Prildy

Sarjana Komputer

A Hybrid Constrained Ant Colony Approach to the Public Bus Assignment Problem with Time Windows

Dissertação para obtenção do Grau de Mestre em

Lógica Computacional

Orientador: Pedro Barahona, Prof. Catedrático,
Universidade Nova de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2016

A Hybrid Constrained Ant Colony Approach to the Public Bus Assignment Problem with Time Windows

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2]. [1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

To my Mom and Dad.

Acknowledgements

First of all I want to say my humble gratitude to my adviser Professor Pedro Barahona for his patience and continuous support throughout this thesis work. He is always available for a discussion and enlighten my work. Without his guidance it will be really difficult for me to finish this thesis.

Next, I want to thanks Erasmus Mundus to give me the opportunity to study Computational Logic in three different countries. Throughout this master study, I acquire not only knowledge in Computational Logic but also a lot of important life experiences.

Thanks to my EMCL colleagues, especially those who stayed in Lisbon. We shared a lot of moments together, from a group study in most vibrant neighborhood in Lisbon to arranging an EMCL workshop. We share not only pleasant but also stressful moments together.

To my college friends: Philip Arthur, Venezia Ryanka Sutrisno, Shieron Antolis, Christian, David, and Nathan. Thanks for the silly chat group that always help me to escape the stress and relax.

To my high school friends: Megaputra Dirgantara, Jonathan Saputra and Prima Trisnawan Suwandana. Thanks for being there to share our life experiences in both happy and sad moments.

To my DOTA friends: Steven Limanto, Satyadharma Tirtarasa, Albert Koesnaidi and Radityo Eko Prasojo. Thanks for the game that always help me get out of my depression. Even I never meet some of you in real life, it is always a pleasure playing a game with you.

To my Indonesian friends in each country I lived in. Thank you for creating a home-like atmosphere in the country far away from Indonesia. It always feels like in Indonesia when I am with you.

My deepest thanks to my family, especially both of my parent, for their continuous mental support during my study. Without their support I will not be able to hold on to finish this master.

Abstract

Public transportation is an important aspect in everyday's life, and a well scheduled public transportation is a desired component in city management. In this thesis we address the Public Bus Assignment Problem, a variant of the Vehicle Scheduling Problem with some additional constraints, aiming at obtaining a bus assignment which minimizes operational cost, satisfying the problem constraints.

In order to improve the quality of the scheduling we exploit a time window feature: starting from an initial fixed schedule, we allow the departure times of the buses to lie within variable-sized time windows around the initial schedule. To solve the main problem we adopted a nature-inspired meta-heuristic technique, Ant Colony Optimization and, to take into account the time windows and guarantee the feasibility of the obtained solutions, we hybridize it with Constraint Programming. More specifically, we interleave the constructive step of our approach with constraint propagation, and add a final step, using Constraint Programming, to produce a fixed schedule that minimizes the distance between the obtained and initial schedules.

We conduct our experiments using various problem instances with several number of lines and various widths for the time windows. The results obtained show that our hybrid Ant Colony Optimization / Constraint Programming solution outperforms other possible constructive alternatives, using greedy and stochastic approaches. Our experiments also show that by introducing time windows the operational costs can be significantly reduced with an acceptable difference to the initial schedules.

Keywords: Public Bus Assignment Problem, Ant Colony Optimization, Constraint Programming, Hybrid ACO-CP, Time Window, Vehicle Scheduling Problem

Resumo

Os transportes públicos constituem uma componente relevante da vida de todos os dias, e um transporte público bem programado é um objectivo importante na gestão de uma cidade. Nesta tese abordamos o Problema de Atribuição de Autocarros ("Public Bus Assignment Problem"), uma variante do Problema de Alocação de Veículos ("Vehicle Scheduling Problem") com algumas restrições adicionais, que visa a obtenção de uma atribuição de autocarros que minimize o custo operacional, satisfazendo as restrições do problema.

A fim de melhorar a qualidade da solução esta dissertação considera a possibilidade de se explorarem janelas temporais: a partir de um horário fixo inicial, permitimos que os horários de partida dos autocarros possam cair dentro de janelas de tempo de tamanho variável em torno do horário inicial. Para resolver este problema foi adoptada uma meta-heurística inspirada na natureza, Optimização por Colónias de Formigas ("Ant Colony Optimization") que, para ter em conta as janelas de tempo e garantir a viabilidade das soluções obtidas, foi hibridizada com Programação por Restrições. Mais especificamente, na fase de construção de uma solução, a atribuição de valores é entremeada com a propagação de restrições, sendo ainda adicionada uma etapa final, usando programação de restrições, para produzir um horário fixo que minimiza a distância entre os horários obtido e inicial.

Para validação, foram realizadas experiências usando várias instâncias de problemas com um número variável de linhas e com diferentes larguras das janelas temporais. Os resultados obtidos mostram que a nossa abordagem híbrida de Optimização por Colónias de Formigas / Programação por Restrições supera outras alternativas construtivas possíveis, usando abordagens gananciosas ("greedy") e estocásticas. As nossas experiências mostram igualmente que com a introdução de janelas de tempo os custos operacionais podem ser significativamente reduzidos com uma diferença aceitável entre os horários obtidos e iniciais.

Palavras-chave: Problema de Atribuição de Autocarros, Optimização por Colónias de Formigas, Programação por Restrições, híbrida ACO-CP, Janelas Temporais, Problema de Alocação de Veículos

Contents

List of Figures

List of Tables

Acronyms

ACO Ant Colony Optimization.

ACS Ant Colony System.

ACSsr Ant Colony System with Stochastic Ranking.

AS Ant System.

COP Combinatorial Optimization Problem.

CP Constraint Programming.

CPO CP Optimizer.

CPO-ACO CP Optimizer - Ant Colony.

CSOP Constraint Satisfaction and Optimization Problem.

CSP Constraint Satisfaction Problem.

GA Genetic Algorithm.

IDE Integrated Development Environment.

LS Local Search.

MDVSP Multiple Depot Vehicle Scheduling Problem.

MIS Maximum Independent Set.

MKP Multidimensional Knapsack Problem.

MMAS Max Min Ant System.

PBAP Public Bus Assignment Problem.

PBU Probability Based Utility.

QAP Quadratic Assignment Problem.

SA Simulated Annealing.

SR Stochastic Ranking.

TS Tabu Search.

TSP Traveling Salesman Problem.

VSP Vehicle Scheduling Problem.

Chapter 1

Introduction

1.1 Background and Scope

Public transportation is an important aspect in everyone's life, people need it to commute easily from one place to another. Making a good public transportation schedule can be a challenging problem to company management, aiming to keep operational cost low while maintaining good quality service.

The scheduling problem of public transportation is categorized as a Combinatorial Optimization Problem (COP) as the task is to find an assignment of vehicles that minimizes the operational cost. Additionally if certain constraints need to be satisfied the problem is then categorized as a Constraint Satisfaction and Optimization Problem (CSOP). Most instances of this type of problems are *NP-hard* [10]. It is widely known that it is not possible, in general, to solve *NP-hard* problems in polynomial time, unless $P = NP$. However the demand of public transportation system is indeed to produce good schedules and assignments in reasonable time.

Two approaches, to address COPs and CSOPs, are *complete search* method and *approximative* method. Complete search methods guarantee finding optimal solutions to the problem, but they take unacceptable computing time as the instances scale up. In contrast, approximative methods do not guarantee optimal solutions but return hopefully good solutions in acceptable time. Additionally, approximative methods handle constraints satisfaction indirectly, relaxing constraints into optimization components which may be unsatisfactory in some applications.

The hybrid approach between complete search and approximative method is employed to give a better solution. In a hybrid approach, complete search methods can be employed as a look ahead mechanism to narrow the search space of approximative method. In particular this thesis will focus on hybridizing an Ant Colony Optimization

metaheuristic as an approximative method with constraint propagation as a complete search method.

Specifically we address a problem being faced by public transportation companies namely the Public Bus Assignment Problem (PBAP). This problem can be considered as a variant of Multiple Depot Vehicle Scheduling Problem (MDVSP) with additional constraints. In this thesis we consider our PBAP as CSOP with finite domain. The problem is to find an assignment of bus/vehicle to each departure points which optimize operational cost. This problem has been addressed before by using similar ACO and Constraint Programming (CP) approach with parallel computation [31], but there are several differences between our problems (as detailed in section 2.6).

Importantly, we investigate the advantage of using time windows to enhance the schedule. Time windows are acquired from a previously fixed departure time expanded by some adequate slack. The hypothesis to be tested is whether allowing vehicle to depart slightly off ideal departure time, a better schedule can be produced.

The main goal of this thesis work is to address PBAP with a hybrid technique between ACO and CP. Since the problem has not been much addressed before, we do not compare our approach with state of the art. Instead we propose to model the problem using time window.

The assessment of resulting technique is done using artificial instances, somewhat similar to real life problem. Several key features of our instances are multiple depot, line exchange and star shaped network topology.

1.2 Problem definition

PBAP, as defined in [31], is a constraint satisfaction and optimization problem arising in public transportation company to assign vehicles such that each trip is covered by one vehicle and the operational cost is minimized, whilst satisfy some set of constraints. Our problem is different from the one defined by [31], the main difference being that we consider homogeneous fleet and time windows. The problem that we tackle is formalized as follow:

Let $L = \{L_1, L_2, \ldots, L_l\}$ be a set of lines and $T = \{T_1, T_2, \ldots, T_{2l}$ be a set of terminals. Each line consist of two terminals such that terminals T_{2i-1} and T_{2i} belong to line L_i. Each terminal has k departure points so that the total number of departure points denoted by $P = \{P_1, P_2, \ldots, P_n\}$, is $n = 2lk$. Let low_i and up_i respectively denote the lower bound and upper bound of a time window for point of departure P_i. Let $D = \{D_1, D_2, \ldots, D_m\}$ be a set of m depots. Location index loc_j is defined for each terminal and depot $j \in \{1, 2, \ldots, 2l+m\}$. $loc_1, \ldots, loc_m$ are associated to location index for depots $D_1, \ldots, D_m$ and $loc_{m+1}, \ldots, loc_{2l+m}$ are associated to location index for terminal $T_1, \ldots, T_{2l}$. A distance matrix $dist$ with size $2l + m \times 2l + m$ denotes the distance between two locations. A time metric is used for the distance.

A trip trp_k is then defined for each point of departure P_k such that $start_k$ and end_k denote start and end location of the trip respectively. $start_k$ will be defined as the location index of the terminal in which point of departure P_k is located while end_k will be defined as the location index of the other terminal, from same line. All trip are required to be scheduled between a period of time s_k and e_k. In particular, all trip are also required to depart within the time window of point of departure $low_k \leq P_k \leq up_k$ where P_k denotes actual departure time of trp_k.

A vehicle v_i, where $i \in \{1, \ldots, nVehicle\}$, denotes the bus assigned for the trip. For each vehicle v_i, associated to a depot D_j, various state information is maintained during the search process, namely:

- $vLoc_i$ denotes current location of v_i
- $vDepot_i$ denotes the depot location index associated to v_i
- $vLow_i$ denotes the current lower bound time of v_i
- vUp_i denotes the current upper bound time of v_i
- $vTotalDist_i$ denotes the total distance traveled so far by v_i
- $vBtd_i$ denotes how many times v_i went back to depot
- $vLtrf_i$ denotes how many times v_i changed line

Additionally there are limitations regarding the stay on a terminal. $maxBusInTerminal$ denotes limitation on the number of vehicle staying at each terminal and, if this is exceeded, the earliest vehicle to arrive should be sent back to depot. $maxWaitTime$ denotes a limitation on the time a bus stays idle at a terminal and, if exceeded, the vehicle will be sent back to depot.

At the end of the process it is required that an assignment of vehicles is made such that these condition are fulfilled:

- All trip / departure point are covered by one and only one vehicle.
- Each vehicle travels through point of departures starting and returning to depot at the end of process.
- The objective function (operational cost) is minimized

Each problem instance is considered to cover a period of one day. Furthermore, line transfers are allowed, so that a vehicle can go from one terminal to other terminal in another line to serve a specific point of departure there. A vehicle can also go back to its depot, but by doing so the operational cost may increase.

The objective function F, which to be minimized, is defined over a solution sol. Let B be a set of indexes of vehicle used in sol.

$$F(sol) = \sum_{i \in B} ((vTotalDist_i * cpm) + cpd) \tag{1.1}$$

where cpm denotes cost to use bus per minute and cpd denotes cost to use a bus.

Based on a review by [22], a special case of Vehicle Scheduling Problem (VSP) can be solved in polynomial time when single depot is considered and each trip has same start time and end time $s_k = e_k$. However for other instances of VSP, namely with multiple depot, ($m \geq 2$) the problem becomes NP-Hard [1].

1.3 Contribution

Our work has the following contributions:

1. The study of time window effect on reducing operational cost of PBAP.

2. Proposing new ACO pheromone matrix.

The first contribution is made by adopting time window to a previously fixed schedule. Time window is specified by expanding the ideal departure time by some adequate slack. More formally we define the time window as the ideal departure time plus and minus slack $low_i = ideal_i - slack_i^{low}$ and $up_i = ideal_i - slack_i^{up}$ (where $ideal_i$ denotes ideal departure time of point of departure i). Both slacks are measured according to the gap between two consecutive departure points on the same terminal. $slack_i^{low}$ is calculated by the gap between previous and current departure point. $slack_i^{up}$ is calculated by the gap between current and following departure point. A constant percentage is used to adjust the width of slacks. By adjusting this departure time we expect some impacts. Firstly if we adjust the departure time as early as possible the bus might wait some more time at the next terminal. Secondly if we adjust the departure time as late as possible the bus might arrive too late to serve expected departure point in next terminal. We observe the effect of adapting this time window to the model and solve it with an hybrid ACO and CP technique. The goal is to investigate whether, and by how much, the cost is reduced.

Our second contribution is to propose a new ACO pheromone matrix. The pheromone is an important implementation aspect of the ACO method. Different pheromone matrices produce different results. Previously [31] proposed a pheromone matrix based on locations and point of departures. We tried to implement ACO using this type of matrix which is very large for big instances. Therefore we propose new pheromone matrix made between locations and locations. This matrix is not only smaller than previous one, but also improve the performance of ACO by providing more meaningful statistics and avoiding overfitting in the learning process.

1.4 Document Organization

This thesis is organized as follow.

1. Background of topics and some related research will be presented in chapter 2. This includes Constraint Programming as complete search method, Metaheuristic as approximative method, hybrid approach, and Public Bus Assignment Problem.

2. Our approach will be presented in chapter 3. This includes how we model the problem, implementation of proposed method, and parameters we used in our approach.

