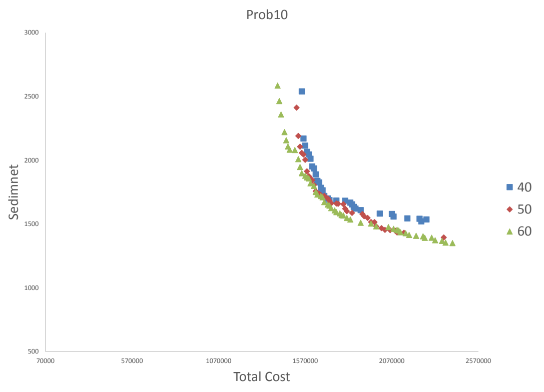

(b) Solution of Problem 8.

Figure 5.9: Distribution of average objective values for test cases 7 and 8.



(a) Solution of Problem 9.



(b) Solution of Problem 10.

Figure 5.10: Distribution of average objective values for test cases 9 and 10.

approximation sets obtained for 60 stop condition were slightly closer to optimal solution set than that of 40 and 50 stop conditions. These results indicate that MOACO's performance may be affected by different stop condition settings and it was able to obtain feasible approximation sets for all problem instances.

5.5 Concluding Remarks

In this chapter, we presented the design and implementation of an MOACO algorithm for solving a BOFTPP where the two objectives were minimizing timber transportation cost and environmental impact. First, the MOACO was designed to consider two pheromone matrices, one for each heuristic factor, to aggregate the cost and sediment information. This design helped the MOACO obtain trade off solutions. Secondly, the MOACO was designed to divide the Pareto front into different regions and obtained an incomparable solution set for each of the Pareto region. The final approximation set was then obtained by combining all incomparable solution sets. This design helps the algorithm explore wider range of the Pareto frontier. Thirdly, the MOACO was designed to have only one parameter, the pheromone evaporation rate ρ, for the underlying ACO algorithm, which reduced the MOACO dependency on its parameter setting and made the algorithm more robust.

To test for algorithm performance, we applied the MOACO to ten different FTPPs. Solution sets were visualized and their qualities were assessed using unary ϵ-indicator method. Experimental results showed that the MOACO algorithm was able to solve all testing problem instances and obtain approximation set under different stop conditions.

6 Solving Bi-objective Forest Transportation Planning Problems with Multilevel ACO Approach

6.1 Introduction

In this chapter, we continue the study from Chapter 5, in which an MOACO [62] is developed and implemented for solving the BOFTPP. The MOACO is devised with different algorithmic components in existing MOACO algorithms, which reduces ACO algorithm dependency on parameter configurations. In the experiments, while MOACO was able to solve all the test cases, the required computing times appear to vary significantly for different stop conditions and test cases. Therefore, in this chapter, we develop a multilevel MOACO (MMOACO) to improve the performance of the MOACO in terms of solution quality and computing time.

Similar to the previous works, the MMOACO is developed by first using a graph coarsening heuristic to generate a set of coarser level problems from the original problem, and then solve the original problem using these coarser level problems. As an expectation, MMOACO should require significantly less computing time compared to the MOACO based on previous experimental results. Additionally, the solutions from coarser level problems are used to help the ACO search for good solutions for finer level problems, which further improves the convergence speed and solution quality. More detailed description is presented below.

6.2 Multilevel Patero ACO

6.2.1 Coarsening Exact Graph

From previous studies, it can be observed, for the multilevel schemes developed, that it is important for finer and coarser level problems to share some common properties in such a way that the coarser level solutions can be effectively used to facilitate the underlying algorithm to find high-quality finer level solutions. For this reason, the coarsening procedure (Algorithm 12) is implemented to produce coarser level networks that are exact replicas of the finer graphs in terms of total attribute weights (sums of costs and sediment on every edge), but smaller in sizes (fewer number of nodes and edges). This is done by adding the weights of the combined edges and retaining the total weights of the collapsed edges to the aggregated nodes, which is different from previous Chapters where we assumed that nodes do not carry any weights.

For instance, when the nodes of a matching edge $edge(u, v)$ are collapsed to form a coarser level node u_v, the weight of node u_v is set to be equal to the sum of the weights of nodes u, v and edge $edge(u, v)$. For edges that are incident on the same node k in one end and incident either on u or v in another end, the weights of these edges are combined for the resulted edge in the coarser level network.

Algorithm 12 Coarsening_Procedure(G)

```
$QE \Leftarrow$ matching edges in graph $G$
foreach $edge(u,v) \in QE$ do
    $u\_v \Leftarrow$ aggregate $u$ and $v$
    weight($u\_v$) $\Leftarrow$ weight($u$) + weight($v$) + weight($u, v$)
    if $u$, $v$ are both adjacent to a node $k$ then
        weight($u\_v$,$k$) := weight($u$,$k$) + weight($v$,$k$)
    end
end
```

For finding the matching edges, we implement the heavy edge matching (HEM), which computes the maximal matchings that contain edges with large weights [53]. The nodes of a network are visited one at a time. If a node u has not been matched yet, then an unmatched adjacent node v is selected such that the weight of the edge (u, v) is maximum over all valid incident edges. For the BOFTPP, weights for an edge are considered to be the sums of associated costs and sediment values. The reasons to use the HEM are two-fold. First, as the two objective functions of the BOFTPP are minimizing total cost and sediment, high-quality solutions are likely to be comprised of small weighted edges. For a finer level network, the HEM finds matching edges with large weights. After having collapsed the matching edges, the resulting coarser level graph will consist of small weighted edges. Second, by visiting nodes in a sequential order and selecting only largest weighted edges, the HEM forces the matching process to be deterministic. This eliminates the randomness in the graph coarsening process, which leads to more stable final solutions.

6.2.2 Aggregation & Importance Factors

While both edges and nodes have cost and sediment attributes, nodes are associated with additional information, which are called aggregation and importance factors in this study. The aggregation factor represents the number of times the node has been aggregated with other nodes in the coarsening process, and the importance factor is a measure of the node's relevance in finding a high quality solution.

Figure 6.1 illustrates how the attribute values for nodes and edges are treated in the coarsening process from the original level (Figure 6.1. A) to the subsequent coarser level (Figure 6.1. B). This process begins with edge cost and sediment values assigned from the original problem. Cost and sediment values for each node are initialized to 0, and importance and aggregation factors are initialized to 1, of which the aggregation factor of 1 indicates that no aggregation has taken place. When matching edges are collapsed, the cost and sediment values of the aggregated node are both calculated by summing the values of combined nodes and collapsed matching edges. The aggregation factor of the new node is the sum of the aggregation factors of the combined nodes, and the importance factor remains 1 throughout the entire coarsening process. This procedure continues in subsequent coarser levels with the new values produced from aggregations.

6.2.3 Solution Mapping Process

After the coarsening process is applied to the original network and a set of increasingly coarser level networks are produced, an underlying ACO algorithm (which is discussed in later sections) is used to search for solutions starting from the coarsest

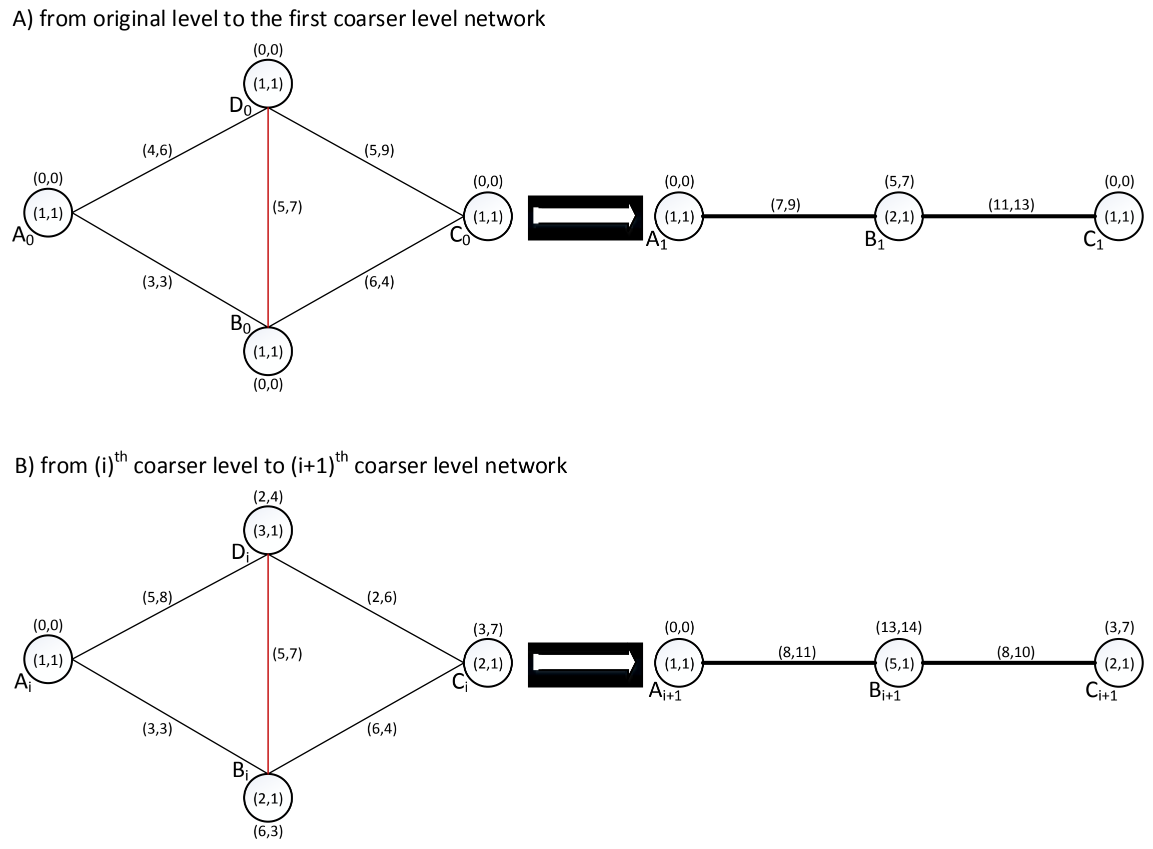


Figure 6.1: Diagram illustrating how the edges and weights of a fine level network are collapsed and aggregated to form edges and weights for the resulted coarser level network. Edge attributes include cost (left value in parentheses on each edge) and sediment (right value in parentheses on each edge). Node attributes include cost (left value in parentheses on each node), sediment (right value in parentheses on each node), aggregation factor (left value in parentheses in each node), and importance factor (right value in parentheses in each node).

level network. The solution search process of the underlying ACO algorithm for a finer level problem is helped by the solution obtained for the coarser level problem. This help comes from a mapping process, which maps the coarser level solution components into interpolated finer level components (similar to the approach used in Chapter 4) of which more considerations are given for constructing solutions.