3. Experimental result will be presented in chapter 4. This includes how we artificially construct the benchmark and fine tune the search parameters, reporting a comparison with a greedy approach, the impact of using different width of time windows, as well as a different pheromone matrix.

4. The main conclusions will be presented in chapter 5.

CHAPTER 2

RELATED WORK

This chapter summarizes some theoretical concepts and techniques that are related to our approach. We start by classifying some problems in section 2.1. In section 2.2 we overview the complete search approach of constraint programming. Then in section 2.3 we address some local search based Metaheuristic approaches. A population based metaheuristic, which is also used in our approach, is presented in section 2.4. The state of the art of the hybrid approach is summarized in section 2.5. Finally, the related work regarding the problem we address is discussed in section 2.6.

2.1 Problem Termination

Before going further into specific algorithm or technique we give a brief description of several types of problems, namely Combinatorial Optimization Problems, Constraint Satisfaction problems, and Constraint Satisfaction and Optimization Problems. These problems will be addressed by techniques that we will describe in this chapter.

2.1.1 Constraint Optimization Problem

The goal of a COP is to maximize or minimize an objective function. Let f be the objective function defined as follow

$$f : \mathfrak{X} \leftarrow \mathbb{R} \tag{2.1}$$

where $\mathfrak{X}$ denotes search space. If the goal is to maximize then we need to find a solution vector s such as

$$s = \arg\max_{x \in \mathfrak{X}} f(x) \tag{2.2}$$

if the goal is to minimize then it formalized as follow

$$s = \arg\min_{x \in \mathfrak{X}} f(x) \tag{2.3}$$

2.1.2 Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) **P** can be defined as combinatorial problem that aims to find a solution that satisfies all constraints of the problem. Formally a CSP **P** [29] is defined as a tuple $< X, D, C >$ where:

1. X is an n-tuple variable $X =< x_1, x_2, \ldots, x_n >$
2. D[1] is a corresponding n-tuple domain $D =< D_1, D_2, \ldots, D_n >$ such that $x_i \in D_i$
3. C is a set of constraints $C = \{C_1, C_2, \ldots, C_t\}$

A constraint C_j is a pair $< R_{S_j}, S_j >$ where R_{S_j} is a subset of Cartesian product of the domain of the variables in S_j (the scope of the constraint). Finally a valid solution to **P** is an n-tuple $A =< a_1, a_2, \ldots, a_n >$ where $a_i \in D_i$ and each C_j is satisfied, in that R_{S_j} holds on the projection of A onto the scope S_j.

2.1.3 Constraint Satisfaction and Optimization Problem

A CSOP is a combination of a CSP and a COP so as to optimize an objective function f but also to satisfy the set of constraints C. Assuming the goal is to minimize, one can formalize a CSOP **P** as tuple $< X, D, C, F >$ where:

1. X is an n-tuple variable $X =< x_1, x_2, \ldots, x_n >$
2. D is a corresponding n-tuple domain $D =< D_1, D_2, \ldots, D_n >$ such that $x_i \in D_i$
3. C is a set of constraints $C = \{C_1, C_2, \ldots, C_t\}$
4. F is an objective function defined over the variables.

An optimal solution to CSOP **P** is an n-tuple assignment $A =< a_1, a_2, \ldots, a_n >$ where $a_i \in D_i$ and F is optimized subject to all constraint in C.

2.2 Constraint Programming

According to [29], CP is a powerful paradigm for solving combinatorial search problem. The term programming in CP has two interpretations [29]: mathematical and computer. Mathematically, a CP user declaratively states a set of constraints over a set of decision variables. In the computational interpretation, a user needs to specify a search strategy to achieve the solution. In the early days, from 1965 to 1985, the constraint satisfaction paradigm became popular amongst research in artificial intelligence and computer science. Two streams of work [29] were developed on this topic: language stream and

[1] In this thesis we are interested in finite domain problems where the domains D_i have a limited number of discrete values

algorithm stream. In the language stream, constraints were applied to a variety of domains and associated programming languages and systems. Consequently this stream focused on developing programming languages and libraries. On the other hand, from the machine vision community interest in constraint solving algorithms emerged an algorithm stream, focused on algorithms and heuristics. Finally both merged to form the CP community in the 1990's.

One of the developments of CP is to solve an optimization problem, i.e. COP. The main idea of using CP to solve COP is to model the problem as a constraint problem and then finding the optimal solution by exploring the entire search space (complete search). This kind of problem is then called CSOP. Since the entire search space is explored, an optimal solution can be guaranteed. However the drawback of this approach is the computation time when dealing with large instances.

In order to solve CSP one will need to specify reasoning algorithm. The reasoning algorithm is categorized into: inference, search and combination of those two. If we are dealing with finite domain we can define a search space ω. Inference is defined as an approach to eliminate subspaces from ω through local constraint propagation. This elimination will remove subspaces which are not possible to be solution. Search is defined as systematical exploration of ω. The detail of each strategy will be explained in the next sections.

2.2.1 Network Consistency as Inference

The background of inference approach is pathological *thrashing* behavior in backtracking process. Thrashing can be defined as a repeated exploration of failing subtrees on a backtrack search tree which are essentially identical [29]. Thrashing can cause ineffective exploration (search) process. The inference approach to identify and eliminate thrashing behavior, by tightening the constraints and making implicit constraints explicit, is popularly known as network consistency algorithm. The naming comes from the analogy of CSP with a graph network system, variable as node/vertex, and constraint as arc/edge.

Several basic network consistency algorithms have been proposed, namely: node consistency, arc consistency, path consistency, k-consistency, and bounds-consistency, described in the following sections. As stated before, we will assume that we are dealing with finite domain constraints, which for simplicity are assumed as unary or binary constraints. The formalization of CSP $\mathbf{P} =< X, D, C >$ will be as defined at 2.1.2. Unary constraint C_i can be formally defined as $C_i =< R_{<x_i>}, < x_i >>$. Shorthand notation R_i will be used for $R_{<x_i>}$. Binary constraint C_s can be formally defined as $C_s =< R_{<x_i,x_j>}, < x_i, x_j >>$ where $i \neq j$. Shorthand notation R_{ij} will also be used for $R_{<x_i,x_j>}$

2.2.1.1 Node Consistency

Node consistency requires that all unary constraints R_i over a variable are satisfied by all values in the domain of their variables. Formally, variable x_i is node consistent if and

only if $D_i \subseteq R_i$. A variable x_i, usually called node i, can be enforced into node consistent by using procedure

$$D_i \leftarrow D_i \cap R_i \tag{2.4}$$

A network is node consistent if and only if all its node are consistent.

2.2.1.2 Arc Consistency

Consider nodes i and j with domains D_i and D_j. When there is a constraint relation R_{ij} we consider a directed arc $< i, j >$. Note that arc $< i, j > \neq < j, i >$. Arc consistency requires that for every member of D_i there exists at least an element in D_j that satisfies R_{ij}. Formally arc $< i, j >$ is arc consistent if and only if

$$D_i \subset \kappa_i(R_{ij} \otimes D_j) \tag{2.5}$$

where κ is projection function and $\otimes$ is the Cartesian product operator. To test or enforce arc consistency of arc $< i, j >$ one can apply the following procedure

$$D_i \leftarrow D_i \cap \kappa_i(R_{ij} \otimes D_j) \tag{2.6}$$

A network is arc consistent if and only if it is node consistent and all its arcs are arc consistent. Initial algorithms to enforce arc consistent to a network are AC-1, AC-2, and AC-3, first detailed in [24]. More recent algorithms include AC-4, AC-4 and AC-6, but AC-3 is still widely used given its simplicity.

2.2.1.3 Path Consistency

Consider a path of length two from i to j through m as path $< i, m, j >$. Path consistency requires that for every pair of values $< a, b >$ allowed by relation R_{ij} there is a value c for variable x_m such that $< a, c >$ and $< c, b >$ are allowed by relation R_{im} and R_{mj} respectively. Formally it can be defined as follow

$$R_{ij} \subset \kappa_{ij}(R_{im} \otimes D_m \otimes R_{mj}) \tag{2.7}$$

Furthermore path $< i, m, j >$ can be check or enforced by procedure as follow

$$R_{ij} \leftarrow R_{ij} \cap \kappa_{ij}(R_{im} \otimes D_m \otimes R_{mj}) \tag{2.8}$$

A network is path consistent if and only if it is arc consistent and all its paths are consistent. Montanari [27] shows that if all paths with length two are path consistent then the network is path consistent. Therefore it is not necessary to check the consistency of another paths with longer length. Several well known algorithms to ensure path consistency to entire network are PC-1 and PC-2. They are detailed in [24, 27].

2.2.1.4 k-Consistency

Freuder [9] introduces the term of *k-consistency*. This consistency requires that given consistent valuation if any set of $k-1$ variables, one can extend it with the value of some k-th variable such that all k values is consistent. Thus node consistency, arc consistency and path consistency are special case of k-consistency, respectively 1-consistency, 2-consistency, and 3-consistency.

2.2.1.5 Bounds Consistency

Bounds consistency exploits the fact that finite integer domains are composed of integers, that have a total ordering and two particular values $min(D_i)$ and $max(D_i)$. There exist several versions of bound consistency. We will describe one which will be used in later chapter of this thesis work, defined by Van Hetenryck et al. [14] as follows:

> A constraint c is *bound-consistent* with respect to $D_1, D_2, \ldots, D_n$ if, for each variable x_i and value $v_i \in \{min(D_i), max(D_i)\}$, there exist values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$ in $D_1^*, \ldots, D_n^*$ such that $c(v_1, \ldots, v_n)$ holds.

where D^* denotes the set $min(D) \ldots max(D)$. A network is bounds consistent if and only if all its constraints are bound consistent.

2.2.2 Search

Search, as a solution finder, is the final step in CP to find a solution. CP, in general, employs *backtrack* to do complete search through the search space. This means such search will guarantee a solution if one exists. Backtrack search can be defined as constructive solution search, completing partial solution by assigning values to variables not included in the partial solution until finding a solution or reaching a dead end. If a dead end is reached, the last choice made is undone and another assignment is tried. The search will be done systematically so that all possible assignments are explored. In constraint programming, backtrack search interleaves constraint propagation every time an assignment is made. The propagation will perform some look ahead to narrow possible values of unassigned variables. If upon propagation a variable has no possible value, then backtrack will occur.

Another component of the CP search mechanism is a heuristic. Good heuristic strategy may significantly reduce the search space thus making the search process faster. Typically in CP search, variable selection and value selection heuristics are used. *Variable selection* heuristic will be in charge of selecting the next variable to be assigned in the search. *Value selection* heuristic will be in charge of selecting which value to be assigned first.

2.3 Metaheuristic

Meta-heuristic was first introduced by Glover [12] as an upgrade of the heuristic approach. Nowadays, the word metaheuristic [11] is also defined as the solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. The heuristic itself has been used a long time before to generate optimum or near optimum solution to hard optimization problem. The work of Lin and Kernighan [23] is one of many researches that lead to modern heuristic. The main goal of metaheuristic as well as approximation algorithm is to generate the best solution possible while maintaining the solution to satisfy certain properties. It was believed that constant ratio polynomial time approximation algorithm does not exist for many practical problems [30]. Metaheuristic appear to be the answer of the problem. Although metaheuristic do not guarantee that near optimum solution can be achieved in short time, it does find near optimum solutions for many practical problem instances. Consequently, it is believed that metaheuristic can be applied to a wide range of problem instances.

One of the most natural ways to find an optimal or suboptimal solution to a problem is probably Local Search (LS) [13]. The main idea of LS is building a solution and then gradually improve it until no further improvement is possible. Local search has quite a long history. The development started from simple iterative improvement to complex metaheuristic algorithms such as tabu search, genetic algorithm and simulated annealing.

Tabu Search (TS) is arguably the first metaheuristic. It is originally introduced in the same paper as the term metaheuristic is introduced [12]. The basic idea of tabu is a combination of LS with adaptive memory. The main important component of TS is a tabu list as the adaptive memory. Tabu list records all previously visited neighbors. This list is used to prevent the algorithm from returning to a previously visited neighbor. TS enable escaping local optima by allowing a non-improving moves.

Simulated Annealing (SA) is a randomized local search metaheuristic introduced independently by Kirkpatrick, et al. [21] and Černỳ [3]. The algorithm itself is inspired by the annealing process of solid compounds. In the annealing process the compound is heated at some high temperature until the solid melts into liquid. After that, the temperature is slowly decreased at a constant rate so that the thermal equilibrium is reached at each temperature. Decreasing the temperature while slowly cooling down the compound will finally form the most regular possible solid compound (with minimum energy state) that is free of defect. Analogically in SA an initial solution is generated and in each iteration a perturbation is performed. The perturbation is performed in such a probabilistic way both to acquire the neighborhood and to decide whether such *hill-climbing* step is accepted. Hill-climbing step is a solution candidate which has worse objective value than current best solution. The probability to accept this hill climbing step is highly affected by a variable called *temperature parameter*. As this temperature parameter decreases, the hill climbing step will be less likely to be accepted. This hill-climbing step is an important

way to escape local minima.

Genetic Algorithm (GA) was firstly introduced by Holland in 1975, and was re published in a 2nd edition of his book [15]. This algorithm is inspired by biological selective breeding. To have a desirable characteristic on a descendant, parents chromosome combination is one of the most important things to be engineered. Previous similar work is evolutionary strategy. GA differs from evolutionary strategy in the way that it uses population and a distinctive problem representation.GA use a separate problem representation from the actual variable. Variable vector (phenotype) is associated with a string (genotype) called *chromosome*. Each element of the string is called a *gene* and the values those genes can take are called *allele*. It is believed that a binary string is an optimal way to represent the problem. Basically some important components of GA are *selection*, *crossover*, and *mutation*. Selection is used to determine a chromosome candidate to breed in next iteration. Crossover and mutation operator are mainly used to do chromosome recombination to obtain new chromosomes. Crossover is basically done by replacing some genes in one parent by corresponding genes of the other. Mutation is done by randomly chose genes of a chromosome and change the allele values. Originally GA was introduced as an adaptive system broadly applicable to many problem domains. However in the development, GA has been mostly used as a function optimizer. A paper by De Jong [18] emphasizes this matter so one does not equate GA to a function optimizer.

2.4 Ant Colony Optimization

ACO is a constructive metaheuristic inspired by the behavior of ants in their colony. The technique is originally discovered in [5, 7]. The main idea came by observing the behavior of ants when they travel with their colony in search for food, each ant communicating with others by using pheromone trails deposited in every step they take. The scent of the pheromone gives the ant the idea of the correct path. The shorter path will lead to faster travel time of the ants, hence make the ants come back faster than other ants using the longer paths. Consequently pheromone on the shorter path will accumulate faster than in other paths. Finally the shortest path can be detected by the strongest scent of the pheromone.