The number of times that each node has been included in the solution set is counted and recorded during the coarsening process, which is used to set the importance factor. For nodes that are not included in the solution set, the importance

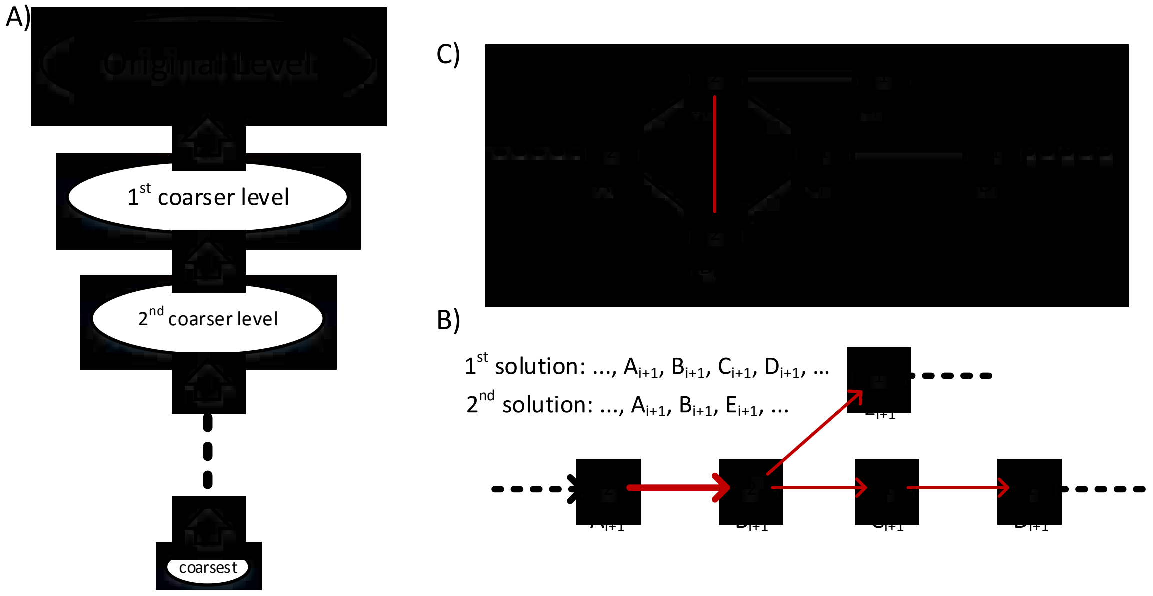

A)
Original level
1st coarser level
2nd coarser level
coarsest
C)
B)
1st solution: ..., Ai+1, Bi+1, Ci+1, Di+1, ...
2nd solution: ..., Ai+1, Bi+1, Ei+1, ...
Ei+1
Ai+1
Bi+1
Ci+1
Di+1

Figure 6.2: Diagram illustrating the solution mapping process and calculations of importance factors. The importance factor of node B_{i+1} and A_{i+1} of $(i+1)^{th}$ coarser level network is 2, as these nodes have been included in two solutions, whereas importance factors of other nodes are 1. The same importance factors are used for the interpolated nodes of i^{th} level network.

factors are set to one. The mapping process maps coarser level solutions into finer level problem components. Essentially, the interpolated problem components are the finer level (second coarsest level) problem components that are either collapsed or directly copied over to form the obtained coarsest level solutions. During the mapping process, the number of times that each node is included in the solution set is counted and used to set the importance factor. The interpolated components connected to nodes with high importance factors may be given higher considerations for being used as solution components, which expects to help the underlying ACO converge to a high-quality solution set faster.

An example of the coverall mapping process is illustrated in Figure 6.2, where solution components found for a coarser network are mapped to obtain interpolated finer level components, which, along with the importance factors assigned to the corresponding nodes, are used for finding high-quality finer level solutions (Figure 6.2. A). At a coarser level network ($(i+1)^{th}$ level network shown in Figure 6.2. B), the obtained solutions are used to calculate importance factors, of which the combined finer level nodes are set to have the same importance factor values as the aggregated coarser level nodes (e.g., importance factors for nodes B_i and F_i set to two, which is the same as the aggregated node B_{i+1} as shown in Figure 6.2. C).

6.3 Solution Construction

6.3.1 Construct Single Solution

The underlying ACO algorithm used to obtain approximation solution set is implemented according to the MOACO developed in Chapter 5, which incorporates many design choices and algorithmic components, such as using separate pheromone matrices for heuristic attributes and locally updating pheromone values with selected elite solution set, from existing multi-objective ACO algorithms that have been identified to be effective in several combinatorial optimization applications. The transition probability formula divides the Pareto frontier into different regions with a parameter λ that increasingly changes its value from 0 to 1. The values of λ are used to adjust the weight of objectives. When λ value is 0 or 1, the algorithm tends to obtain solutions towards each objective function (i.e., cost or sediment). When λ value is a fraction value in $[0, 1]$, however, higher consideration is given to trade-off solutions. The MOACO searches solutions from different Pareto regions. The final solution is obtained by combining solutions found for all the regions.

In the multilevel scheme, the MOACO searches for solutions from the coarsest level network to the finest level network (original problem). Because the settings of attributes at coarser level networks differ from the original network, the transition probability and objective function formulations are also defined differently compared to that in [62]. The newly defined formulations take the node attributes into consideration. Let Z denote the aggregation factor, I the importance factor. For a solution component $E_{i,j}$, the cost and sediment values are calculated as the following.

$$Cost_{i,j} = \frac{F\_C_{i,j} + F\_C_j/Z_j}{Vol_{i,j}} + (V\_C_{i,j} + V\_C_j/Z_j) \tag{6.1}$$

$$Sed_{i,j} = Sed_{i,j} + \frac{Sed_{i,j}}{Z_j} \tag{6.2}$$

where the $cost_{i,j}$ is the sum of unit fixed cost ($ per timber volume) and variable cost. Both edge and node costs are considered, where the node costs (Fix_Cost_j and Var_Cost_j) are divided by the aggregation factor. Similarly, the sediment is calculated by adding edge and node sediments, and the node sediment value is divided by the aggregation factor.

Note that the node attributes are divided by the aggregation factor for calculating both cost and sediment values. This is because we want to consider not only the total aggregated cost value, but also the number of matching edges collapsed. Dividing the node attributes by the aggregation factor provides expected attribute values of the collapsed matching edges from the coarsening process. The transition probability is then defined using the $Cost_{i,j}$ and $Sed_{i,j}$ as the following.

$$P^{(t)}_{i,j} = \frac{Cost^{(1-\lambda)}_{i,j} \times Sed^{(\lambda)}_{i,j} \times \tau^{(1-\lambda)\times I_j}_{Cost_{i,j}} \times \tau^{(\lambda)\times I_j}_{Sed_{i,j}}}{\sum_{l \in N^t_i} Cost^{(1-\lambda)}_{i,l} \times Sed^{(\lambda)}_{i,l} \times \tau^{(1-\lambda)\times I_l}_{Cost_{i,l}} \times \tau^{(\lambda)\times I_l}_{Sed_{i,l}}} \tag{6.3}$$

where the importance factor is used to adjust weights given to the pheromone values. Accordingly, the objective functions at coarser level networks are defined as:

$$f_{Cost} = \sum_{e_{i,j} \in E} [(F\_C_{i,j} + \frac{F\_C_j}{Z_j}) + (V\_C_{i,j} + \frac{V\_C_j}{Z_j}) \times Vol_{i,j}] \times b_{i,j} \tag{6.4}$$

$$f_{Sed} = \sum_{e_{i,j} \in E} (Sed_{i,j} + Sed_j/Z_j) \times b_{i,j} \tag{6.5}$$

At the original level, the aggregation and importance factors are set to one, which turns off their effects on calculating the transition probability and objective functions, and the node heuristic attributes are set to zero, which follows the original design in [62] and ensures that the calculations only consider edge attributes.

6.3.2 Construct Approximation Set

Interative Search Procedure

The approximation set is generated iteratively by the MOACO, which maintains an archive of non-dominated solutions that is updated whenever a new solution is obtained. These kind of archiving/selection strategies have shown the ability to improve solution quality towards the Pareto-optimal set and discover a diverse range of non-dominated solutions [59].

An abstract description of the iterative search in the MOACO is given in Algorithm 13, where t denotes the iteration count, f_t a newly generated solution at iteration t, and $A^{(t)}$ the archive that contains non-dominated solution selected from solutions obtained up to the t^{th} iteration. When a new solution is found, it is compared with solutions stored in the archive from previous iterations. The new solution will be discarded if it is dominated by any solutions in the archive. In this case, a new iteration will start as long as the stop conditions are not satisfied. Otherwise, the new solution is included in the archive. In this case, before starting a new iteration, the algorithm will first find those archived solutions dominated by the new solution, and then remove them from the archive.

Stop Conditions

The iterative search process (Algorithm 13) stops constructing approximation set when the stop conditions are satisfied, which should indicate that further iterations will not improve the solution quality. This is achieved by evaluating non-dominated solutions stored in the archive, and counting number of iterations that the quality of

Algorithm 13 Iteratively Constructing Approximation Set

```
begin
    $A^{(0)} := \phi$
    while $stop\ condition\ not\ satisfied$ do
        $t := t + 1$
        $f_t \Leftarrow ACO$
        if $\exists f_{t-1} \in A^{(t-1)}$, $such\ that\ f_{t-1} \succ f_t$ then
            discard $f_t$
        end
        else
            $A^{(t)} \Leftarrow A^{(t-1)} + f_t$
            $A^{(t)} \Leftarrow A^{(t)} \setminus S$, where $S \subset A^{(t-1)} \cap f_t \succ f_s, \forall f_s \in S$
        end
    end
end
```

the archived solution set being unchanged. In this study, the quality is assessed by ϵ-indicator, which is calculated using a reference point [89, 62]. The essential idea is to calculate the distances between each of the obtained approximation sets and the reference point. Based on the distances, comparisons can be made to determine better solution sets. The two heuristic values of the reference point are set to be small, then the solutions with closer distance to the reference point are considered better. As an example in the Figure 6.3, the solution B is better than A.

However, in some cases, using the ϵ-indicator alone is inadequate to properly compare two approximation sets because the set quality is determined not only by its closeness to the Pareto frontier, but also by the diverseness of the contained non-dominated solution. This is illustrated in the Figure 6.3 that the solution B appears to be better than C, when only the ϵ-indicator is considered. In practice, we may prefer the solution C because it covers wider range of the Pareto frontier.

Therefore, in the implementation, the stop condition is set to consider both the ϵ-indicator and the size of the approximation set (Algorithm 14). Specifically, a counter, that continuously adds up its value for each iteration, is used to stop the algorithm. When a new approximation set is generated, its ϵ-indicator is calculated and compared

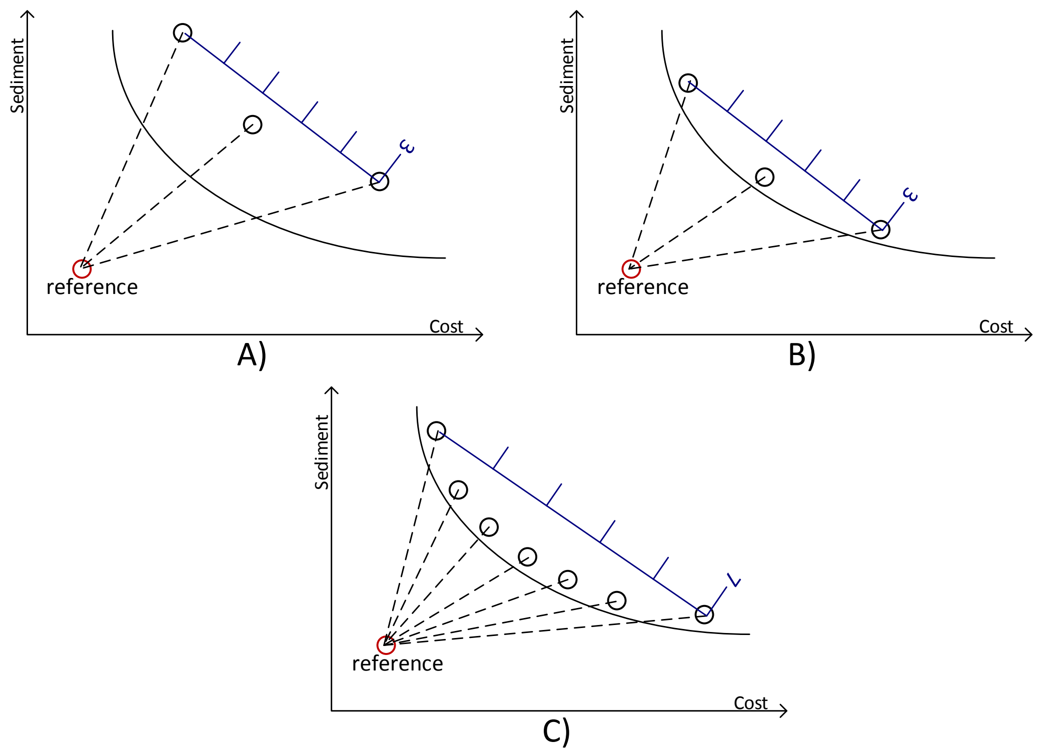


Figure 6.3: Preferred approximation set and its ϵ-indicator value.

Algorithm 14 Stop Conditions

```
begin
    ; /* Use a Counter and Threshold to stop iteration */
    while Counter < Threshold do
        if ε-indicator improved then
            Counter ⇐ 0 ; // reset Counter
        end
        else if ε-indicator not improved AND Size of approximation set increased then
            Counter ⇐ 0 ; // reset Counter
        end
        else
            Counter ⇐ Counter + 1 ; // update Counter
        end
    end
end
```

with the current best one. The counter is reset to zero if the new approximation set has a better ϵ-indicator, or bigger size in terms of non-dominated solutions. Otherwise, the counter value is updated and increased by one. The algorithm will stop when the counter exceeds a threshold value.

6.3.3 Construct Approximation Set from Selected Nodes

In addition to the importance factors that carry coarser solution information to the finer level, the MOACO is designed to search only the interpolated solution components for the finer level problem, i.e., the nodes that constitute the non-dominated solutions and the edges connecting them (Figure 6.4). At the beginning, after an approximation set is obtained by the MOACO, the contained non-dominated solutions are mapped to obtain second coarsest level solutions (interpolated solutions). The nodes and connecting edges that form the interpolated solutions are then used by MOACO to search for better solutions. The search is limited to using only these nodes and edges that constitute the interpolated solutions. After an approximation set is obtained for the second coarsest, the contained non-dominated solutions are again mapped to produce interpolated solutions for the third coarsest level problem, and MOACO will again search for better solutions using the nodes and edges that construct the approximation set. This process continues to each of the finer level problems until the original problem is solved.

Algorithm 15 Applying Dijkstra to Improve Initial Solutions

```
begin
    for each solution $C$ in the obtained approximation set from $G_n$ do
        1. Apply mapping process to $C$ and get an interpolated solution $I$ from $G_{n-1}$.
        2. Apply Dijkstra algorithm to $G_{n-1}$, limit the search to nodes that form $I$, and
           find solutions that minimize the considered objectives.
        3. Store the obtained solutions into an archive $A$.
    end
    • Refine and update $A$.
end
```

Moreover, a solution refinement process is implemented to further improve the

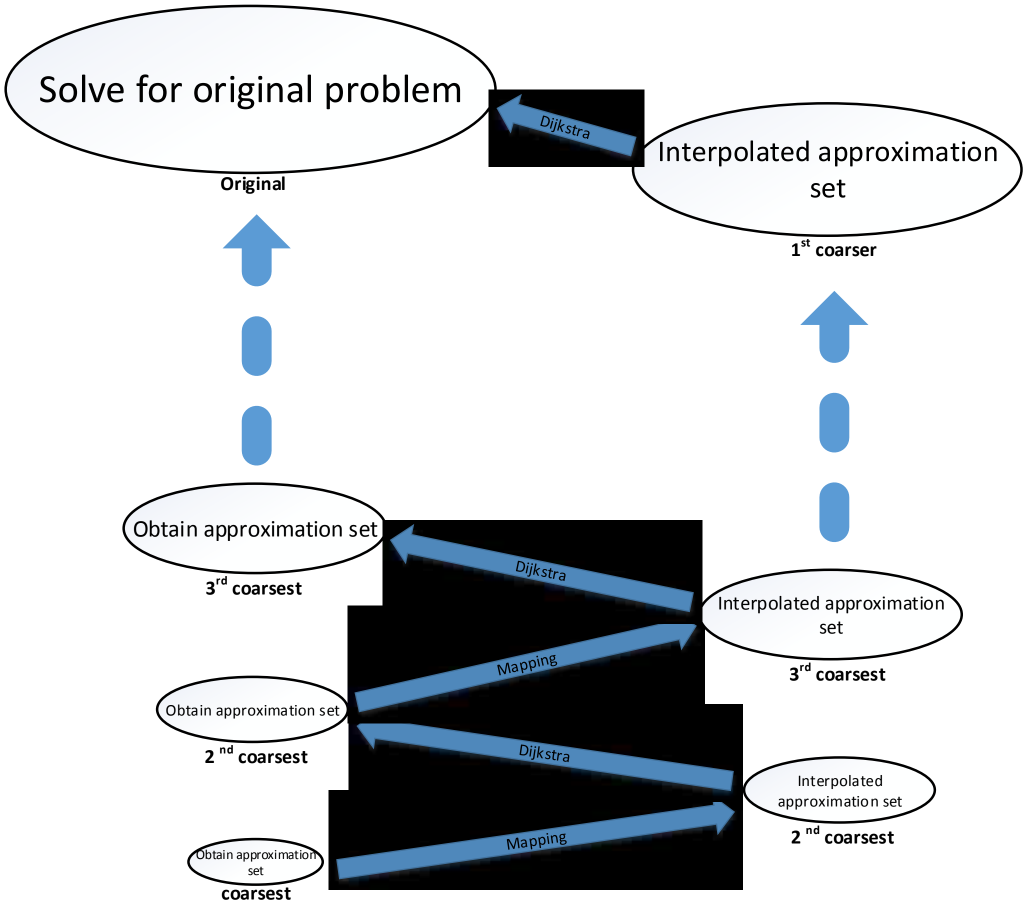


Figure 6.4: Constructing solutions from a set of selected problem components.

solution quality (Algorithm 15). For each of the interpolated solutions, the nodes and connecting edges form a smaller network, where the Dijkstra algorithm is applied to search for two routes that minimize total cost and sediment, separately. These two routes are, by definition, non-dominated to each other because they are guaranteed to be optimal in the given network. The Dijkstra is applied to every interpolated solution, resulting a new set of initial solution set which includes twice as many solutions as before. Finally, these solutions are refined again to remove any dominated solutions to obtain the approximation set.

6.4 Experimental Studies

6.4.1 Experiment Setup

The procedures and techniques presented in this study were implemented using C++. Programs were executed on the computing nodes of Lipscomb High Performance Computing Cluster. To test the performance, we applied the developed algorithm to the ten FTPP problem instances used for the previous chapters. Comparisons were made between the MOACO and MMOACO for solution quality in terms of the ϵ-indicator and computing time. Because MOACO is essentially a multi-objective ACO algorithm that searches for the best solution on the finest level problem, we refer to it as SMOCAO, which stands for single level MOACO, in order to distinguish from MMOACO. The threshold for the stop counter was set to 10000, and the reference point for calculating the ϵ-indicator is the same as the one used in Chapter 5. We followed the same experimental designs that set three conditions such that the first condition requires searching process stops when an approximation set containing 40 non-dominated solutions is found, and then the second and third conditions increase the stop requirement to 50 and 60.