The pheromone mechanism illustrated in figure 2.1 shows how the pheromones are used by the ants to find the shortest path. After arriving at the decision point without pheromone (figure 2.1-a) the ants will choose randomly which path they will take (figure 2.1-b). Since the ants travel at the same speed, the ants on shorter path (lower path) will arrive at the destination faster than the ants on other path (figure 2.1-c). As a result, the pheromone (dashed line) will accumulate faster in the shorter path (figure 2.1-d).

Each artificial ant in ACO practically produces a solution stochastically. The solution is built incrementally by adding solution component to the partial solution in each iteration. The solution component to add is selected stochastically by considering heuristic information of the problem and pheromone trails that will be changed dynamically

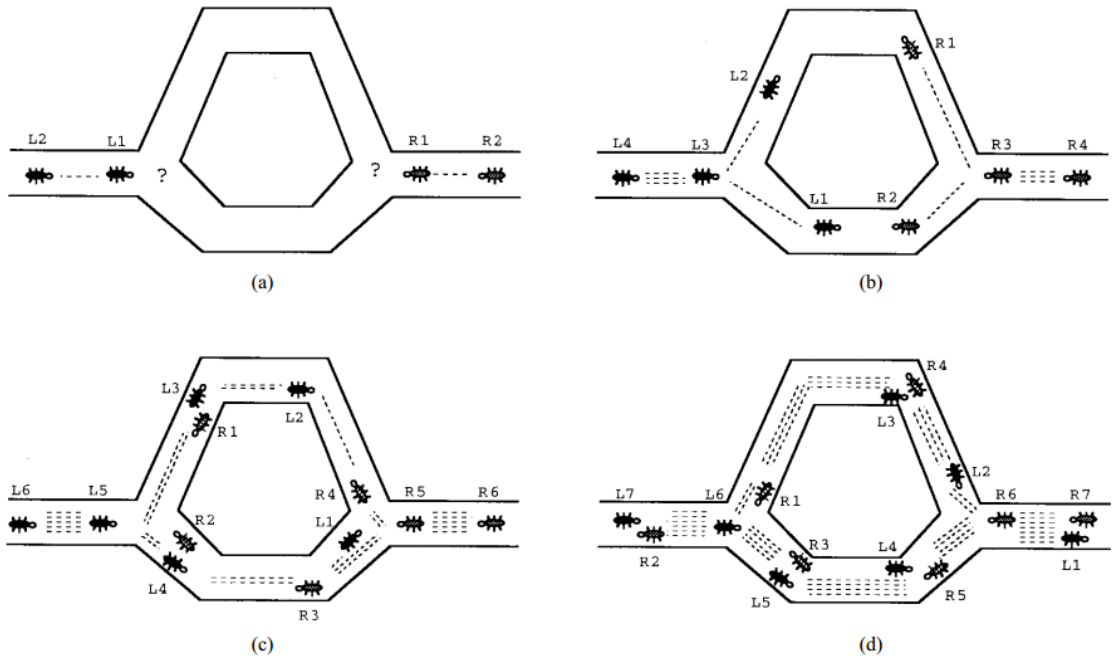


Figure 2.1: Ant Colony illustration [6]

throughout each run.

The outline of ACO is given in algorithm 1. In the *Initialization* step all variables, especially the pheromone variable, is set to their initial value. The *ConstructAntSolution* procedure consists of N simpler procedures, each constructing a complete candidate solution. Once a complete candidate solution is acquired, the procedure can optionally use *ApplyLocalSearch* to improve the result. Finally *UpdatePheromones* is applied so that solution components from good solutions are more favored to ants in subsequent iterations. The pheromone variable is modified by pheromone deposit and pheromone evaporation. Pheromone deposit increases the pheromone value of solution component from good solution while pheromone evaporation decreases the value of pheromone over time.

Algorithm 1 Ant Colony Optimization [11]

```
procedure ACO Algorithm
    Initialization
    while termination condition not met do
        ConstructAntSolution
        ApplyLocalSearch (optional)
        UpdatePheromones
    end while
end procedure
```

In the next sections we will discuss several variants of ACO mechanic: Ant System (AS) [8], Ant Colony System (ACS) [6] and Max Min Ant System (MMAS) [32]. There

exist other variants of ACO, however we consider that these three approaches are not only relevant to our thesis but the most widely used in recent research.

2.4.1 Ant System

AS is the original ACO technique proposed by Marco Dorigo in his PhD dissertation [7], it was later published in [8]. Consequently AS is the simplest form of ACO.

AS started by initializing the pheromone variables τ_{ij} to the initial value tau_0 (step 1). On step 2, each ant will be responsible to construct a complete candidate solution. At each construction iteration the probability $pr(c_{ij}|\pi)$ to choose the solution componentc_{ij} to be added to partial solution π is defined as follow

$$pr(c_{ij}|\pi) = \frac{[\tau_{ij}]^{\alpha}.[heur(c_{ij})]^{\beta}}{\Sigma_{k \in neigh(\pi)}[\tau_{ik}]^{\alpha}.[heur(c_{ik})]^{\beta}}, j \in neigh(\pi) \tag{2.9}$$

where α and β are respectively the impact power of pheromone and the heuristic value. If $\alpha = 0$ the selection will depend only on the heuristic value so the algorithm will become stochastic greedy. If $\beta = 0$ the selection will only depends on pheromone value. $heur(.)$ is heuristic function to measure the solution component. $neigh(.)$ is neighboring function which will return all feasible solution components for partial solution given. The probability above is only defined on feasible solution component k, any other infeasible solution component will be assigned 0 probability. The final step of AS to update the pheromone consists two steps: pheromone accumulation and pheromone evaporation. Pheromone evaporation is defined as:

$$\tau^{*}_{ij} = (1 - \rho)\tau_{ij} \tag{2.10}$$

where $0 < \rho \leq 1$ is pheromone evaporation rate. Pheromone accumulation is defined as

$$\tau^{*}_{ij} = \tau_{ij} + \Sigma_{s \in S_d | c_{ij} \in s} eval(s) \tag{2.11}$$

where S_d is a set of solutions that are used to accumulating pheromone and $eval(.)$ is *evaluation function* to assess the quality of a solution. Evaluation function is designed to return a higher value for a good solution and lower value for a bad solution. Typical value design for $eval(s)$ is $1/obj(s)$ where $obj(.)$ is objective function (for example, in the case of the Traveling Salesman Problem it is defined as length of the tour). Consequently pheromone value of good solutions components will be accumulated while value of bad solution component will mostly evaporate.

2.4.2 Max-Min Ant System

MMAS is introduced by [32, 33] as an improvement of traditional AS system. The improvement can be outlined as follow:

- Limiting pheromone value

- Different pheromone initialization value
- Strong elitist strategy to update pheromone trails
- Occasional pheromone reinitialization

2.4.2.1 Strong Elitist Strategy

MMAS adapt their pheromone update process to strong elitist strategy where only the best ant can update the pheromone variable. Selected ant can be iteration-best or global-best. Iteration-best of iteration i defined as the ant with the best solution on iteration i. Global-best is defined as the ant with best found solution so far. This modification will affect the S_b of equation 2.11 above replaced to iteration-best S_{ib} or global-best S_{gb}.

2.4.2.2 Pheromone Value Limitation

Strong elitist strategy as described in 2.4.2.1 combined with regular AS can lead to stagnation. In case of stagnation, a dominant trail with high value of pheromone is found. To control the trail strength, explicit limit is introduced as follow.

$$\tau_{min} \leq \tau_{ij} \leq \tau_{max}, \forall \tau_{ij} \tag{2.12}$$

where τ_{min} and τ_{max} is respectively lower bound and upper bound of τ_{ij}. The principles to decide the correct value of τ_{min} and τ_{max} are discussed in [32, 33].

2.4.2.3 Initialization

In the initialization step τ_0 will be assigned to τ_{max}. By using this setting, during elitist process only a good solution can maintain their pheromone value since the evaporation procedure will gradually reduce the pheromone value of bad solutions. In case of stagnation a reinitialization might be used to reset the pheromone value.

2.4.3 Ant Colony System

ACS introduced by [6] is originally aimed to improve AS performance on symmetric and non-symmetric Traveling Salesman Problem (TSP). Three main modification from AS are as follow:

1. Modified state transition rule to balance the exploration of new edges and exploitation of *a priori* knowledge.
2. Global updating rule. Strong elitist mechanic to update global pheromone as described in 2.4.2.1.
3. Local pheromone update while an ant construct a solution.

As the global updating rule is similar to strong elitist mechanism in 2.4.2.1, we will only explain in detail the modified state transition rule and local pheromone update.

2.4.3.1 Modified state transition

The selection of next solution component c_{ij} to be added into partial solution π is specified as follow

$$c_{ij} = \begin{cases} \arg\max_{j \in neigh(\pi)}\{[\tau_{ij}].[heur(c_{ij})]^{\beta}\} & q \leq q_0 \\ c_r & otherwise \end{cases} \tag{2.13}$$

where q is a random uniformly distributed number in $[0,1]$, $0 \leq q_0 \leq 1$ is a parameter, and c_r is a random variable selected stochastically according to equation 2.9. This state transition rule is known as *pseudo-random-proportional-rule*. The parameter q_0 determines the importance between exploration and exploitation. In each solution construction step, a random number $0 \leq q \leq 1$ is sampled. If $q \leq q_0$ then the exploitation is done by choosing the best known solution component according to equation 2.13, otherwise biased exploration is done by choosing new solution component according to equation 2.9.

2.4.3.2 Local Pheromone Update

In order to make visited solution components less appeal local pheromone update is performed as follow

$$\tau_{ij} = (1 - \omega)\tau_{ij} + \omega\delta \tag{2.14}$$

where $0 \leq \omega \leq 1$ and δ are parameters. This update procedure is executed each time an ant visits a solution component.

2.5 Hybrid Ant Colony Optimization and Constraint Programming

The concept of hybridization aims to bring up the advantages of two methods. In this case one want to obtain the time effectiveness of the ACO but still effectively deals with constraints of the problem using CP.

The first research to do hybridization between ACO and CP is [25, 26]. Hybridization is employed to machine scheduling problem with sequence-dependent setup times. They proposed two different types of hybridization namely loose coupling and tight coupling. Loose coupling means that ACO and CP runs in parallel exchanging only important information on bounds and solution candidates. On other hand, tight coupling is done by interleaving CP into the solution construction part of ACO. Inspired by Stochastic Ranking (SR) on Evolution Strategies, they adapt SR to replace traditional penalty based ACO. Together with ACS as the base algorithm, the resulting metaheuristic for loose coupling is Ant Colony System with Stochastic Ranking (ACSsr). Tight hybridization keeps ACS as their metaheuristic because the tight coupling cannot deal with infeasible solution that are possibly included by SR. As for the result, loose coupling slightly improve on the

computation time from pure approach (ACSsr and CP). In constrast, tight coupling outperform the other approaches. Note that tight hybridization is an approximative search not a complete search. In the interest of global optimal solution one need to use loose hybridization.

Another similar tight hybridization can be found at [19], where the authors employ MMAS scheme as the metaheuristic and IBM ILOG solver as the constraint engine. Using a proposed algorithm called Ant-CP, they address the car sequencing problem. On their research, they also make experiments using different pheromone strategies. The consequence of ACO being master in this algorithm is incompleteness of this algorithm. Later on, [20] proposed a complete search hybrid ACO and IBM ILOG solver, using a specific ILOG solver called CP Optimizer (CPO). The algorithm is a two phase algorithm, where both ACO and CP take parts in both phases. In first phase CP sample the space of feasible solution, while ACO apply reinforcement learning resulting in pheromone trail for the second phase. Then in second phase CP does a complete search to find optimal solution using pheromone trail provided by ACO for value ordering heuristic. The resulting CP Optimizer - Ant Colony (CPO-ACO) algorithm employed into three different COP: Multidimensional Knapsack Problem (MKP), Quadratic Assignment Problem (QAP), and Maximum Independent Set (MIS). The experiment shows that CPO-ACO performs better than CPO for most instance on all three problem.

Investigating on incomplete search of hybrid ACO and CP bring up an idea to speed up the computation time. Parallel computing is introduced to the hybrid via Beam ACO [2]. The idea to replace the current ACO with Beam ACO is investigated in [34, 36, 37]. Beam ACO basically employ parallel non-independent construction of ACO solution in Beam search style. All these research address different problem domain and employing hybrid CP and Beam ACO.

- [36] address Single Machine Job Scheduling problem with MMAS metaheuristic.
- [37] address Car Sequencing problem with MMAS metaheuristic.
- [34] address Shared Resource Constrained Scheduling problem with ACS metaheuristic.

Experiments were conducted to compare the effectiveness between three algorithm pure ACO, CP-ACO, and Beam CP-ACO, and show that the addition of Beam search can improve the feasibility of optimal solution given the same time limit. Furthermore [35] addresses the Component Deployment problem with the hybrid CP-ACS approach with LS. Component Deployment problem is believed to be extremely constrained. The result itself shows that pure ACS outperform the CP-ACS approach.

2.6 Public Bus Assignment Problem

Since we are trying to adapt real life public transportation problem, our problem instance is quite unique. A similar problem instance has been addressed by David Semedo in his thesis [31]. However we need to point out that there is some differences between our work and Semedo's work.

Semedo's problem model is heterogeneous fleet. In spite of coming from the same depot, each vehicle has a type. Each type is associated with a different constraints set. Constraints of each type are: maximum working time, maximum time between two maintenance tasks, minimum time of maintenance task, restriction to some lines, and operational cost of vehicle.

In our work we consider homogeneous fleet, so all vehicle is assumed to be identical. However we introduce time windows to investigate better schedule and bus assignments. By using time window we hope that the system can assign the vehicles slightly off the initial departure time. The new departure time hopefully can improve the assignment quality.

As stated by Semedo, PBAP is a variant of MDVSP but with several additional constraints. In this work we are not going to compare our approach with state of the art of MDVSP. The main reason is because the problem instance itself has a lot of differences to typical MDVSP and to the best of our knowledge, no one has addressed the same PBAP as ours. Our main focus is thus to introduce time window to the system and observe its impact for reducing the overall cost.

Chapter 3

Proposed Method

This chapter gives a detailed description of our approach to the PBAP. Firstly we discuss the main process algorithm in section 3.2. Then vehicle behavior is described in section 3.3. Vehicle selection is the core of our PBAP approach and is described in sections 3.4 and 3.5. Finally our proposal to interleaving the process with CP is described in section 3.6.

3.1 Glossary of Notation

Before we discuss our approach to PBAP, we recall some notations from problem definition in section 1.2. Initially the problem consists of sets of lines, terminals, point of departures, and depots. Additionally a set of location indexes and a distance matrix is defined. Location indexes are defined as domain values for depots and terminals. The distance matrix, which uses a time metric, provides information about distance between two locations. These notations are given in table 3.1.

Table 3.1: PBAP Main Notation

$L = \{L_1, L_2, \ldots L_l\}$	set of lines
$T = \{T_1, T_2, \ldots T_{2l}\}$	set of terminals
$P = \{P_1, P_2, \ldots P_{2lk}\}$	set of point of departures
$D = \{D_1, D_2, \ldots D_m\}$	set of depots
$loc_j,\ j \in \{1, 2, \ldots, 2l + m\}$	a location index
$dist[loc_a][loc_b]$	distance between location loc_a and location loc_b where $a, b \in \{1, 2, \ldots, 2l + m\}$

Each point of departure P_k in P lies within a time window with a lower bound and

an upper bound. We also define a trip for each point of departure P_k which has start and end location. These attributes are defined for each point of departure as in table 3.1

Table 3.2: Attributes of P_k

low_k	time window lower bound
up_k	time window upper bound
$start_k$	start location
end_k	end location

We also define a vehicle v_i where $i \in \{1,\ldots,nVehicle\}$ as a bus to be assigned to each point of departure in P. Each vehicle carries some state information as given in table 3.3.

Table 3.3: State Information of v_i

$vLoc_i$	current location
$vLow_i$	time window lower bound in current location
vUp_i	time window upper bound in current location
$vTotalDist_i$	total distance traveled so far
$vBtd_i$	how many times v_i visited the depot
$vLtrf_i$	how many times v_i changed line

3.2 Main Process

The main goal of the PBAP is to find an assignment of buses to each of departure points P_k such that the operational cost is minimized. As mentioned before, we will follow a constructive approach. We implement three strategies to select the bus to be allocated to each departure point, which will be detailed in section 3.4 and 3.5.

The main process of our approach is summarized in Algorithm 2. Initially all points of departure $P_k \in P$ are sorted based on their lower bound time window low_k. Then for each departure point P_s in the sorted set, the bus is selected based on specified strategy $strat$.

Algorithm 2 PBAP Algorithm

```
procedure PBAP Algorithm return Assignment of bus to each departure point
    depSorted ← sort all P_k ∈ P based on low time window low_k
    for all P_s in depSorted do
        listBus ← bus pre-selection
        setting utility for each bus in listBus
        busAllocated ← SelectBus(strat)
    end for
end procedure
```

A pre-selection is done to eliminate infeasible buses before executing the vehicle selection strategy. This process is mainly aimed at vehicles located in the terminals, vehicles in the depot are feasible in general. The following items are considered in the pre-selection process:

- Incompatible bus
- Maximum wait time
- Maximum number of bus
- Single bus in terminal

Bus incompatibility is checked in both terminal and depot locations. A bus v_i is compatible to be used in the point departure P_s if the following criteria is verified:

$$vLow_i + dist[vLoc_i][start_s] \leq up_s \tag{3.1}$$

Criteria above is measured using lower time window of vehicle $vLow_i$ and upper time window of point of departure up_s. This is designed to allow some bus which possibly arrives at the last minute before the upper time limit of the point of departure. Special case is considered for a bus in a depot which has never been used before, that we consider to be always compatible.

For each bus we also check whether it already stayed too long in the terminal. To calculate the waiting time of bus v_i in departure P_s we should consider previous task done by v_i. Let r be the index of previous point of departure served by bus v_i. The waiting time of bus v_i can be measured in two ways. Optimistic measure of waiting time is formalize as follow

$$oWait = low_s - (up_r + \delta) \tag{3.2}$$

Pessimistic measure of waiting time is formalize as follow

$$pWait = up_s - (low_r + \delta) \tag{3.3}$$

Both measurements use the distance δ between start location of previous task and end location of previous task plus the distance between end location of previous task and start location of current task.

$$\delta = dist[start_r][end_r] + dist[end_r][start_k] \tag{3.4}$$

Optimistic wait time measures the lowest waiting time possible. This is done by assuming the bus departed in the last minute at previous point of departure P_r and scheduled to depart in the earliest possible minute at current point of departure P_s. In contrast, pessimistic wait time measures the highest possibility of waiting time. This is done as opposite of optimistic way, considering early departure at previous point of departure P_r and last minute departure at current point of departure P_s.

In this thesis, we fix optimistic measure as a default to estimate waiting time. A constant $maxWaitTime$ is specified to denote maximum waiting time allowed. If a bus locating at terminal stays more than $maxWaitTime$ the system will send it back to depot.

To limit the number of buses in each terminal we implement a stack data structure with *first in first out* rule. A stack, denoted by $stack_i$, is allocated for each location index loc_i. In the beginning of the pre-selection process, each stack of terminal location index is inspected. If the stack is not empty and the total item in the stack exceed $maxStackSize$, each item will be removed until number of item in it is $maxStackSize$. Removed buses will be sent back to depot. Initially all bus items are added to the stack of associated depot location. Each time a bus is moved to a certain location, a bus item is removed from the stack of current location to the stack of the destination location.

When a single bus is alone in a terminal, it is prevented from serving a point of departure in other line. For a point of departure P_s in loc_j, a vehicle v_t located in loc_i cannot be assigned if loc_j and loc_i are on different lines and v_t is the only bus in loc_i (i.e. the size of $stack_i$ is 1).

3.3 Vehicle Behavior

Before we discuss the possible strategies to select vehicle we will explain certain behaviors that may occur throughout the process. These behaviors will directly or indirectly affect the main process.

Vehicle and depot. A vehicle in the system can always go back to depot after a trip. However by doing so, the operational cost will increase. A vehicle will nevertheless be enforced to go back to depot under certain circumstances:

- A vehicle stayed too long in a terminal, exceeding $maxWaitTime$.
- Too many vehicles in one terminal, $maxStackSize$ is violated.
- At the end of the process, all buses need to go back to depot.

When a bus goes back to its depot the distance between its current location and depot location is considered as distance it travels. Let v_i be a vehicle going back to its depot $vDepot_i$.

$$vTotalDist_i = vTotalDist'_i + dist[vLoc_i][vDepot_i]; \tag{3.5}$$

The system will also increment $vBtd_i$ counter which denote how many times v_i went back to depot.

$$vBtd_i = vBtd'_i + 1 \tag{3.6}$$

Prime ($'$) symbol denotes the old value of corresponding variable.

Vehicle do trip. Whenever a vehicle v_i is selected to be assigned to serve point of departure P_j it will serve a trip trp_j that starts at location index $start_j$ and ends at location

index end_j. First of all it needs to travel from its current location $vLoc_i$ to $start_j$ then it needs to travel from $start_j$ to end_j. Total distance traveled is updated as follows

$$vTotalDist_i = vTotalDist'_i + dist[vLoc_i][start_j] + dist[start_j][end_j]; \tag{3.7}$$

A special case occurs when the location of vehicle $vLoc_i$ and the start of trip $start_j$ is not associated to the same line. We call this a line transfer and as a consequence, the line transfer counter $vLtrf_i$ will be incremented

$$vLtrf_i = vLtrf'_i + 1 \tag{3.8}$$

3.4 Vehicle Selection

Vehicle selection is the core of PBAP algorithm. The selection mainly depends on the utility value which is assigned to each vehicle on each iteration, as defined in algorithm 2. For point of departure P_s the utility value of vehicle v_i is defined taking into account two aspects:

1. Distance between location of vehicle $vLoc_i$ and point of departure $start_s$.
2. Possible wait time to serve point of departure.

Using the two aspects above an utility function is formalized so that it grows inversely and exponentially with them. This means the greater the distance and the wait time the smaller utility value will be assigned. We formalize it as follow

$$util(v_i, P_s) = \mathrm{e}^{-\left(\frac{(\Delta + wait(v_i, P_s) * w_1)}{w_2} + new(v_i) * w_3\right)} \tag{3.9}$$

where wait time for vehicle located in terminal is defined as follows

$$wait(v_i, P_s) = \begin{cases} low_s - (\Delta + vLow_i) & \text{if } \Delta + vLow_i < low_s \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Note that wait time is not relevant when a vehicle v_i is located in its depot.

Δ denotes the distance between vehicle's location $vLoc_i$ and start location of point of departure $start_s$, i.e.

$$\Delta = dist[vLoc_i][start_s] \tag{3.11}$$

To calculate the waiting time at equation 3.10 we assume that the vehicle departs as early as possible from their location and also from the current departure point.

$new(v)$ is a term denoting whether vehicle v was ever used before. We formulate it as follow

$$new(v) = \begin{cases} 1 & \text{if } v \text{ was never been used before} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

Three constants w_1, w_2 and w_3 are specified as a weight for each term. w_1 is specified within a range $\{0 \dots 1\}$ to reduce the influence of waiting time (with the distance), so that the function will give more value to a vehicle with a closer location. Additionally w_2 is a denominator for both distance and waiting time, and is specified such that it avoids producing too small utility values, as we intend that utility values are spread between range $\{0 \dots 1\}$ evenly. If the total distance and wait time is too big the utility value will be closer to 0. We also specify weight w_3 for the term $new(v_i)$ to give less priority for an unused bus from depot. This is convenient to give higher preference to a bus that is in use so the total number of used buses tends to be minimized.

As we mentioned above the growth of the utility function is inverse to the distance and wait time. If the Δ and $wait(v_i, P_s)$ equal to 0 the function will give value 1, since $e^0 = 1$. As the distance and waiting time grow larger, the utility value will grow exponentially smaller. The exponential growth of this function is illustrated in figure 3.1.

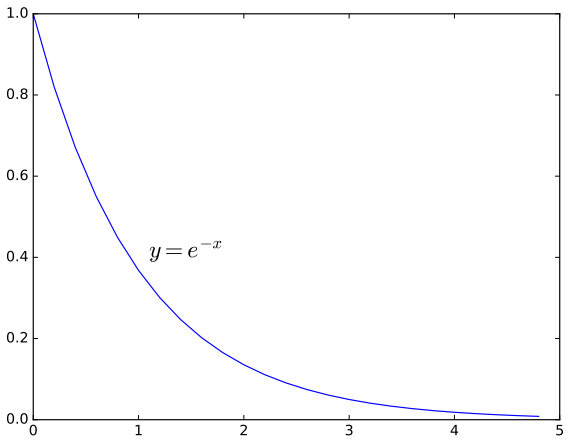


Figure 3.1: $y = e^{-x}$

It is worth mentioning that we did use other form of utility function before. However the growth of the function was too ragged thus the results are bad.

In this thesis, we employ three different vehicle selection strategies to solve PBAP

- Greeedy selection
- Probabilistic based utility
- Ant Colony Optimization Approach

3.4.1 Greedy Vehicle Selection

A greedy vehicle selection is the simplest heuristic to select the vehicle. Given a list of bus $listBus$ from pre-selection process, greedy vehicle selection for point of departure P_s

is defined as follow

$$v = select_{greedy}(P_s) = \arg \max_{v_i \in listBus} util(v_i, P_s) \tag{3.13}$$

As defined above, the greedy vehicle selection heuristic will choose the vehicle which has the greatest utility value. This selection heuristic is convenient for providing some bound on the operational cost against which the performance of the other approaches can be measured.

3.4.2 Probabilistic Based Utility Vehicle Selection

A PBU heuristic is defined over the list of $nBusPBU$ buses. The idea is to give some possibility to choose a non-best vehicle in the list. By this hypothesis we expect that the search may escape local optima leading to a better solution. The process is summarized as follow:

1. Sort $listBus$, from pre-selection process, based on utility value.
2. Filter duplicate unused items from same depot.
3. Take the best $nBusPBU$ items to a new list $nListBus$.
4. Select the bus from $nListBus$ with probability $prob(v_i, P_s)$.

Before selecting the best $nBusPBU$ items we remove duplicate buses from same depot which have never been used before. The removal is necessary to assure we are selecting a bus from a list of items with unique distance and waiting time. Two unused vehicles from the same depot will have same distance and waiting time.

Probability $prob(v_i, P_s)$ for vehicle v_i to be selected for departure point P_s is formalized as follows

$$prob(v_i, P_s) = \frac{util(v_i, P_s)}{\sum_{v_i \in nListBus} util(v_i, P_s)} \tag{3.14}$$

We limit the selection to a small number of $nBusPBU$ in order to avoid selecting bad vehicle most of the time. For example assume we have a good vehicle with utility value 1 and there are 100 other vehicles with utility value 0.04. The probability assigned for good vehicle is 20%. In contrast the chance for one of vehicles with utility 0.04 to be selected is 80%. If we limit the $nBusPBU$ to a small value we can favor good vehicles in most cases while maintaining a small chance to choose bad vehicle.

Finally we apply the PBU selection strategy by doing the solution construction $maxIter$ times and take the best solution as the output. The whole PBU approach is summarized in algorithm 3.

Algorithm 3 PBAP PBU

```
procedure PBAP PBU return best solution
    initialization
    for iter = 0 to maxIter do
        sol ← construct solution
        if sol is better than best solution then
            update best solution
        end if
    end for
end procedure
```

3.5 Ant Colony Optimization Approach

So far we mention vehicle selective approaches who are focusing on local point of departure at a time. Even with slight probability to choose not-best vehicle we still made decisions locally, not taking advantage of previous runs. To do so, we adopted an ACO metaheuristic approach, by exploiting the reinforcement learning feature provided by the ACO. In this approach we modified the probabilistic based utility approach adding a value of pheromone to the calculation. Pheromone matrix is defined to memorize a good path that have been explored in previous runs. In this thesis we adapt the approach of MMAS, see section 2.4.2.

3.5.1 Base Ant Algorithm

The main idea of the ACO approach is to assign each ant a task to run the PBAP algorithm. Using strong elicit strategy, the local best ant on each iteration will update the pheromone matrix, as outlined in algorithm 4.

Algorithm 4 PBAP Ant Algorithm

```
procedure PBAP Ant Algorithm return global best solution
    initialization
    for iter = 0 to maxIter do
        for ant = 0 to nAnt do
            construct solution
            update iteration best solution
        end for
        update pheromone matrix using iteration best solution
        degrade and limit pheromone matrix
        update global best solution
    end for
end procedure
```

In algorithm 4, *maxIter* and *nAnt* denote the maximum number of iterations and ants, respectively. In the initialization phase we assign value 1.0 to pheromone matrix. The solution construction phase will follow the same constructive approach specified in

algorithm 2 with a modified probability function specified as follow.

$$prob_{ant}(v_l, P_s) = \frac{util(v_l, P_s)^{\alpha} \lambda^{\beta}_{start_s vLoc_l}}{\sum_{v_i \in nListBus} util(v_i, P_s)^{\alpha} \lambda^{\beta}_{start_s vLoc_i}} \tag{3.15}$$

where α and β denote the power of utility and pheromone, respectively. In each iteration the iteration best solution is updated based on the assessment of objective function. The iteration best solution will then be used to update pheromone matrix. The pheromone matrix update is formalized as follow

$$\lambda'_{ij} \leftarrow \lambda_{ij} + \frac{1}{ibCost} \tag{3.16}$$

where $ibCost$ denotes the value of the objective function of the best solution in the iteration.