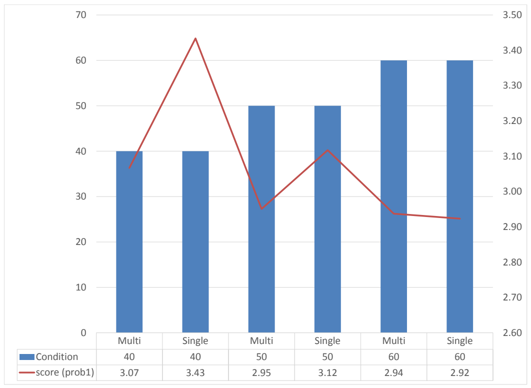
6.4.2 Experimental Results

Table 6.1 lists the statistical results of the ϵ-indicator value. It shows that, for most of the tested problems, there are significant differences between the two algorithms (SMOACO and MMOACO) and the three conditions (40, 50, and 60), whereas no significant differences can be observed for combinations of methods and conditions (method*condition). This indicates that the solution quality in terms of the ϵ-indicator is greatly affected by different methods and stop conditions, but the statistical inference between methods and conditions are not as significant. An exception is Problem 1, where the method and condition are affecting each other greatly. In this case, the individual analysis has to be conducted. Therefore, Table 6.1 also includes the statistical results considering each combination of methods and stop conditions.

Table 6.1: Statistical test (ANOVA) of solution quality in terms of ϵ-indicator (score). Values less than 0.05 indicate significant difference between the tested subjects.

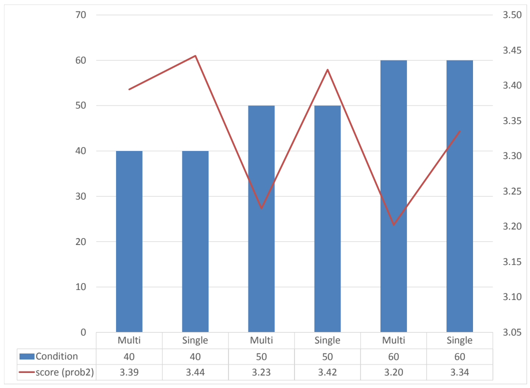
(Score)	**Prob1**	**Prob2**	**Prob3**	**Prob4**	**Pro5**	**Prob6**	**Prob7**	**Prob8**	**Prob9**	**Prob10**
Method	0.0006	0.0066	0.3122	0.2096	0.0011	0.7383	0.0043	$< .0001$	0.0166	0.4196
Condition	$< .0001$	0.0275	0.0041	$< .0001$	0.0004	0.033	$< .0001$	0.0095	0.007	0.0014
Method*Condition	0.0077	0.3964	0.5668	0.1403	0.135	0.6631	0.3707	0.7878	0.1475	0.2631
Multi vs. Single (40)	0.0006	0.9892	0.9663	0.9968	0.0277	0.9838	0.5264	0.0174	0.0426	0.5239
Multi vs. Single (50)	0.3463	0.1271	0.8259	0.9938	0.1686	0.9978	0.0862	0.1111	0.9994	0.9997
Multi vs. Single (60)	1	0.5245	0.9999	0.2237	0.9987	0.9944	0.9791	0.1649	0.9386	0.9909

Figure 6.5 displays ϵ-indicator comparisons for all the tested problems. It shows that, for the 40 condition, MMOACO generally produced better solution qualities than SMOACO for most of the problems. The cases where MMOACO had worse solution qualities are Problems 3 and 4. However, the statistical results in Table 6.1 indicate that the performance is not greatly affected by different tested methods and both multilevel and SMOACO methods are not expected to produce significantly different solution qualities for Problems 3 and 4. Similarly, for the 50 stop condition, MMOACO performed better than SMOACO for most of the problems, except for Problem 3, where SMOACO was able to obtain smaller ϵ-indicator value. As suggested from the statistical results, there is no significant difference between methods for Problem 3, indicating the result obtained by multilevel MOACO for Problem 3 is not much different from the single MOACO. For the 60 stop condition, MMOACO had better performances for 6 problems (Problems 2, 3, 5, 7, 8, and 9), and had worse performances for other problems (Problems 1, 4, 6, 10). Again, the statistical results indicate that although the multilevel MOACO performed worse than single MOACO for Problems 4, 6 and 10, the performance differences are not very significant. The statistical results also indicate that, for Problem 1, there is a correlation between methods and condition, and the individual tests show that there is no significant difference between the two methods.



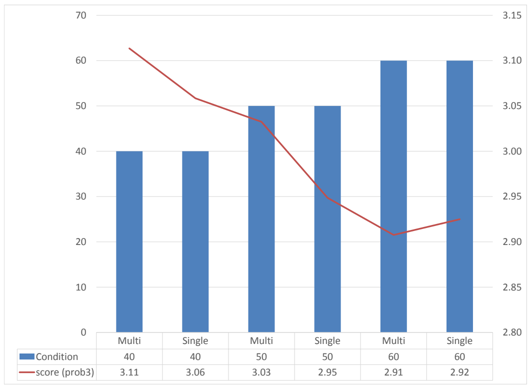
	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob1)	3.07	3.43	2.95	3.12	2.94	2.92

(a) ϵ-indicator of Problem 1



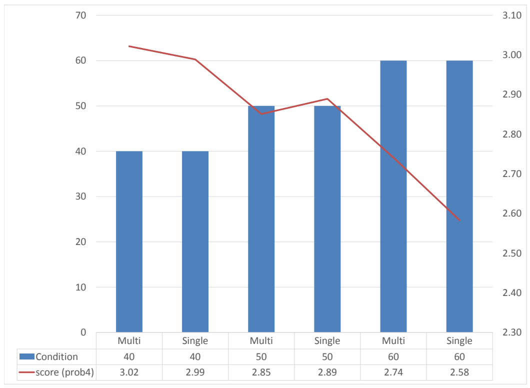
	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob2)	3.39	3.44	3.23	3.42	3.20	3.34

(b) ϵ-indicator of Problem 2



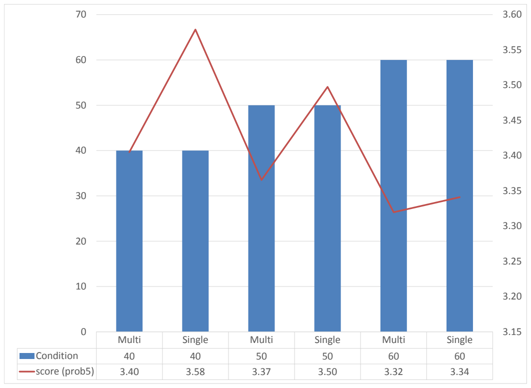
	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob3)	3.11	3.06	3.03	2.95	2.91	2.92

(c) ϵ-indicator of Problem 3



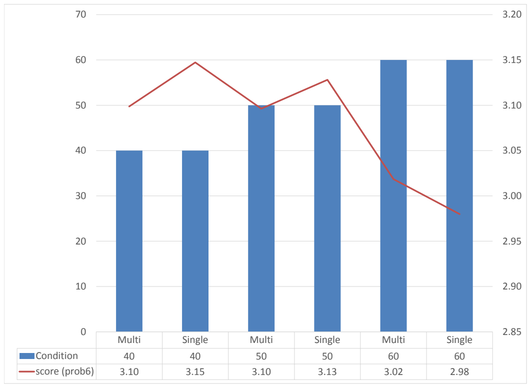
	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob4)	3.02	2.99	2.85	2.89	2.74	2.58

(d) ϵ-indicator of Problem 4



	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob5)	3.40	3.58	3.37	3.50	3.32	3.34

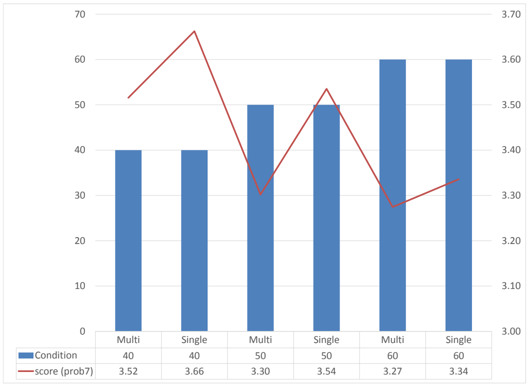
(e) ϵ-indicator of Problem 5



	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob6)	3.10	3.15	3.10	3.13	3.02	2.98

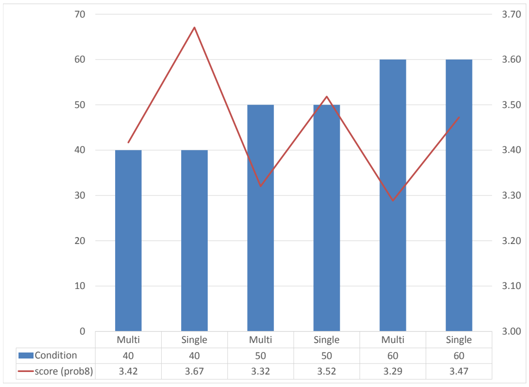
(f) ϵ-indicator of Problem 6

Figure 6.5: Comparisons of ϵ-indicator (score) between single and multi ACO algorithms in different stop condition.



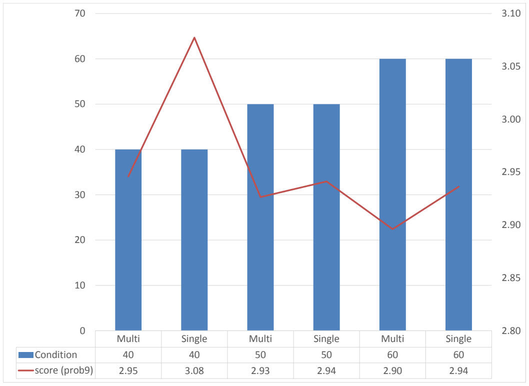
	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob7)	3.52	3.66	3.30	3.54	3.27	3.34

(g) ϵ-indicator of Problem 7



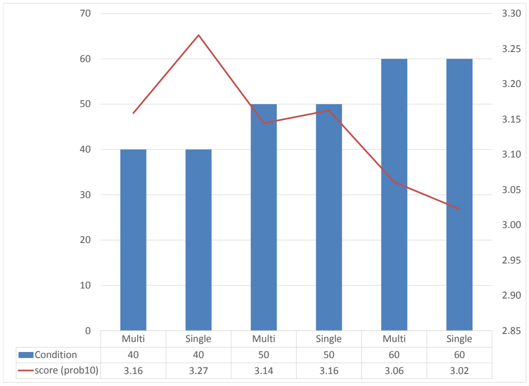
	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob8)	3.42	3.67	3.32	3.52	3.29	3.47

(h) ϵ-indicator of Problem 8



	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob9)	2.95	3.08	2.93	2.94	2.90	2.94

(i) ϵ-indicator of Problem 9

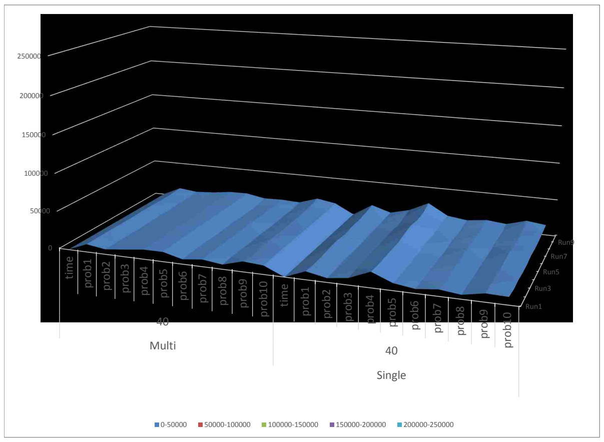


	Multi	Single	Multi	Single	Multi	Single
🟦 Condition	40	40	50	50	60	60
— score (prob10)	3.16	3.27	3.14	3.16	3.06	3.02

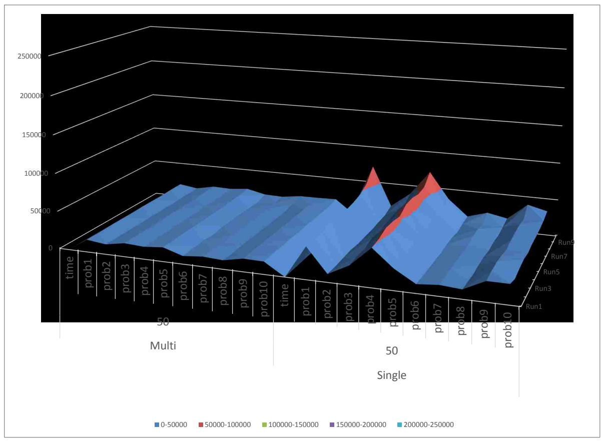
(j) ϵ-indicator of Problem 10

Figure 6.5: Comparisons of ϵ-indicator (score) between single and multi ACO algorithms in different stop condition.

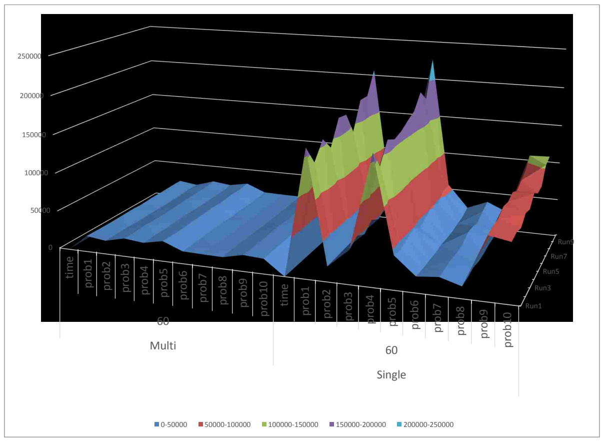
Figure 6.6 shows the time comparisons of the MMOACO and SMOACO among the three stop conditions. It shows that the time differences are smallest for all tested problems when the stop condition was set to 40. This time difference increases when the stop condition was set to 50, and became largest when the stop condition was set to 60. It can also be observed that the time change fluctuations were small for the MMOACO and large for the SMOACO. These results show that 1) when the time differences between MMOACO and SMOACO were not significant when the stop condition was set less complex and vise versa; 2) MMOACO was able to spend less computing times than SMOACO for most of the tested problems, especially when the 60 stop condition was used; 3) the MMOACO is more robust and stable in terms of computing time compared to SMOACO (less fluctuation).



(a) Computing time comparison between multi and single ACO for 40 stop condition.



(b) Computing time comparison between multi and single ACO for 50 stop condition.



(c) Computing time comparison between multi and single ACO for 60 stop condition.

Figure 6.6: Computing time comparisons between multi and single level ACO for the three stop conditions.

Table 6.2, which has the same layout as Table 6.1, lists the statistical results for computing time. Similar to the statistical results for the ϵ-indicator, the value in the table that is less than 0.05 indicates the tested subjects are significantly different. It shows that, for most of the problems, the performances of the two methods were significantly different in terms of the computing time, except for the Problems 2 and 8, where the p values are greater than 0.05. It also can be observed that different stop condition settings greatly influenced the computing time for all problems. Furthermore, because the methods and conditions are all affecting each other, the individual statistical results are also given. Generally, the two methods were not significantly different when the stop condition was 40, and the differences became significant when the condition became harder to achieve. These results correspond to the results observed from Figure 6.6.

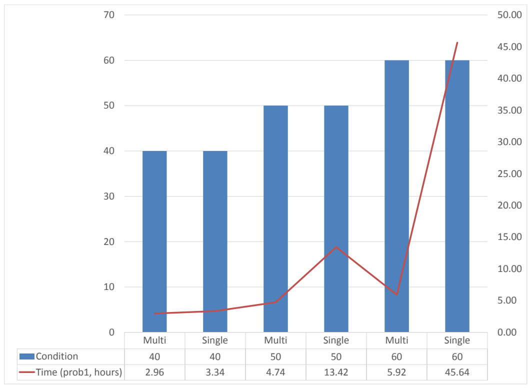
Table 6.2: Statistical test (ANOVA) of computing time. Values less than 0.05 indicate significant difference between the tested subjects.

(Time)	**Prob1**	**Prob2**	**Prob3**	**Prob4**	**Pro5**	**Prob6**	**Prob7**	**Prob8**	**Prob9**	**Prob10**
Method	$< .0001$	0.5948	$< .0001$	$< .0001$	$< .0001$	0.0163	$< .0001$	0.6748	$< .0001$	$< .0001$
Condition	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$
Method*Condition	$< .0001$	0.0015	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	0.093	$< .0001$	$< .0001$
Multi vs. Single (40)	0.9996	0.9973	0.9948	0.4426	0.0695	0.499	0.9564	0.9794	0.8922	0.9336
Multi vs. Single (50)	$< .0001$	0.3905	$< .0001$	$< .0001$	$< .0001$	0.9969	0.2353	0.9893	$< .0001$	0.0041
Multi vs. Single (60)	$< .0001$	0.0189	$< .0001$	$< .0001$	$< .0001$	$< .0001$	$< .0001$	0.3223	$< .0001$	$< .0001$

Figure 6.7 shows the computing time comparisons for each problem. For 40 condition, MMOACO tends to spend more time than SMOACO for most of the problem. However, according to the statistical results (Table 6.2), the differences in computing time between the two algorithms were not significant. For 50 condition, MMOACO performed better for 7 problems, but worse for the other 3 problems. For the problems that MMOACO had worse performances, the results were not significantly different (Table 6.2). For the 60 stop condition, the similar results can be observed. These results clearly showed that MMOACO is superior than SMOACO in computing time.

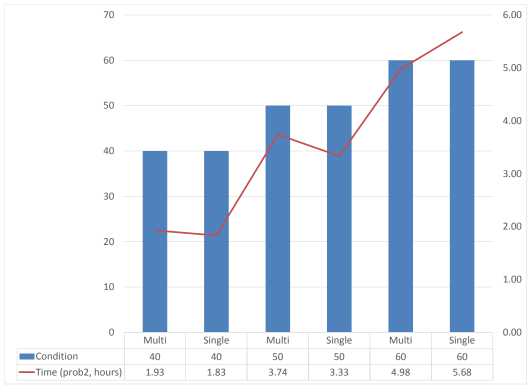
6.5 Concluding Remarks

In this Chapter, a multilevel multi-objective ACO algorithm is developed. First, the exact graph coarsening technique is recursively applied to the original problem to produce a set of coarser level problems that have the same total amount attribute values as the original problem. This helps the coarser level problems to retain important properties of the original problem so that the coarser level solutions may be



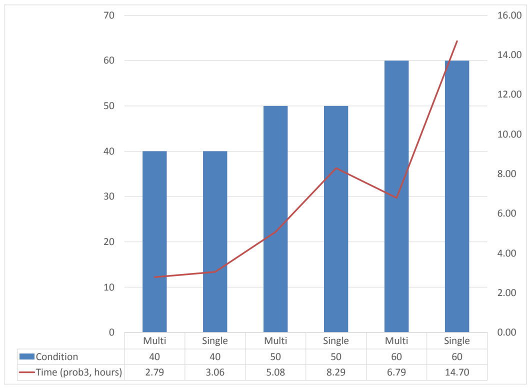
	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob1, hours)	2.96	3.34	4.74	13.42	5.92	45.64

(a) Computing Time of Problem 1



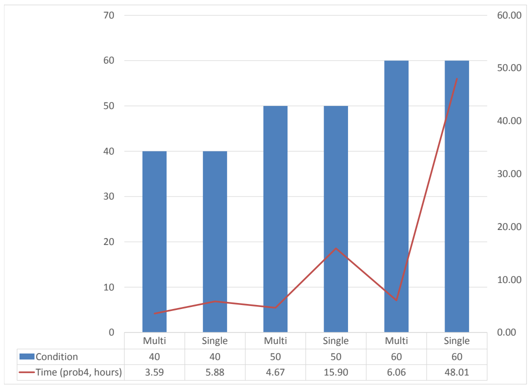
	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob2, hours)	1.93	1.83	3.74	3.33	4.98	5.68