Next, the pheromone matrix will be degraded by a factor of ρ. The degradation of the λ values is formalized as follow

$$\lambda'_{ij} \leftarrow \lambda_{ij} * \rho \tag{3.17}$$

We use following formulation to limit pheromone value

$$\lambda_{min} \leq \lambda_{ij} \leq \lambda_{max} \tag{3.18}$$

All pheromone value will be checked at each update, if it violates the limit, it will be set to the limit. Finally the process will return the global best solution as output.

3.5.2 Pheromone Matrix

In this thesis initially we tried to implement a matrix similar as that used in [31], whose size is given by the size of location times the size of point of departure $|P| \times |L|$. Each element of matrix λ_{ij} represents the desirability to assign a vehicle from location L_i to the point of departure P_j. However in our case this matrix gave bad results since it becomes really rather large for some big instance with numerous point of departure and thus overfits the underlying reinforcement learning implemented by the ACO.

We thus adopted a new pheromone matrix with the size of location times the size of location $|L| \times |L|$. Each element of matrix λ_{ij} corresponds to the assignment of a bus in location L_j to a point of departure in location L_i. The matrix is square and illustrated as follow

$$\begin{pmatrix} \lambda_{1,1} & \lambda_{1,2} & \cdots & \lambda_{1,|L|} \\ \lambda_{2,1} & \lambda_{2,2} & \cdots & \lambda_{2,|L|} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{|L|,1} & \lambda_{|L|,2} & \cdots & \lambda_{|L|,|L|} \end{pmatrix} \tag{3.19}$$

By using this matrix we model the pheromone as the desirability of assigning vehicle located in L_j to the point of departure in location L_i.

3.6 Interleaving with CP

Tight coupling hybridization of ACO and CP is originally aimed to narrow the neighborhood [25, 26]. ACO as the master algorithm and CP is interleaved to prune the search space. In our case, we not only use this approach to narrow the search space but we also use CP to ensure bounds consistency [14]. We are motivated to do so since we are dealing with time windows. Without enforcing bounds consistency we noticed that it was almost impossible to obtain valid timetables from the assignments produced by our algorithms. The problem is that in each assignment step we make decisions based on local bounds. These decisions produce several constraints that, when propagated to the whole problem, might violate other bounds and cause a contradiction which lead to bound inconsistency. Thus the interleaving propagation is really important in each assignment step to ensure bounds consistency of the constraints. Interleaving bounds consistency in our problem we can ensure that any bus assignment leads to a valid timetable solution. We specified the theorem and proof that bounds consistency implies satisfiability in appendix A.

We thus modify our base PBAP approach to interleave constraint propagation in algorithm 5. Solver is initialized in the initialization step. After each vehicle assignment

Algorithm 5 PBAP-CP Algorithm

```
procedure PBAP-CP return Assignment of bus to each departure point
    depSorted ← sort P based on low time window low_i
    Solver initialization
    for all P_s in depSorted do
        listBus ← bus pre-selection
        setting utility for each bus in listBus
        while listBus is not empty do
            busAllocated ← SelectBus(strat,listBus)
            constraint propagation using busAllocated
            if contradiction found then
                unpost constraint
            else
                assign busAllocated for P_s
                break out of while loop
            end if
        end while
    end for
end procedure
```

step, a constraint regarding time window compatibility is posted. It is meant to test whether a contradiction occurs. If a contradiction is found, there is no timetable solution for corresponding assignment, a backtrack is done by un-posting the constraint and repeat the process with another vehicle. If no contradiction is found, the assignment is made and the process continues to another point of departure. A special case happens when no more bus available from the *listBus* in which case we consider it does not have timetable solution and therefore that iteration is discarded.

We model the constraint problem using a variable $P_i, i \in \{1,\ldots,|P|\}$ for each point of departure. Initially the domain for each variable P_i is defined as the corresponding time window $dom(P_i) = \{low_i,\ldots,up_i\}$. The constraint is posted only for the buses that have been used before. If the process selects a bus that has never been used before, no constraint will be posted and constraint propagation will be irrelevant. To define a constraint which will be propagated, when we select vehicle v_l to be assigned to point of departure P_s, we consider previous task served by v_l. Let m be the index of the previous task served by vehicle v_l. Note that if v_l located at depot but has been used before, we consider a task which is served by v_l before it went back to depot. A constraint that needs to be propagated is defined as follows

$$\delta \leq P_s - P_m \tag{3.20}$$

where δ, when $vLoc_l$ is a depot location index, is defined as follow

$$\delta = dist[start_m][end_m] + dist[end_m][vLoc_l] + dist[vLoc_l][start_s] \tag{3.21}$$

otherwise it will be defined as follow

$$\delta = dist[start_m][end_m] + dist[end_m][start_s] \tag{3.22}$$

The constraint specified in 3.20 imposes that the scheduled departure time from P_s should be at the time of departure $P_m + \delta$ or later. It also denotes that the scheduled departure time of P_m should be at the time departure $P_s - \delta$ or earlier. When this constraint is propagated, it might possibly narrow the bound of both P_s and P_m.

It is worth mentioning that the topology of our constraint network is a sequence of difference constraints, each constraints being a binary constraint imposed over two consecutive variables.

3.7 Finding solution near the ideal departure time

After acquiring the assignment of buses to each point of departure and narrowing the domain of each point of departure, we are interested to search a solution in which the difference between the scheduled departure time and the ideal departure time are minimized. Ideal departure time $ideal_j$ is the initial time for each point of departure P_j. More detail about ideal departure time is presented in section 4.1.2.

The implementation is as follows. Initially we use variable P_j to model each point of departure. Bounds for P_j are defined by using lower time window bound low_j and upper time window bound up_j.

$$low_j \leq P_j \leq up_j$$

Next, we define additional variables $d[j]$, where $j \in \{1,\ldots,|P|\}$, for each variable of point of departures P_j and impose following constraint

$$d[j] = |P_j - ideal_j|$$

We also define an objective function $\mathcal{J}$ as a sum of difference variables $d[j]$.

$$\mathcal{J} = \sum_{j} d[j]$$

Finally we find a solution that minimize the objective function $\mathcal{J}$. To search for the optimal solution we use *mindom_LB* heuristic from choco-solver. It is a search strategy using non-instantiated smallest domain variable selection and lower bound value selection heuristic. We limit the optimization search by using a fail limit of 1000000.

Chapter 4

Experimental Result

In this chapter we present and discuss experimental results obtained with the benchmarks that were constructed, and the tools that we used. Firstly, we address benchmark construction in section 4.1. Next, tools and technologies we used are discussed in section 4.2. Then the experiment result to determine parameters for utility function are presented in section 4.3. We discuss the experiment to determine ACO parameters in section 4.4. Finally we assess the different strategies used across various problem instances in section 4.5.

4.1 Benchmark construction

As mentioned before our definition of PBAP is quite unique. Since we could not find benchmarks that meet our requirements, we constructed our own benchmarks considering several key features: star shaped topology, multiple depot, and line exchange. In general we generate a benchmark as outlined in algorithm 6.

In each initiation step, each departure point P_s is associated to a terminal and a line thus will give the starting position $start_s$ and end position end_s. The distribution of each point of departure will be detailed in section 4.1.1. Next, an ideal departure time for each point of departure is generated. After the ideal departure time is fixed the process generates time windows according to the slack given. The process on generating time window will be presented in more detail in section 4.1.3.

The construction of distance matrix $dist$ is done according to a star topology network, as discussed in section 4.1.4. Finally each vehicle will be randomly assigned to a depot.

We construct ideal departure time as initial schedule by randomly selecting a value in certain intervals. This initial schedule are made similar to real world instance. The process of generating ideal departure time will be detailed in section 4.1.2.

Algorithm 6 PBAP Benchmark Construction Algorithm

procedure CONSTRUCT PBAP BENCHMARK(nLine,nDepot) **return** An instance of benchmark
  for all P_s in P **do**
    Initiate P_s
  end for
  Generate ideal departure time for all point of departure
  $generateTimeWindow(slack)$
  $dist \leftarrow generateDistMat()$
  for $i = 1$ to $nVehicle$ **do**
    Randomly assign v_i to any $D_j \in D$.
  end for
end procedure

4.1.1 Distribution of Point of Departure

As defined in section 1.2, each problem instance has set of lines $L = \{L_1, L_2, \ldots, L_l\}$ and set of terminals $T = \{T_1, T_2, \ldots, T_{2l}\}$. The distribution of terminals to each line is defined by associating T_{2i-1} and T_{2i} to line L_i. We can define the distribution of points of departure $P = \{P_1, P_2, \ldots, P_n\}$ as illustrated in figure 4.1.

Assume that we have 2 line $L1, L2$, the distribution of point of departures $P_1, P_2, \ldots P_n$, where $n = 2kl$, are as follow:

- Each line has $2k$ point of departures. $P_1, P_2, \ldots P_{2k}$ belong to L_1, $P_{2k+1}, P_{2k+2}, \ldots, P_{4k}$ are belong to L_2.
- Initial point of departure P_1 is located in terminal T_1 on Line L_1
- Consecutive point of departure is located in the opposite terminal within same line. For example P_2 as the consecutive point of departure from P_1 is located in the opposite terminal T_2, both T_1 and T_2 are belong to L_1.
- The ideal departure time of all point of departure within same terminal is in increasing order. Thus $ideal_i \leq ideal_j$ for all $i < j$ located in same terminal.
- The consecutive point of departure from the last point of departure in each line is located in the first terminal on the next line. For example consecutive point of departure from P_{2k} namely P_{2k+1} is located in first terminal T_3 of next line L_2.

4.1.2 Ideal Time

The initial departure time for each point of departure before we do expand it into time window is considered the ideal time of that point of departure. For each terminal we generate ideal times of departures according to a combination of normal, rush and night/-morning phases, arranged as in figure 4.2. We arrange the phase so that it mimics real life

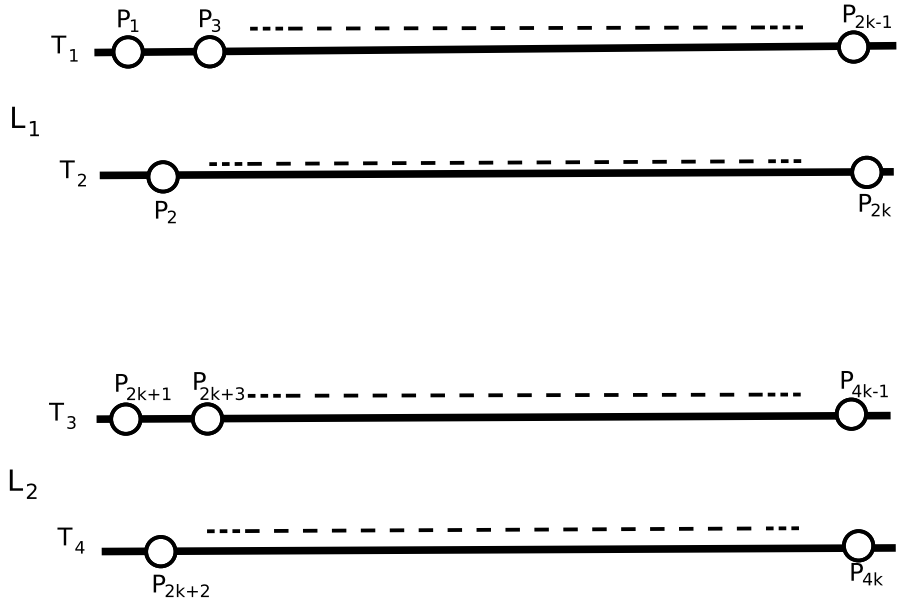


Figure 4.1: Distribution of Point of Departures

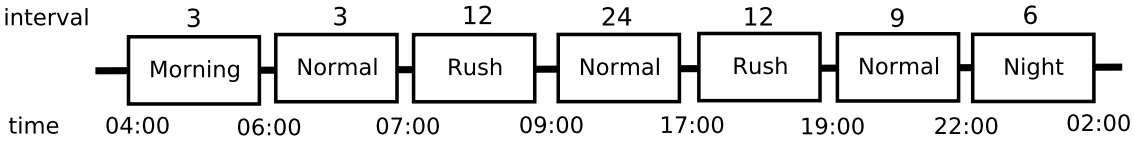


Figure 4.2: PBAP Phase Organization

situations. The number of intervals between departures on each phase is calculated by taking the difference of time limit and divide it by base interval on corresponding phase.

While generating ideal time we consider several constants as follow

- k denotes the number of point of departure on each terminal.
- $nNormal$ denotes an array of number of intervals in each normal phase.
- $nRush$ denotes an array of number of intervals in each rush phase.
- $nNight$ denotes array of number of intervals in each night and morning phase.
- $startTime$ denotes the beginning of time. We use 240 which denote 4 : 00 in the morning.

- $endTime$ denotes the end of time. We use 1560 which denotes 2 : 00 in the morning next day.
- $intvNorm$ denotes base interval for normal phase.
- $intvRush$ denotes base interval for rush phase.
- $intvNight$ denotes base interval for night and morning phase.

All constants above are adjustable according to user preference. In this thesis we use benchmarks with uniform constant setting. However the ideal time on each line is generated with a deviation dev.

Uniform constant values being used in our work are as follow

- $k = 70$
- $nNormal = [3, 24, 9]$
- $nRush = [12, 12]$
- $nNight = [3, 6]$
- $intvNorm = 20$
- $intvRush = 10$
- $intvNight = 40$
- $dev = 1$

Formally ideal time for each point of departures are generated as outlined in algorithm 7. $random(intv, dev)$ is a function to generate random value with $intv$ as the base and

Algorithm 7 PBAP Ideal Time of Departure

```
procedure GENERATE IDEAL DEPARTURE TIME
    for all T_j in T do
        for all P_s located at T_j do
            P_r is previous point of departure located at T_j.
            if P_s is first point of departure located at T_j then
                ideal_s ← random(startTime, dev)
            else if P_s is in normal phase then
                ideal_s ← ideal_r + random(intvNorm, dev)
            else if P_s is in rush phase then
                ideal_s ← ideal_r + random(intvRush, dev)
            else
                ideal_s ← ideal_r + random(intvNight, dev)
            end if
        end for
    end for
end procedure
```

dev as deviant. Let val be a value produced by this function, following criteria should be satisfied

$$intv - dev \leq val \leq intv + dev \tag{4.1}$$

4.1.3 Time window construction

Given an ideal time for each departure point we can generate time windows as specified in algorithm 8. sl_p denotes the slack percentage, the longer value of which produces wider

Algorithm 8 PBAP Time Window

procedure GENERATE TIME WINDOW(sl_p)
  for all T_j in T **do**
    for all P_s located in T_j **do**
      define P_r as previous point of departure from P_s which is also located in T_j
      if P_s is first point of departure in T_j **then**
        $sl_{lo} \leftarrow (ideal_s - startTime) * sl_p/100$
      else
        $sl_{lo} \leftarrow (ideal_s - ideal_r) * sl_p/100$
      end if
      $low_s \leftarrow ideal_s - sl_{lo}$
      define P_t as next point of departure from P_s which is also located in T_j
      if P_s is last point of departure in T_j **then**
        $sl_{up} \leftarrow intvNight * sl_p/100$
      else
        $sl_{up} \leftarrow (ideal_t - ideal_s) * sl_p/100$
      end if
      $up_s \leftarrow ideal_s + sl_{up}$
    end for
  end for
end procedure

time windows. In our experiment we use several values of sl_p: 0, 10, 20, 30, 40.

4.1.4 Star Network Topology

We take a star network topology to simulate real world instances. The topology is illustrated in a Cartesian coordinate system as shown in figure 4.3 where deg denotes angle difference (in degrees) from one location index to another. m and l respectively denote number of depot and line in the system. Thus $2l + m$ denotes total number of location indices. We calculate deg as follows

$$deg = \frac{360^o}{2l + m} \tag{4.2}$$

As illustrated in figure 4.3 we create the network with a uniform angle distance deg between two consecutive location index. We select a random length of hypotenuse len_i for each location index loc_i.

$$len_i = random(lenMin, lenMax) \tag{4.3}$$

where $lenMin$ and $lenMax$ respectively denote minimum and maximum possible hypotenuse length. Function $random(min, max)$ give a random value $rVal$ in the interval $min \leq rVal \leq max$.

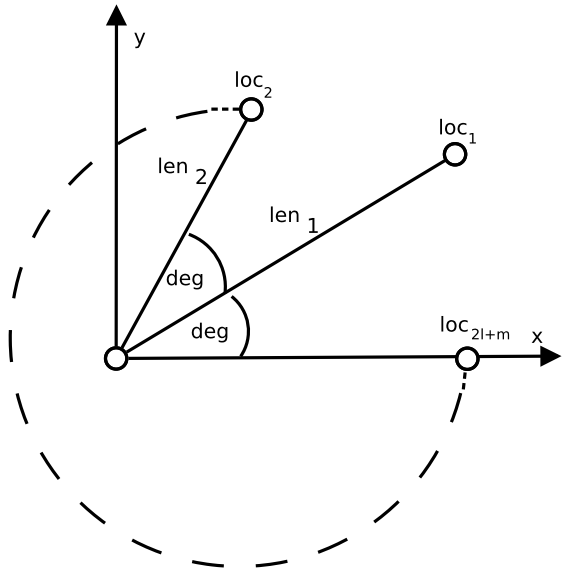


Figure 4.3: Star network topology illustration

The x and y coordinates for each initial location index can then be calculated as follow.

$$locX_i = len_i * \cos(i * deg) \tag{4.4}$$

$$locY_i = len_i * \sin(i * deg) \tag{4.5}$$

Before calculating the distance, we perturb the coordinate of each location index as follow

$$locX_i \leftarrow rand(locX'_i, tol) \tag{4.6}$$

$$locY_i \leftarrow rand(locY'_i, tol) \tag{4.7}$$

where $locX'_i$ and $locY_i^prime$ respectively denote initial x and y coordinate of location loc_i. Function $rand(val, tol)$ generate a random value $rVal$ within interval $val - tol \leq rVal \leq val + tol$

Finally we calculate the distance between two location index loc_i and loc_j using euclidean distance.

$$dist[i][j] = euclid(loc_i, loc_j) \tag{4.8}$$

$$= \sqrt{(locX_i - locX_j)^2 + (locY_i - locY_j)^2} \tag{4.9}$$

A special case is occurred when both loc_i and loc_j are associated to depot location. In this case $dist[i][j]$ is not relevant (a bus does not travel between depots).

4.2 Tools and Technology

We implement our approach using the Java programming language. Java is a versatile programming language that can be used for general purpose. Java is designed as an object

oriented and class based language. Thus any component of the application can easily be reused to another purpose. It is also intended to let developers *write once*, *run anywhere*. Once the code compiled, it can be executed in any platform that supports Java.

We do the coding using IntelliJ IDEA [17]. IntelliJ IDEA is a Java Integrated Development Environment (IDE) that eases the development of Java application. This IDE provides coding assistance such as code completion, navigation, refactoring, and quick fix for errors. It also eases the integration of 3rd party library and framework such as maven, SQL, PostgreSQL, etc.

To manage and propagate constraints we use the choco-solver [28] library. Choco is a Java library for Constraint Programming. This solver has been awarded many medals from MiniZinc competition in 2013 and 2014. MiniZinc is a world-wide competition for CP solver. This solver provides several features such as: various variable types, global constraints and various search strategies. The key feature that we utilize in this work is unpost constraint. In choco solver, it is possible to remove constraints that have been posted before. We utilize this feature to incrementally propagate constraints while dynamically remove them when contradiction occurs.

Other tools that we used to create report are Dia Diagram Editor [4] and Matplotlib [16]. **Dia** is free software to draw structured diagram. We utilize this software to make illustration of many components of our work. Main advantages of using this software is we can easily draw the diagram and export it as vector graphic format (.svg). Thus, the resulting diagram can be rescaled without losing any detail. **Matplotlib** is a python library to draw 2D plot. We utilize matplotlib to draw plot and chart for data illustration. This tool is widely used by scientists to make 2D plot and chart for their publications.

4.3 Utility Function Parametrization

Our utility function, as presented in section 3.4, is controlled by the weights w_1, w_2 and w_3. Each weight is customizable by the user to get a better result. In our experiment we tried several value combinations of w_1 and w_2. To give an uniform penalty for a new vehicle, we fixed value $w_3 = 1$ based on [31]. The experiment results are summarized in table 4.1. We use benchmark with 5 line and $sl = 30$ for the experiments. We set the number of iteration to 1000.

Improvement in table 4.1 is calculated using following formulation

$$improvement(\%) = \frac{gCost - cost}{gCost} * 100 \tag{4.10}$$

where $gCost$ and $cost$ denote cost by greedy strategy and global best cost of corresponding (PBU) strategy respectively.

In the experiment, we use 3 different values for w_2: 1, 5 and 10. Increasing w_2, the denominator of the waiting time and the distance terms on equation 3.9, the resulting effect of waiting time and distance terms is reduced and the output of exponential function can spread around the range $0 \ldots 1$. However, as we can see the result on table 4.1, an

Table 4.1: Result of experiment on w_1 and w_2. We assess the parametrization using PBU strategy. Cost of greedy strategy is provided as a baseline. Min, mean and std respectively denote global best cost, mean value of all solution's cost, and standard deviation of all solution's cost. Improvement is provided to measure how far PBU performance improved from greedy.

Weight		Greedy	PBU			Improvement (%)
w1	w2		Min	Mean	Std	
	1	111080	**110620**	**112754.19**	746.42	1.5
0	5	111080	114240	120044.29	1822.41	-8
	10	111080	137900	153334.37	4406.64	-24
	1	115390	**113880**	**114845.72**	297.64	1.3
0.5	5	115340	121040	128032.84	2592.26	-4.94
	10	115340	158570	173722.83	5341.25	-37.48
	1	130930	**125890**	**127743.18**	548.26	**3.84**
1	5	130020	134140	141894.13	3236.63	-3.16
	10	129840	172750	191979.03	5625.59	-33.04

anomaly happens as the value of w_2 increases. It causes the performance of PBU strategy to go down and standard deviation value go up. We suspect that the value of numerator is not big enough thus by assigning w_2 with a value more than 1 it will make the probability value of each item, to be selected, closer to each other hence making the strategy more similar to a complete random selection. This phenomenon can be confirmed in table 4.1, the standard deviation value goes up as the value of w_2 goes up. In fact we avoided too small utility value by using our CP interleaving process. Since we embed CP propagator on each step of bus assignment, if contradiction occurs the process is forced to backtrack on current assignment and select another item from the list of candidate. When backtrack occurs, the failed item is removed from the list and next selection will not consider it on probability function. This local backtrack mechanism will allow us to assign a really small utility value since, if contradiction occurs, it will repeat the selection process without considering item with bigger probability (fail item). Thus we can safely assign $w_2 = 1$ without losing the chance to select item with really small probability.

We did the experiment using values 0, 0.5 and 1 for w_1 since the purpose of w_1 is to limit the effect of waiting time to utility function. Value 0 will nullify its effect so that the output of utility function will fully depend on distance and new vehicle term. Value 0.5 will half the effect of waiting time and value 1 will give no reduction at all. As we can see in the table 4.1 the best cost is obtained by assign 0 to w_1. The best improvement is achieved when we assign value 1 to w_1. Since the improvement value give us the idea how far utility function based strategy can improve from greedy strategy, it is questionable

why assigning $w_1 \neq 0$ give bad results. We suspect that the cost function indirectly affects the poor results of assigning $w_1 \neq 0$. As shown in equation 1.1 our formalization of cost function does not consider waiting time term. If it is taken into account, we think that it would fix the anomaly and thus assigning $w_1 \neq 0$ yields better operational cost than $w_1 = 0$. Based on reasons above we decide to keep value $w_1 = 1$ to maximize the improvement rate.

To sum up, based on our experiment, we decide to use the following assignments for our experiment: $w_1 = 1$ and $w_2 = 1$. In addition we already fixed assignment $w_3 = 1$. These parameters, unless stated otherwise, will be set as default in the other experiments of this chapter.

4.4 ACO Parametrization

In order to implement a good approach, we need to consider the best ACO parameter for our specific problem. The core ACO approach has several parameters as follow:

- Number of iteration, $maxIter$
- Number of ant, $nAnt$
- α as power of utility value
- β as power of pheromone value
- ρ as evaporation rate of pheromone matrix
- λ_{min} and λ_{max} as the limit of pheromone value

For simplicity, we use the default value as stated in table 4.2. The experiment is done using default benchmark with 5 line and $sl = 30$.

Table 4.2: Default ACO parameter

Parameter	value
$maxIter$	200
$nAnt$	20
α	0.5
β	2
ρ	0.9
λ_{min}	0
λ_{max}	1

In some experiments (see appendix B) we compute linear regression of minimal operational cost over each iteration. It produces a trend line which can be used to observe

the learning direction of the ACO strategy. The slope of the line provides a measurement on how well the ants learn. A negative slope value $slope < 0$ denotes that the ants learn to produce better solution in each iteration. A positive slope value $slope > 0$ denotes that the ant learn to produce worse solution. Finally a slope value near 0 denotes that the ants do not learn at all.

4.4.1 Number of iteration

To determine an adequate number of iterations we started our experiments with $maxIter = 500$. The result is as shown in figure 4.4, which plots minimal operational cost acquired on each iteration. As a result it turns out that our PBAP ACO approach is able to quickly learn and converge the operational cost within 100 iterations. To be safe we fixed the rest of our experiments to 200 iteration.

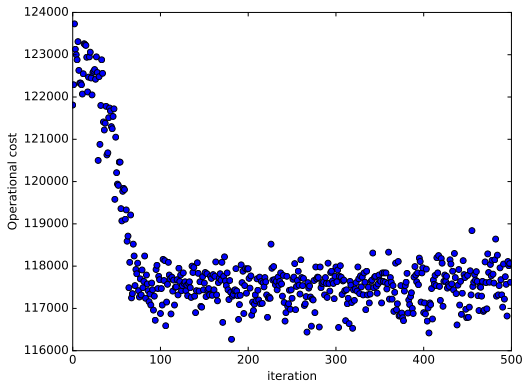


Figure 4.4: Scatter plot of operational cost over iteration using ACO strategy with 500 iteration.

4.4.2 Power to Pheromone and Utility value

The main parameters of ACO probability function, as presented in equation 3.15, are α and β. Both of them decide how the ants make decisions on each of their steps. Since the domain of both α and β is within range $[0 \ldots 1]$, we specify following values: 0, 0.5, 1, 2 to be tested in our experiments. We exclude the experiment using $\alpha = 0$ since it will give a similar result to randomized approach without utility value. The result of experiment is summarized in table 4.3. We also draw a scatter plot for each assignment made, these figures can be found on appendix B.

Assigning 0 to β will nullify the power of pheromone value. The result will be similar as to using the PBU strategy. It is shown in figure B.2, B.3, and B.4 that the approach did no learning at all, slope is near 0. As a result the operational cost of using $\beta = 0$ is the biggest among other assignments $\beta \neq 0$. However we keep this result as a baseline to measure the quality of other assignment.

Table 4.3: Result of experiment on α and β for ACO strategy. We set the number of iteration to 200. *min*, *mean* and *std* denote global best operational cost, mean value of all iterations and standard deviation value of all iteration respectively. *slope* denote the linear progression's trend line slope of operational cost over all iteration.

α	β	*min*	*mean*	*std*	*slope*
0.5	0	120060	122519.9	657.27	0.445
	0.5	117650	120869.45	1168.38	-15.059
	1	117410	119871.1	1619.98	-21.947
	2	**115860**	118759.25	2095.98	**-29.581**
1	0	118480	120186.7	510.15	-1.138
	0.5	117820	119580.9	645.62	-5.376
	1	117560	119000.95	774.55	-9.639
	2	116670	**118352.3**	1057.87	-15.817
2	0	118410	119232.15	272.60	0.346
	0.5	118410	119076.4	278	-0.765
	1	117830	118929.7	312.45	-2.096
	2	117250	118594.15	476.25	-5.904

If we assign 0.5 to either α and β it will give more power to corresponding term, since both terms are in range $[0 \dots 1]$. In contrast if we assign 2 the resulting term will have smaller value thus giving less power to corresponding term. Finally by assigning 1 we do not change the power of corresponding term.

Based on the result shown in table 4.3, the smallest *mean* is acquired by assigning $\alpha = 1$ and $\beta = 2$. However the smallest *best* operational cost is acquired by assigning $\alpha = 0.5$ and $\beta = 2$. It also shows that $\alpha = 0.5$ and $\beta = 2$ give the best slope -29.581. The best learning process is achieved by assigning $\alpha = 0.5$ and $\beta = 2$, see B.11, as it converges faster than assigning $\alpha = 1$ and $\beta = 2$, see B.12.