(b) Computing Time of Problem 2



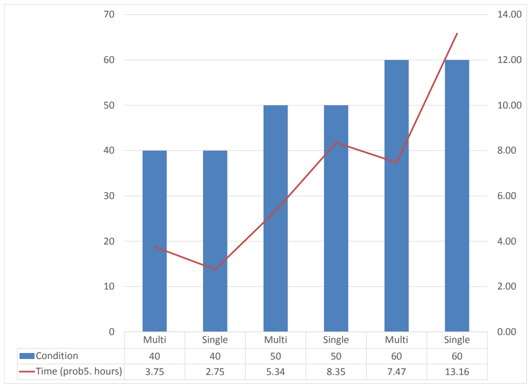
	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob3, hours)	2.79	3.06	5.08	8.29	6.79	14.70

(c) Computing Time of Problem 3



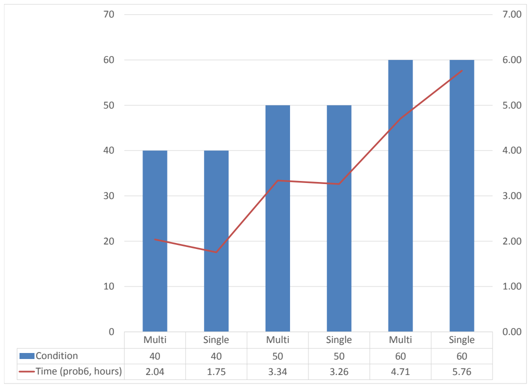
	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob4, hours)	3.59	5.88	4.67	15.90	6.06	48.01

(d) Computing Time of Problem 4



	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob5. hours)	3.75	2.75	5.34	8.35	7.47	13.16

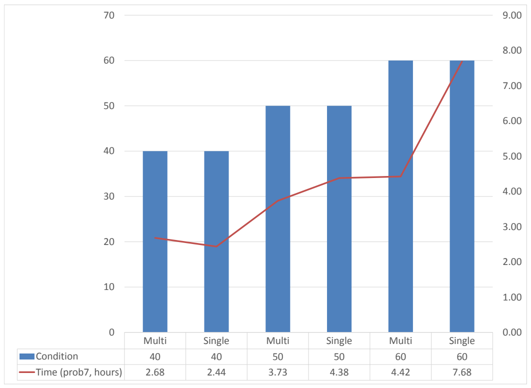
(e) Computing Time of Problem 5



	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob6, hours)	2.04	1.75	3.34	3.26	4.71	5.76

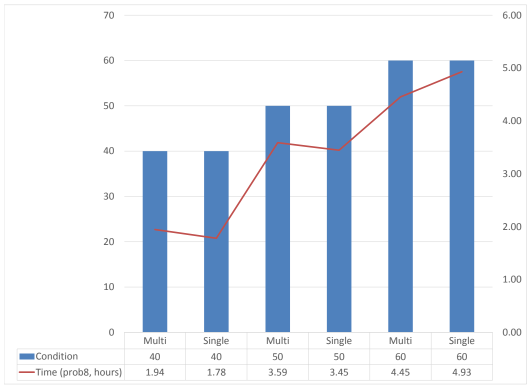
(f) Computing Time of Problem 6

Figure 6.7: Comparisons of computing time between single and multi ACO algorithms in different stop condition.



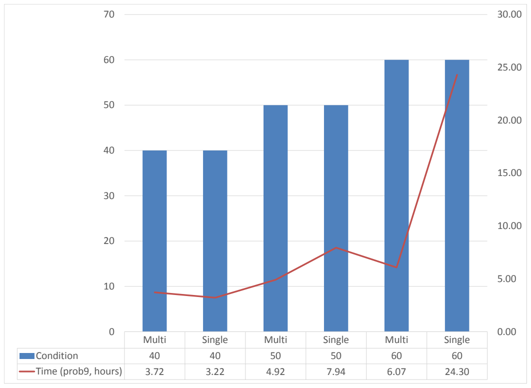
	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob7, hours)	2.68	2.44	3.73	4.38	4.42	7.68

(g) Computing Time of Problem 7



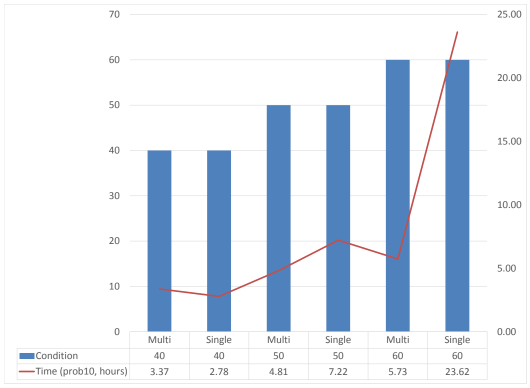
	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob8, hours)	1.94	1.78	3.59	3.45	4.45	4.93

(h) Computing Time of Problem 8



	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob9, hours)	3.72	3.22	4.92	7.94	6.07	24.30

(i) Computing Time of Problem 9



	Multi	Single	Multi	Single	Multi	Single
▬ Condition	40	40	50	50	60	60
— Time (prob10, hours)	3.37	2.78	4.81	7.22	5.73	23.62

(j) Computing Time of Problem 10

Figure 6.7: Comparisons of computing time between single and multi ACO algorithms in different stop condition.

used to help search for finer level solutions. Second, the underly MOACO is designed to consider importance and aggregation factors to obtain solutions. These two factors represent importances of the interpolated finer level edges and the aggregation degrees, which can guide MOACO towards searching for better solutions. Third, the interpolated solutions are refined using the Dijkstra algorithm before they are used as initial good solution set for MOACO. This design further increases the chance of finding a good solution set for the finer level problems. Finally, the MOACO is constrained to search solutions only using nodes that are included in the interpolated solutions. This design helps reduce the total computing time. The MMOACO was applied to ten problem instances and its performance was compared with SMOACO.

The experimental results show that MMOACO can produce better solutions with much less time in comparison to SMOACO.

7 Conclusion and Future Work

This dissertation presents a research work in computational science to develop a multilevel paradigm, which targets at solving large-scale transportation problems. Although the constrained forest transportation planning problem is used as the test problem, the developed algorithms can be extended to other problem domains with minor modifications. This work involves different Ant Colony Optimization algorithms, graph coarsening heuristics, mixed integer programming, automatic parameter configuration techniques, constrained and unconstrained fixed charge transportation problems, multi-objective optimization problems, and quantifying the quality of approximation sets. In this chapter, I summarize my dissertation work and discuss some possible future research topics.

7.1 Research Accomplishments

In the computational science and operations research, solving complex optimization problems with metaheuristic algorithms has received considerable attention among practitioners and researchers. Hence, many metaheuristic algorithms have been developed over the years. These algorithms are often inspired by various phenomena of nature, and are designed to provide solutions to many difficult engineering optimization problems in which the search spaces grow exponentially with the problem size. As a counterpart, the traditional optimization methods usually fail with the limited computational resources.

The optimization problem dealt with in this thesis is called a fixed charge transportation problem, which is a special case of the fixed cost linear programming problem induced in the origins of the operations research [2, 82]. It is also a special case of the cost minimization network problem that has been the center of many distribution and network design problems [68]. The introduction of the fixed cost to the linear programming and transportation problems has changed the objective function to be concave and discontinuous in the search region [44], which also implies that the fixed charge problems are much more difficult to solve than their corresponding linear versions. In fact, this has been shown in [40, 56] that most minimization network problems with strictly concave objective functions, including these with fixed-charges, are NP-hard.

Two-stage ACO Algorithm to Solve CFTPP

The first contribution of this research work is that an ACO algorithm is designed to solve the CFTPP, which is a real world application of the constrained fixed charge transportation problem. The algorithm is comprised of two stages such that it tries to determine the existence of the feasible solution in the first stage, and improve the quality of the obtained feasible solution in the second stage. Moreover, the ACO algorithm follows different search routines, performs edge selection and pheromone update mechanisms under different situations, and integrates a local search at the end, which further improves the solution quality. All these designs are customized specifically for the constrained fixed charge transportation problem, providing an algorithmic tool for solving the CFTPP.

Multilevel Approach to Automatically Configure ACO

The second contribution is that a multilevel ParamILS technique is proposed and implemented to configure ACO for solving transportation problems. The MParamILS is designed by combining a graph coarsening heuristic and the state of the art parameter configuration framework ParamILS. The graph coarsening is applied recursively to the original problem, which produces a set of increasingly smaller problems of which the basic network layout and attribute distribution of the original problem are well inherited. Because of this, a philosophical assumption was made that a set of high-quality parameters is shared among the coarser and finer level problems. Therefore, the MParamILS applies the ParamILS to each coarser level problem to refine a parameter combination domain, which identifies the high-quality parameter combinations and eliminates the low-quality ones. This parameter refinement process starts from the coarsest one to the finest level, until the original problem is solved. As shown in the experimental results, the proposed MParamILS can save computing time significantly compared to ParamILS.

Multilevel ACO Algorithm to Solve CFTPP

The third contribution is a continuing study of the previous works. A multilevel ACO algorithm is developed to solve CFTPP that integrates the MParamILS for configuring the parameters and uses the solutions found for the coarser level problems to facilitate the ACO solution search process for the finer level problems. The coarser level solution components are mapped into the corresponding finer level problem components. These interpolated finer level components are given more considerations when searching for the best solution by initializing larger amounts of pheromone values to them. The essential idea behind this is that the interpolated problem components of a finer level problem are expected to form the best solution because they are mapped from the best solution of its next coarser level problem. Therefore, concentrating the search process on these problem components could reduce the search space, and eventually lead to the best solution much faster. This has been proven in the experiments that the multilevel ACO shows considerable improvement in computing time and better objective function values in most of the test cases, comparing to only the MParamILS used to solve the CFTPP.

Applying the ACO to Solve Bi-objective FTPP

The fourth contribution is that we present a multi-objective ACO algorithm to solve a bi-objective FTPP in which the objectives are to minimize both the transportation cost and negative environmental impact. The goal is to provide the forest managers or planners alternative solution choices, in order to make informed decisions. Many algorithmic components in existing ACO algorithms that are designed for solving multi-objective optimization problems are considered. As a result, the proposed MOACO offers a new possibility in algorithm design choices that was never considered before. In addition, the incorporated flexibility from the different existing ACO algorithm components reduces the MOACO's dependency on the parameter settings. From the experimental results, the MOACO has shown to successfully solve all the tested problem instances.