4.4.3 Pheromone value limit

Since we adapt the approach of MMAS we need to define the limit of pheromone value, $\lambda_{min}, \lambda_{max}$. Based on the definition in section 2.4.2, all pheromone values are initialized using λ_{max} and then modified over iterations but limited to $\lambda_{min} \leq \lambda \leq \lambda_{max}$. We conduct an experiment to investigate the best values for λ_{min} and λ_{max}. The results are summarized on table 4.4.

Based on the result in table 4.4, our ACO approach do the learning only if we set $\lambda_{min} = 0$, otherwise it achieves no learning. This can be confirmed by observing the slope value. When λ_{min} is set to 0 the ants can learn over each iteration, $slope < -25$,

Table 4.4: Experiment result on λ_{min} and λ_{max}. *min*, *mean* and *std* denote global best operational cost, mean value of all iterations and standard deviation value of all iteration respectively. *slope* denote the linear progression's trend line slope of operational cost over all iteration.

λ_{min}	λ_{max}	*min*	*mean*	*std*	*slope*
0	1	**115850**	**118776.3**	2053.69	-28.368
	1.5	116130	118875.2	2100.09	-29.952
	3	116040	118995.8	2202.11	**-31.269**
0.1	1	120760	122571.75	686.83	-0.349
	1.5	120100	122566.35	751.76	-0.668
	3	120520	122502.8	729.71	0.828
0.5	1	119940	122507.6	698.85	0.137
	1.5	119790	122534.05	730.75	-0.494
	3	120720	122549.65	693.07	0.053

otherwise they do not learn, as the slope is around 0. The best slope is acquired by assigning $\lambda_{min} = 0$ and $\lambda_{max} = 3$, where slope = -31.269. However the best cost and mean is acquired by assigning $\lambda_{min} = 0$ and $\lambda_{max} = 1$, best = 115850 and mean =118776.3. Since we want to keep our pheromone value in range $[0 \dots 1]$, we decide to assign $\lambda_{min} = 0$ and $\lambda_{max} = 1$.

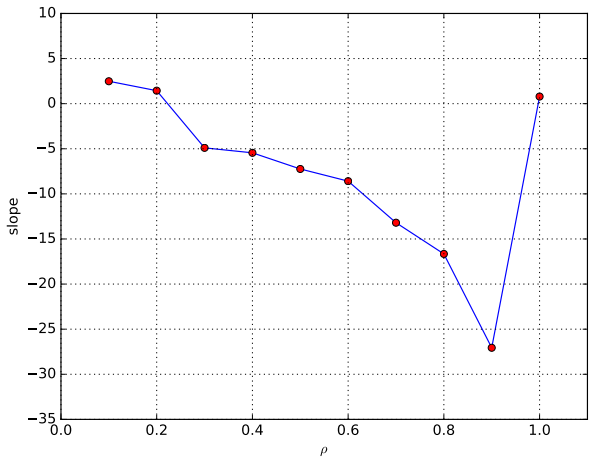
4.4.4 Evaporation Rate

Another parameter of ACO is the evaporation rate ρ. As we specified in 3.17, ρ is used as the evaporation rate of pheromone value. If we specify a big value for ρ it will evaporates slowly, otherwise it will evaporates quickly. We conduct an experiment using various value for ρ, $[0, 1, 0.2, \dots, 1]$. The result is shown in figure 4.4.4.

We assess the influence of ρ in our algorithm by using slope. As shown in figure 4.4.4, as the ρ value goes down the learning performance is getting worse since the pheromone value evaporates quickly. In contrast by assigning $\rho = 1$ the pheromone values will not evaporate. As a result the ants are constantly adding the pheromone value without reducing them and it will stuck at 1 since we used MMAS approach and set the default $\lambda_{max} = 1$. In either case assigning too small ρ value or $\rho = 1$ will make the ant not learn. It can be observed from the graph in 4.4.4, the slope is near 0 if we assign those values.

The best slope is achieved by assigning $\rho = 0.9$, pheromone value will evaporate slowly enough over each iteration thus producing an adequate environment for the ants to do the learning.

Figure 4.5: The influence of ρ throughout the learning capability of ACO (slope). ρ lies at x-axis while slope lies at y-axis.



4.4.5 Pheromone Matrix Experiment

As we mentioned before in section 3.5.2, we propose a pheromone matrix model with the size of $|L|\times|L|$. Initially we tried to implement a pheromone matrix with the size of $|P|\times|L|$ as proposed by [31]. However the performance of our approach using pheromone matrix $|P|\times|L|$ is not as expected. The operational cost did not improve over iteration and the ants did no learning. This behavior can be confirmed in figure 4.6.

As we can see, the scatter plot of performance of our algorithm using $|P|\times|L|$ pheromone matrix is bad. The result over iterations are scattered around with mean value 122465.32 and min value 119830. The slope is near to 0 which denote that the ants did not learn during the process. In contrast by using the proposed pheromone matrix with size $|L|\times|L|$, the algorithm performs better. The ants did learn during the first 100 iterations and they successfully improve the operational cost. Mean value of the results over all iterations is 117052.14 and minimal operational cost is 115450. The proposed pheromone matrix $|L|\times|L|$ successfully improves the performance of our algorithm and gives result improving the mean value by 5413.18 and minimal operational cost by 2777.86.

4.5 Comparison of Vehicle Selection Strategy

We conduct experiments across various problem instances to assess the influence of adding time window feature. We use 3 different number of lines: 5 lines, 10 lines and 20 lines. We also inspect 5 different slack widths specified by sl_p: 0, 10, 20, 30 and 40. The minimal operational cost output of each strategy is summarized in table 4.5.

Both PBU and ACO strategy use default parameter value. The greedy strategy is deterministic so it only output 1 operational cost. On the other hand, PBU and ACO strategy are executed with respectively 1000 and 200 iterations. We use 1000 iterations

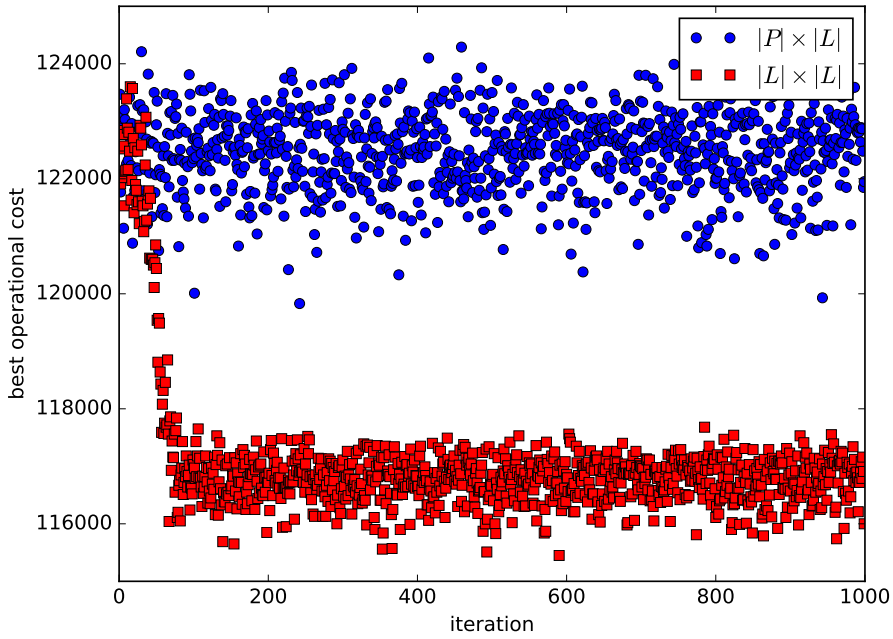


Figure 4.6: Comparison of best operational cost growth from two pheromone matrix in scatter plot over 1000 iterations. Blue dots denote the cost of $|P| \times |L|$ matrix while red squares denote the cost of $|L| \times |L|$ matrix

on PBU because the operational cost does not improve well over more iterations, thus extending the running time is not worth doing. We take the minimal and the mean value of all cost from all iterations.

As shown in table 4.5, in each strategy, the operational cost can always be reduced by introducing time window features. The resulting operational costs by assigning $sl_p = 0$ (no time window) are always worse than the operational cost by introducing time window $sl_p \neq 0$.

We can also observe that the resulting operational costs (min and mean value) of ACO approach outperform the resulting operational costs of the other strategies. The improvement is measured by a percentage as formalized in 4.10. By using ACO we can achieve maximum improvement of 28.42% while by using PBU we can only achieve maximum improvement of 17.91%, both when using problem instance with 20 lines and using $sl_p = 40$.

For the ACO approach the best minimal cost and mean value is achieved by using $sl_p = 30$ or $sl_p = 40$. However the best minimal cost output for PBU and greedy strategy are achieved by assigning $sl_p = 20$ or $sl_p = 30$.

Another results regarding number of bus and line transfer for the identical experiments are summarized in table 4.6.

The number of line transfers in the greedy strategy is always higher than in the other

Table 4.5: Operational cost output across various problem instances and time window slacks. #Line denotes number of line on each problem instance and sl_p denotes percentage of slack for time window, see subsection 4.1.3. *imp* denotes the improvement in percentage(%) of min cost compared to greedy cost.

Instance		Greedy	PBU			ACO		
#Line	sl_p		min	mean	imp	min	mean	imp
5	0	150600	142250	145490.74	5.54	136700	140445.7	9.23
	10	133580	128350	131836.52	3.92	123520	126695.85	7.53
	20	131200	**124320**	**126883.34**	5.24	120650	122148.05	8.04
	30	**130930**	126010	127743.46	3.76	**115900**	**118270.2**	11.48
	40	137550	131630	134188.13	4.30	116850	120974.7	15.05
10	0	189500	177780	184847.43	6.18	161420	173993.8	14.82
	10	184990	161330	166912.57	12.79	149240	158562.85	19.33
	20	**172500**	**155100**	162308.78	10.09	143510	152468.1	16.81
	30	175630	152420	**158457.46**	13.22	138220	**145658.9**	21.30
	40	177550	156850	163660.01	11.66	**137760**	146030.85	22.41
20	0	402240	366370	375495.1	8.92	302790	336444.65	24.72
	10	364510	326880	336697.48	10.32	285000	311927.05	21.81
	20	340840	284530	293567.62	16.52	256400	271727.95	24.77
	30	**339860**	**283420**	**292232.05**	16.61	251900	269527.8	25.88
	40	351890	288870	302411.96	17.91	**251870**	**269418.75**	28.42

strategies. This might be caused by the topology of the network. As we described in section 4.1.4, we adapt a star network topology. This might cause the distance between two terminals of different lines to be closer than distance to their each opposite terminal on the same line. As greedy approach always take a vehicle with largest utility value, a vehicle located in a closer terminal will be most likely assigned.

Another observation from table 4.6 is the number of buses is reduced by introducing time window features. Assigning $sl_p \neq 0$ produces results using less buses than assigning $sl_p = 0$. This influence of time window feature is affecting all strategy.

An anomaly is happened in number of bus result from ACO strategy. The number of buses is slightly larger than with greedy and PBU. This might happens because the number of line transfers with ACO is less. However this is not a big deal since the final operational cost of ACO strategy outperform them.

Beside the operational cost, number of bus and line transfer we also observe the running time of our approach. We summarized the running time of each strategy across

Table 4.6: Number of bus (#bus) and line transfer (#ltrf) across various problem instances and slacks.

Instance		Greedy		PBU		ACO	
#Line	sl_p	#bus	#ltrf	#bus	#ltrf	#bus	#ltrf
5	0	23	25	26	10	29	2
	10	21	18	22	7	26	3
	20	18	27	20	7	23	1
	30	19	31	18	3	22	2
	40	20	44	20	7	26	1
10	0	35	190	38	54	44	14
	10	33	232	37	41	41	8
	20	25	203	29	37	34	14
	30	27	201	29	38	35	11
	40	28	238	28	48	34	16
20	0	59	750	65	325	68	64
	10	53	741	58	227	62	41
	20	45	713	50	135	56	16
	30	44	783	48	146	47	30
	40	48	882	46	182	47	45

various problem instances in table 4.7.

As shown in table 4.7, the runtime is increased as the number of line is increased. It is caused by the increasing number of point of departures as the number of line goes up. A problem instance with 5 line has 700 point of departures, 10 line has 1400 point of departures, and 20 lines has 2800 point of departures.

Theoretically to construct a solution of bus assignment, in the worst case, our PBAP CP approach as presented in algorithm 5 have a running time $\mathcal{O}(kn^2)$ where k denotes number of bus and n denotes number of point of departures.

As expected, greedy strategy can solve the assignment problem very fast. The longest running time to find bus assignment for greedy strategy is 5 seconds. On the other hand the running time of PBU and ACO strategy to solve assignment problem is longer than greedy since they require t runs to construct a solution. t denotes the number of iterations for the PBU strategy and the number of iterations times the number of ants for ACO strategy.

The running time to find actual schedule $rt2$ in all strategy is similar for the same problem instance. A special condition is for a problem instance without time window

Table 4.7: Running time of each strategy across various problem instances. rt1 denotes running time (sec) of each strategy to find bus assignment. rt2 denotes running time (sec) of Constraint Optimizer to find such schedule with minimum distance to ideal departure time. Total denotes rt1 + rt2.

Instance		Greedy			PBU			ACO		
#Line	sl_p	rt1	rt2	total	rt1	rt2	total	rt1	rt2	total
5	0	1	0	1	46	0	46	163	0	163
	10	1	100	101	49	102	151	165	104	269
	20	1	88	89	54	78	132	179	75	254
	30	1	87	88	53	66	119	187	65	252
	40	1	98	99	58	66	124	207	64	271
10	0	1	0	1	111	0	111	485	0	485
	10	1	249	250	149	265	414	510	242	752
	20	1	173	174	130	208	338	508	189	697
	30	1	202	203	152	195	347	610	240	850
	40	1	279	280	156	251	417	649	213	862
20	0	2	0	2	392	0	392	1937	0	1937
	10	5	487	492	714	484	1198	1642	530	2172
	20	4	507	**511**	491	465	956	1954	560	2514
	30	3	423	426	615	363	978	2304	437	**2741**
	40	3	274	277	571	283	854	1917	303	2220

$sl_p = 0$. In this case the running time of constraint optimizer $rt2$ is equal to 0, as only 1 possible schedule is available.

The longest total running time, based on table 4.7, is acquired by applying ACO strategy to a problem instance with 20 line and $sl_p = 30$, which takes 2741 seconds or approximately 45 minutes. We consider this runtime quite acceptable since the problem instance is rather large.

The trade off of using time window features is measured by the distance between the ideal departure time and the scheduled departure time. As described in 3.7 we take objective function $\mathcal{J}$ as the measurement of total distance between ideal departure time and scheduled departure time. The experiment result on $\mathcal{J}$ values are summarized in table 4.8.