Multilevel ACO to Sovle Bi-objective FTPP

The last contribution of this thesis study is that a multilevel MOACO algorithm is developed to improve the performance of the MOACO. The algorithm applies a exact graph coarsening technique to produce the coarse level problems of which the aggregated attribute weight is the same as that of the original problem. This increases the connections between the coarser and finer level problems so that the coarser level solutions can be used to help search for finer level solutions. Furthermore, the interpolated solutions are refined using the Dijkstra algorithm before they are used for initializing the pheromone, hence increasing the quality of the selected problem components that are given more consideration. The experimental results show that the multilevel MOACO is substantially faster than the MOACO for all the tested cases.

7.2 Future Work

In the future, I plan on continuing my current lines of research. In particular, I would like to continue the research on the multilevel scheme, hierarchical computing, and extending the developed multilevel framework to other problem domains and optimization algorithms. The following sections briefly explain my ideas and different directions that I would like to pursue.

Parallelization for Multilevel ACO

For large size problems, ACO will require a significantly large amount of time for searching good parameter combinations and evaluating algorithms. ACO algorithms are highly distributed algorithms in which a set of ants construct solutions independently. Thus, their parallelization is relatively straightforward. However, the parallelization has not yet been explored in the multi-objective optimization context [19]. Furthermore, the multi-level ACO framework allows many components to combine in many different ways to improve ACO performance. In the future, I would like to integrate parallelism to the multi-level ACO to solve constraint multi-objective optimization problems. The plan is to first design a parallel multi-level ACO framework and then implement it using massage passing interface (MPI).

Automatically Determine the Number of Coarser Level Problems

We have examined the performance of using multilevel approaches to solving transportation problems and parameter configuration problems. In both cases, the general idea is to first coarsen the original problem to produce a set of increasingly smaller problems, then solve the original problem, which is expensive to solve, using the obtained coarser level problems of which the computational expense is cheaper. In our studies, the number of the coarser level problems is four, which is set according to the ratio of the sizes between the original problem and the coarser level problems. However, there is no formal proof that can theoretically justify this setting. The future work should focus on providing some genetic rules or techniques to automatically determining the right number of coarser level problems to be used in the multilevel approaches.

Extending the Multilevel ACO to Solve Other Problem Domains

As aforementioned, although the multilevel ACO is designed for solving the CFTPPs, it can be easily modified to tackle other problem domains. In particular, I will focus on the interdisciplinary studies between computer science and forest management to develop computational tools that incorporate multilevel schemes for solving problems related to operations research. Possible projects include 1) forest fire treatment problems where good tree planning strategies are needed to prevent or reduce forest fire hazards; 2) forest transportation planning problems where additional variables (such as detailed operational costs, different timber harvest periods, other than the environmental impact, sediment.) may be considered; 3) Analyzing the digital elevation model (DEM) and LiDAR data for forest management problems such as tree detections in a specific area, tree species identifications, and timber volume inventory and so on.

Bibliography

[1] Ajith Abraham and Lakhmi Jain. *Evolutionary Multi-objective Optimization*. Springer, 2005.

[2] Jesús Sáez Aguado. Fixed charge transportation problems: a new heuristic approach based on lagrangean relaxation and the solving of core problems. *Annals of Operations Research*, 172(1):45–69, 2009.

[3] A. E. Akay, O. Erdas, M. Reis, and A. Yuksel. Estimating sediment yield from a forest road network by using a sediment prediction model and gis techniques. *Building and Environment*, Vol.43(5):687–695, 2008.

[4] Daniel Angus and Clinton Woodward. Multiple objective ant colony optimisation. *Swarm Intelligence*, 3(1):69–85, 2009.

[5] Ronald G Askin, Steven H Cresswell, Jeffrey B Goldberg, and Asoo J Vakharia. A hamiltonian path approach to reordering the part-machine matrix for cellular manufacturing. *International Journal of Production Research*, 29(6):1081–1100, 1991.

[6] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics*, pages 108–122. Springer, 2007.

[7] Ali Berrichi, Farouk Yalaoui, Lionel Amodeo, and M Mezghiche. Bi-objective ant colony optimization approach to optimize production and maintenance scheduling. *Computers & Operations Research*, 37(9):1584–1596, 2010.

[8] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: An International Journal*, 8(2):239–287, 2009.

[9] Leonora Bianchi, Luca Maria Gambardella, and Marco Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In *Parallel Problem Solving from Nature PPSN VII*, pages 883–892. Springer, 2002.

[10] Mauro Birattari, Thomas Stützle, Luis Paquete, Klaus Varrentrapp, et al. A racing algorithm for configuring metaheuristics. In *GECCO*, volume 2, pages 11–18. Citeseer, 2002.

[11] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Sttzle. F-race and iterated f-race: An overview. In Thomas Bartz-Beielstein, Marco Chiarandini, Lus Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer Berlin Heidelberg, 2010.

[12] Norbert Blum. *A New Approach To Maximum Matching In General Graphs*. Springer, 1990.

[13] Eric Bonabeau and Christopher Meyer. Swarm intelligence: A whole new way to think about business. *Harvard Business Review*, 79(5):106–115, 2001.

[14] Juergen Branke and Jawad Asem Elomari. Meta-optimization for parameter tuning with a flexible computing budget. In *Proceedings of The Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, pages 1245–1252. ACM, 2012.

[15] Jürgen Branke and Michael Guntsch. New ideas for applying ant colony optimization to the probabilistic tsp. In *Applications of Evolutionary Computing*, pages 165–175. Springer, 2003.

[16] M. Contreras, W. Chung, and G. Jones. Applying ant colony optimization metaheuristic to solve forest transportation planning problems with side constraints. *Canadian Journal of Forest Research*, pages 2896–2910, 2008.

[17] Marco A Contreras, Woodam Chung, and Greg Jones. Applying ant colony optimization metaheuristic to solve forest transportation planning problems with side constraints. *Canadian Journal of Forest Research*, 38(11):2896–2910, 2008.

[18] ILOG Cplex. 11.0 users manual. *ILOG SA, Gentilly, France*, 2007.

[19] M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, Vol.1:28–39, 2006.

[20] M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, Vol. 43, no. 2:73–81, 1997.

[21] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, Vol. 1, no. 1:53–66, 1997.

[22] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems Man and Cybernetics*, Vol. 26, no. 1:29–41, 1996.

[23] Marco Dorigo and Mauro Birattari. Ant colony optimization. In *Encyclopedia of Machine Learning*, pages 36–39. Springer, 2010.

[24] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[25] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[26] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[27] Haibin Duan, Guanjun Ma, and Senqi Liu. Experimental study of the adjustable parameters in basic ant colony optimization algorithm. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 149–156. IEEE, 2007.

[28] W. J. Elliot, P.R. Robichaud, and R.B. Foltz. Erosion processes and prediction in nw u.s. forests. *International Symposium on Erosion and Landscape Evolution Hilton Anchorage Hotel, Anchorage, Alaska*, 2011.

[29] Daniela Favaretto, Elena Moretti, and Paola Pellegrini. On the explorative behavior of max–min ant system. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 115–119. Springer, 2009.

[30] R. B. Foltz, W. J. Elliot, and N. S. Wagenbrenner. Improving road erosion modeling for the lake tahoe basin and evaluating bmp strategies for fine sediment reduction at watershed scales. *Rocky Mountain Research Station, USDA Forest Service, Moscow, ID 83843*, 2010.

[31] Gianpiero Francesca, Paola Pellegrini, Thomas Stützle, and Mauro Birattari. Off-line and on-line tuning: a study on operator selection for a memetic algorithm applied to the qap. In *Evolutionary Computation in Combinatorial Optimization*, pages 203–214. Springer, 2011.

[32] Harold N Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *Journal of the ACM*, 23(2):221–234, 1976.

[33] Dorian Gaertner and Keith L Clark. On optimal parameters for ant colony optimization algorithms. In *IC-AI*, pages 83–89. Citeseer, 2005.

[34] L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, Vol.12(3):237–255, 2000.

[35] Luca Maria Gambardella and Marco Dorigo. Solving symmetric and asymmetric tsps by ant colonies. In *International Conference on Evolutionary Computation*, pages 622–627, 1996.

[36] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. An empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. In *Ant Colony Optimization and Swarm Intelligence*, pages 61–72. Springer, 2004.

[37] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. *European Journal of Operational Research*, 180(1):116–148, 2007.

[38] P. Gray. Exact solution of the fixed charge transportation problem. *Operations Research*, Vol. 19:1529–1538, 1971.

[39] F. Greulich. Transportation networks in forest harvesting: early development of the theory. *In: Yoshimura T (eds) Proceedings of International Seminar on New Roles of Plantation Forestry Requiring Appropriate Tending and Harvesting Operations held at Tokyo, Japan*, pages 57–65, 2002.

[40] Geoffrey M Guisewite and Panos M Pardalos. Minimum concave-cost network flow problems: Applications, complexity, and algorithms. *Annals of Operations Research*, 25(1):75–99, 1990.

[41] Walter J Gutjahr. S-aco: An ant-based approach to combinatorial optimization under uncertainty. In *Ant Colony Optimization and Swarm Intelligence*, pages 238–249. Springer, 2004.

[42] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*, volume 4. Springer-Verlag Berlin, 1985.

[43] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 1995. ACM.

[44] Warren M Hirsch and George B Dantzig. The fixed charge problem. *Naval Research Logistics Quarterly*, 15(3):413–424, 1968.

[45] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 186–202. Springer, 2010.

[46] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[47] J. M. Grace III. Factors influencing sediment plume development from forest roads. *In Environmental Connection '05: Proc. Conference*, pages 221–230, 2005.

[48] J. M. Grace III. Forest operations and water quality in the south. *Trans. ASAE 48(2): 871-880*, 2005.

[49] J. M. Grace III and B. D. Clinton. Protecting soil and water in forest road management. *Transactions of the ASABE*, Vol.50(5):1579–1584, 2007.

[50] Steffen Iredi, Daniel Merkle, and Martin Middendorf. Bi-criterion optimization with multi colony ant algorithms. In *Evolutionary Multi-Criterion Optimization*, pages 359–372. Springer, 2001.

[51] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 7(1):69–79, 1999.

[52] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[53] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[54] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.

[55] Madjid Khichane, Patrick Albert, and Christine Solnon. An aco-based reactive framework for ant colony optimization: First experiments on constraint satisfaction problems. In *Learning and Intelligent Optimization*, pages 119–133. Springer, 2009.

[56] Andreas Klose. Algorithms for solving the single-sink fixed-charge transportation problem. *Computers & Operations Research*, 35(6):2079–2092, 2008.

[57] Peter Korošec and Jurij Šilc. The multilevel ant stigmergy algorithm for numerical optimization. *Facta Universitatis-Series: Electronics and Energetics*, 19(2):247–260, 2006.

[58] K. Kowalski. On the structure of the fixed charge transportation problem. *International Journal of Mathematical Education in Science and Technology*, Vol.36:879–888, 2005.

[59] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[60] Ming Leng and Songnian Yu. An effective multi-level algorithm based on ant colony optimization for bisecting graph. In *Advances in Knowledge Discovery and Data Mining*, pages 138–149. Springer, 2007.

[61] Pengpeng Lin, Marco Contreras, Jun Zhang, and Woodam Chung. Applying ant colony optimization to solve constrained forest transportation planning problems. *Council on Forest Engineering Annual Meeting, Missoula, Montana*, 2013.

[62] Pengpeng Lin, Jun Zhang, and Marco A. Contreras. Applying pareto ant colony optimization to solve bi-objective forest transportation planning problems. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 385–392, Aug 2014.

[63] Pengpeng Lin, Jun Zhang, and Marco A. Contreras. Automatically configuring aco using multilevel paramils to solve transportation planning problems with underlying weighted networks. *Swarm and Evolutionary Computation*, (0):–, 2014.

[64] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[65] Manuel López-Ibáñez and Thomas Stützle. The impact of design choices of multiobjective antcolony optimization algorithms on performance: an experimental study on the biobjective tsp. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 71–78. ACM, 2010.

[66] Manuel Lpez-Ibez and Thomas Sttzle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569 – 582, 2014.

[67] Dorigo Marco and Gambardella Luca Maria. Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):215–218, 1997.

[68] George L Nemhauser and Laurence A Wolsey. *Integer and Combinatorial Optimization*, volume 18. Wiley New York, 1988.

[69] Andreas Noack and Randolf Rotta. Multi-level algorithms for modularity clustering. In *Experimental Algorithms*, pages 257–268. Springer, 2009.

[70] U. S. Palekar, M. H. Karwan, and S. Zionts. A branch and bound method for the fixed charge transportation problem. *Management Science*, Vol.36:1092–1105, 1990.

[71] Paola Pellegrini, Thomas Stützle, and Mauro Birattari. Off-line vs. on-line tuning: A study on max–min ant system for the TSP. In *Swarm Intelligence*, pages 239–250. Springer, 2010.

[72] Andreea Radulescu, Manuel López-Ibáñez, and Thomas Stützle. Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization*, pages 825–840. Springer, 2013.

[73] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, 1994.

[74] K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university timetabling problem. *in Proc. ANTS 2002, ser. LNCS, Springer Verlag*, Vol.2463, 2002.

[75] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[76] D. I. Steinberg. The fixed charge problem. *Naval Research Logistics Quarterly*, pages 217–235, 1970.

[77] T. Stützle. Local search algorithms for combinatorial problems: analysis, improvements, and new applications. *DISKI. Infix, Sankt Augustin, Germany*, Vol.220, 1999.

[78] T. Stützle and H. H. Hoos. The max-min ant system and local search for combinatorial optimization problems, kluwer academic publishers, dordrecht. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 137–154, 1999.

[79] T. Stützle and H.H. Hoos. Maxmin ant system. *Future Generation Computer Systems*, Vol.16:889–914, 2000.

[80] Thomas Stützle and Holger H Hoos. MAX–MIN ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[81] Thomas Stützle, Manuel López-Ibánez, Paola Pellegrini, Michael Maur, Marco Montes de Oca, Mauro Birattari, and Marco Dorigo. Parameter adaptation in ant colony optimization. In *Autonomous Search*, pages 191–215. Springer, 2012.

[82] Minghe Sun, Jay E Aronson, Patrick G McKeown, and Dennis Drinka. A tabu search heuristic procedure for the fixed charge transportation problem. *European Journal of Operational Research*, 106(2):441–456, 1998.

[83] Shang-Hua Teng. Coarsening, sampling, and smoothing: elements of the multilevel method. In *Algorithms for Parallel Processing*, pages 247–276. Springer, 1999.

[84] Virginia Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.

[85] Chris Walshaw. A multilevel approach to the travelling salesman problem. *Operations Research*, 50(5):862–877, 2002.

[86] Chris Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, 131(1-4):325–372, 2004.

[87] Mikhail Zaslavskiy, Francis Bach, and J-P Vert. A path following algorithm for the graph matching problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2227–2242, 2009.

[88] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *Evolutionary Computation, IEEE Transactions on*, 3(4):257–271, 1999.

[89] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, 2003.

Vita

Personal Data:

Name: Pengpeng Lin

Place of Birth: Zhengzhou, Henan, China

Educational Background:

- Master of Science in Computer Science, University of Kentucky, USA 2013.
- Master of Science in Computer Technology, Western Kentucky University, USA 2009.
- Bachelor of Engineering in Computer Science and Technology, Xian Institute of the PLA Academy of Telecommunications, China 2004.

Professional Experience:

- Student Teaching Assistant, 01/2010 - 05/2015. Department of Computer Science, University of Kentucky, Lexington, KY, USA.
- Graduate Research Assistant, 08/2007 - 05/2009. Department of Computer Science, Western Kentucky University, Bowling Green, KY, USA.
- Network Administrator, 2002 - 2004. Xian Institute of the PLA Academy of Telecommunications, China.

Awards:

- Harrison D. Brailsford Graduate Fellowship, University of Kentucky (2014-2015).
- Student Travel Grant Recipient of IRI IEEE International Conference (2014).

- Student Research Travel Award, University of Kentucky (2014).
- Research Conference Travel Award, Wallace Forestry Endowment (2013).
- ACM Outstanding Teaching Assistant Nominee, University of Kentucky (2012).
- Student Research Travel Award, University of Kentucky (2012).
- Student Research Travel Award, University of Kentucky (2011).
- Graduate Non-Res Scholarship, Western Kentucky University (2008 - 2009).
- Graduate Tuition Scholarship, Western Kentucky University (2008 - 2009).
- Wurster, Robert J. Scholarship, Western Kentucky University (2008).

Publications:

JOURNAL

Under Review

1. **Lin, Pengpeng**, Jun Zhang, Marco A. Contreras. "A multilevel ACO approach for solving forest transportation planning problems with environmental constraints." Journal of Heuristics, Springer, (Under review).
2. **Lin, Pengpeng**, Jun Zhang, Marco A. Contreras. "Using ant colony optimization for solving constrained forest transportation planning problems." Natural Computing, Springer, (Under review).

Peer Reviewed

1. **Lin, Pengpeng**, Jun Zhang, Marco A. Contreras. "Automatically configuring ACO using multilevel ParamILS to solve transportation planning problems with underlying weighted networks." Swarm and Evolutionary Computation, Elsevier, Vol. 20, (2014): 48-57, ISSN 2210-6502.

2. Wang, Xiwei, Jun Zhang, **Pengpeng Lin**, Nirmal Thapa, Yin Wang and Jie Wang. "Incorporating auxiliary information in collaborative filtering data update with privacy preservation." International Journal of Advanced Computer Science and Applications Vol. 5, no. 4, (2014): 224-235.
3. **Lin, Pengpeng**, Nirmal Thapa, Ingrid St Omer, Lian Liu, and Jun Zhang. "Feature Selection: A preprocess for data perturbation." IAENG International Journal of Computer Science Vol. 38, no. 2 (2011): 168-175.

CONFERENCE

1. **Lin, Pengpeng**, Jun Zhang, Marco Contreras."Applying Pareto ant colony optimization to solve bi-objective forest transportation planning problems." In Proceedings of 15th International Conference on Information Reuse and Integration (IRI), IEEE, pp. 795-802. California, USA. 2014.
2. **Lin, Pengpeng**, Jun Zhang, Ran An. "Data dimensionality reduction approach to improve feature selection performance using sparsified SVD." In Proceedings of International Joint Conference on Neural Networks (IJCNN), IEEE WCCI, pp. 1393-1400. Beijing, China. 2014.
3. **Lin, Pengpeng**, Marco Contreras, Jun Zhang, and Woodam Chung. "Applying ant colony optimization to solve constrained forest transportation planning problems." In Proceedings of Council on Forest Engineering (COFE), University of Montana MT, USA. 2013.
4. **Lin, Pengpeng**, Jun Zhang, Xiwei Wang, and Art Shindhelm. "Simultaneous pattern and data hiding in supervised learning." In Proceedings of 13th International Conference on Information Reuse and Integration (IRI), IEEE, pp. 385-392. Nevada, USA. 2012.
5. Thapa, Nirmal, **PengPeng Lin**, Lian Liu, Jie Wang, and Jun Zhang. "Constrained nonnegative matrix factorization based data distortion techniques-

study of data privacy and utility." In Proceedings of International Conference on Data Management Technologies and Applications (DATA), pp. 51-56. Rome, Italy. 2012.

6. Thapa, Nirmal, Lian Liu, **Pengpeng Lin**, Jie Wang, and Jun Zhang. "Constrained nonnegative matrix factorization for data privacy." In Proceedings of the International Conference on Data Mining (DMIN), pp. 88-94. Nevada, USA. 2011.

7. **Lin, Pengpeng**, Ingrid St Omer Jun Zhang, Huanjing Wang, and Jie Wang. "A comparative study on data perturbation with feature selection." In Proceedings of the IAENG International Conference on Data Mining and Applications (ICDMA). pp. 454-459. Hongkong, China. 2011.

8. **Lin, Pengpeng**, Huanjing Wang, and Taghi M. Khoshgoftaar. "A novel hybrid search algorithm for feature selection." In Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 81-86. Boston, USA. 2009.