The average $\mathcal{J}$ per point of departure is measured by dividing $\mathcal{J}$ with total number of departure. As shown in table 4.8 the average trade-off for each problem instance is not so big, ≤ 5 minute. From this experiment we can learn that by adding a slight change to departure time we can achieve a better schedule, ±20% operational cost improvement.

Table 4.8: $\mathcal{J}$ output across various problem instances and time window slacks. #Line denotes number of line on each problem instance and #req denotes total number of request $|P|$. $\mathcal{J}$ denotes the total difference of schedule output to ideal departure time, see equation 3.7. $\mathcal{J}$ is measured in minute.

Instance		Greedy		PBU		ACO	
#Line	sl_p	$\mathcal{J}$	$\mathcal{J}$/#req	$\mathcal{J}$	$\mathcal{J}$/#req	$\mathcal{J}$	$\mathcal{J}$/#req
5	10	820	1.17	801	1.14	801	1.14
	20	1810	2.585	1831	2.615	1802	2.57
	30	2537	3.62	2609	3.727	2638	3.768
	40	3080	4.4	3305	4.72	3391	4.84
10	10	1638	1.17	1620	1.157	1603	1.145
	20	3733	2.666	3507	2.505	3548	2.53
	30	5286	3.775	5211	3.722	5196	3.711
	40	6579	4.699	6746	4.818	6808	4.86
20	10	3391	1.211	3416	1.22	3399	1.213
	20	6941	2.478	7122	2.543	7167	2.559
	30	9940	3.55	10511	3.75	10593	3.78
	40	12359	4.41	14019	5.006	14220	5.078

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The main purpose of this work was to introduce a time window feature to improve the operational cost of Public Bus Assignment Problem (PBAP). As the problem itself is a variant of VSP, which in most case is a NP-hard problem, we employ an approximative method namely Ant Colony Optimization. To tackle the constraint satisfaction problem, which appears as we introduce time window, we use a constraint propagator. In each constructive step we propagate corresponding constraint to make sure no contradiction occurs. We use our main approach to search for the best bus assignment that minimized operational cost. Then as a final step, we use constraint programming optimization to find an actual schedule with the smallest distance to ideal departure time.

In addition we develop a greedy and a probability based utility (PBU) selection strategy to assess the performance of our ACO approach. In our experiments we use greedy selection as the baseline to compare the performance of other strategies. We utilize PBU selection strategy to determine the best parameters for utility function.

We develop our ACO approach according to the guideline of Min-Max Ant System (MMAS). Moreover, we propose a smaller pheromone matrix compared to another PBAP previously tried.

Since our problem is not commonly addressed we simulated our own problem instances using a star network topology. Then we define the ideal departure time for each point of departure. Next we widen the ideal departure time, within a time window, according to some slack percentage on the difference between consecutive departures.

We conduct experiments using various problem instances each with different number of lines and different widths of time windows. We use 3 different number of lines: 5, 10 and 20 lines. As for the width of time window we use 5 different slack percentage: 0, 10,

20, 30, and 40. In total we have 15 different problem instances.

In all the experiments described, our ACO approach successfully outperform the other selection strategies. We also show that by introducing time window we can significantly reduce the operational cost with only small trade-off regarding a shift in departure times.

We measure the trade-off by total distance between departure time in the schedule produced and ideal departure time. The average distance in our experiment result is at maximum around 5 minute from ideal departure time.

Since our approach is based on constructive approach interleaved with a constraint propagation, we proved that by enforcing bound consistency we can imply satisfiability. This proof was necessary to ensure no contradiction occurs when we determine the actual schedule in final step without risk of failure.

5.2 Future Work

In future work we are interested to investigate the following aspects:

- More realistic problem instances
- Different cost functions
- Parallel mechanism
- Constraint optimization technique

More realistic problem instances need to be investigated in the future research since we want to see whether our approach is adequate to solve problem instances with other constraints, such as various types of vehicle (heterogeneous fleet) and restrictions for some buses to serve certain routes (compatibility constraint).

It also catch our interest to see whether modifying the cost function might fix the anomaly of the weight of waiting time term in section 4.3. By adding the waiting time term to the cost function we assume that it does not affect the feasible solutions but influences the operational cost.

To speed up the execution time we might also want to implement a parallel mechanism. As the nature of the ACO approaches consists of individual work of several ants, it is very likely that the algorithm runtime may be speed up by using parallel mechanism.

A better constraint optimization technique might also be an interesting work to do. It can affect the search process to produce better actual schedules which have less distance to ideal departure time.

Bibliography

[1] A. A. Bertossi, P. Carraresi, and G. Gallo. "On some matching problems arising in vehicle scheduling models". In: *Networks* 17.3 (1987), pp. 271–281. ISSN: 1097-0037. DOI: 10.1002/net.3230170303. URL: http://dx.doi.org/10.1002/net.3230170303.

[2] C. Blum. "Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling". In: *Computers & Operations Research* 32.6 (2005), pp. 1565 –1591. ISSN: 0305-0548. DOI: http://dx.doi.org/10.1016/j.cor.2003.11.018. URL: http://www.sciencedirect.com/science/article/pii/S0305054803003599.

[3] V. Černỳ. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: *Journal of optimization theory and applications* 45.1 (1985), pp. 41–51.

[4] *Dia Diagram Editor*. URL: http://dia-installer.de/ (visited on 02/10/2016).

[5] M. Dorigo. "Optimization, Learning and Natural Algorithms (in Italian)". PhD thesis. Milan, Italy: Dipartimento di Elettronica, Politecnico di Milano, 1992.

[6] M. Dorigo and L. M. Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 53–66. DOI: http://dx.doi.org/10.1109/4235.585892.

[7] M. Dorigo, V. Maniezzo, and A. Colorni. *Positive Feedback as a Search Strategy*. Tech. rep. 91-016. Milan, Italy: Dipartimento di Elettronica, Politecnico di Milano, 1991.

[8] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant System: Optimization by a Colony of Cooperating Agents". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part B* 26.1 (1996), pp. 29–41. DOI: http://dx.doi.org/10.1109/3477.484436.

[9] E. C. Freuder. "Synthesizing Constraint Expressions". In: *Commun. ACM* 21.11 (Nov. 1978), pp. 958–966. ISSN: 0001-0782. DOI: 10.1145/359642.359654. URL: http://doi.acm.org/10.1145/359642.359654.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[11] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. 2nd. Springer Publishing Company, Incorporated, 2010. ISBN: 1441916636, 9781441916631.

[12] F. Glover. "Future Paths for Integer Programming and Links to Artificial Intelligence". In: *Comput. Oper. Res.* 13.5 (May 1986), pp. 533–549. ISSN: 0305-0548. DOI: 10.1016/0305-0548(86)90048-1. URL: http://dx.doi.org/10.1016/0305-0548(86)90048-1.

[13] T. F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC, 2007. ISBN: 1584885505.

[14] P. V. Hentenryck, V. Saraswat, and Y. Deville. "Design, implementation, and evaluation of the constraint language cc(FD)". In: *The Journal of Logic Programming* 37.1–3 (1998), pp. 139 –164. ISSN: 0743-1066. DOI: http://dx.doi.org/10.1016/S0743-1066(98)10006-7. URL: http://www.sciencedirect.com/science/article/pii/S0743106698100067.

[15] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0-262-58111-6.

[16] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[17] JetBrains. *IntelliJ IDEA*. URL: https://www.jetbrains.com/idea/ (visited on 02/10/2016).

[18] K. A. D. Jong. "Genetic Algorithms Are NOT Function Optimizers". In: *Foundations of Genetic Algorithms 2*. Ed. by D. L. Whitley. San Francisco, CA: Morgan Kaufmann, 1993, pp. 5–17. URL: http://www.mpi-sb.mpg.de/services/library/proceedings/contents/foga92.html.

[19] M. Khichane, P. Albert, and C. Solnon. "Integration of ACO in a Constraint Programming Language". English. In: *Ant Colony Optimization and Swarm Intelligence*. Ed. by M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. Winfield. Vol. 5217. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 84–95. ISBN: 978-3-540-87526-0. DOI: 10.1007/978-3-540-87527-7_8. URL: http://dx.doi.org/10.1007/978-3-540-87527-7_8.

[20] M. Khichane, P. Albert, and C. Solnon. "Strong Combination of Ant Colony Optimization with Constraint Programming Optimization." In: *CPAIOR*. Ed. by A. Lodi, M. Milano, and P. Toth. Vol. 6140. Lecture Notes in Computer Science. Springer, June 22, 2010, pp. 232–245. URL: http://dblp.uni-trier.de/db/conf/cpaior/cpaior2010.html#KhichaneAS10.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". In: *SCIENCE* 220.4598 (1983), pp. 671–680.

[22] J. K. Lenstra and A. H. G. R. Kan. "Complexity of vehicle routing and scheduling problems." In: *Networks* 11.2 (1981), pp. 221–227. URL: http://dblp.uni-trier.de/db/journals/networks/networks11.html#LenstraK81.

[23] S. Lin and B. W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". In: *Oper. Res.* 21.2 (Apr. 1973), pp. 498–516. ISSN: 0030-364X. DOI: 10.1287/opre.21.2.498. URL: http://dx.doi.org/10.1287/opre.21.2.498.

[24] A. Mackworth. "Consistency in Networks of Relations". In: *Artificial Intelligence* 8.1 (1977). Reprinted in Readings in Artificial Intelligence, B. L. Webber and N. J. Nilsson (eds.), Tioga Publ. Col., Palo Alto, CA, pp. 69-78, 1981. This paper was honoured in Artificial Intelligence 59, 1-2, 1993 as one of the fifty most cited papers in the history of Artificial Intelligence. It also received the 2013 AIJ Classic Paper Award: http://www.journals.elsevier.com/artificial-intelligence/news/announcing-winners-of-the-2013-aij-classic-paper-award/, pp. 99–118.

[25] B. Meyer. "Hybrids of Constructive Metaheuristics and Constraint Programming: A Case Study with ACO." In: *Hybrid Metaheuristics*. Ed. by C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels. Vol. 114. Studies in Computational Intelligence. Springer, Jan. 31, 2009, pp. 151–183. ISBN: 978-3-540-78294-0. URL: http://dblp.uni-trier.de/db/series/sci/sci114.html#Meyer08.

[26] B. Meyer and A. Ernst. "Integrating ACO and Constraint Propagation". English. In: *Ant Colony Optimization and Swarm Intelligence*. Ed. by M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, and T. Stützle. Vol. 3172. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 166–177. ISBN: 978-3-540-22672-7. DOI: 10.1007/978-3-540-28646-2_15. URL: http://dx.doi.org/10.1007/978-3-540-28646-2_15.

[27] U. Montanari. "Networks of constraints: Fundamental properties and applications to picture processing". In: *Information Sciences* 7 (1974), pp. 95 –132. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/0020-0255(74)90008-5. URL: http://www.sciencedirect.com/science/article/pii/0020025574900085.

[28] C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241 & COSLING S.A.S. 2014. URL: http://www.choco-solver.org.

[29] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc., 2006. ISBN: 0444527265.

[30] S. Sahni and T. Gonzalez. "P-Complete Approximation Problems". In: *J. ACM* 23.3 (July 1976), pp. 555–565. ISSN: 0004-5411. DOI: 10.1145/321958.321975. URL: http://doi.acm.org/10.1145/321958.321975.

[31] D. F. Semedo. "A Public Transport Bus Assignment Problem: Parallel Metaheuristics Assessment". MA thesis. Lisboa, Portugal: Faculdade Ciencas Universidade Nova de Lisboa, 2015.

[32] T. Stützle and H. Hoos. "Improvements on the Ant-System: Introducing the MAX-MIN Ant System". English. In: *Artificial Neural Nets and Genetic Algorithms*. Springer Vienna, 1998, pp. 245–249. ISBN: 978-3-211-83087-1. DOI: 10.1007/978-3-7091-6492-1_54. URL: http://dx.doi.org/10.1007/978-3-7091-6492-1_54.

[33] T. Stützle and H. H. Hoos. "MAX–MIN Ant System". In: *Future Generation Computer Systems* 16.8 (2000), pp. 889 –914. ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/S0167-739X(00)00043-1. URL: http://www.sciencedirect.com/science/article/pii/S0167739X00000431.

[34] D. Thiruvady, G. Singh, A. T. Ernst, and B. Meyer. "Constraint-based {ACO} for a shared resource constrained scheduling problem". In: *International Journal of Production Economics* 141.1 (2013). Meta-heuristics for manufacturing scheduling and logistics problems, pp. 230 –242. ISSN: 0925-5273. DOI: http://dx.doi.org/10.1016/j.ijpe.2012.06.012. URL: http://www.sciencedirect.com/science/article/pii/S092552731200240X.

[35] D. Thiruvady, I. Moser, A. Aleti, and A. Nazari. "Constraint Programming and Ant Colony System for the Component Deployment Problem". In: *Procedia Computer Science* 29 (2014). 2014 International Conference on Computational Science, pp. 1937 –1947. ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2014.05.178. URL: http://www.sciencedirect.com/science/article/pii/S187705091400355X.

[36] D. R. Thiruvady, C. Blum, B. Meyer, and A. T. Ernst. "Hybridizing Beam-ACO with Constraint Programming for Single Machine Job Scheduling." In: *Hybrid Metaheuristics*. Ed. by M. J. Blesa, C. Blum, L. D. Gaspero, A. Roli, M. Sampels, and A. Schaerf. Vol. 5818. Lecture Notes in Computer Science. Springer, Oct. 12, 2009, pp. 30–44. ISBN: 978-3-642-04917-0. URL: http://dblp.uni-trier.de/db/conf/hm/hm2009.html#ThiruvadyBME09.

[37] D. R. Thiruvady, B. Meyer, and A. T. Ernst. "Car sequencing with constraint-based ACO." In: *GECCO*. Ed. by N. Krasnogor and P. L. Lanzi. ACM, 2011, pp. 163–170. ISBN: 978-1-4503-0557-0. URL: http://dblp.uni-trier.de/db/conf/gecco/gecco2011.html#ThiruvadyME11.

Appendix A

Proof Bound Consistency implies Satisfiability

Theorem: Let $\mathbf{P}$ be a problem over a set of $\mathbf{X}$ of n variables, $\mathbf{x}_1,\ldots,\mathbf{x}_n$, each with a convex domain $\mathbf{D}_i = \mathbf{a}_i,\ldots,\mathbf{b}_i$, composed of a sequence of difference constraints $\mathbf{p}_i \leq \mathbf{x}_{i+1} - \mathbf{x}_i$ between consecutive variables $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$. If it is possible to make $\mathbf{S}$ bounds-consistent, then any value $\mathbf{v}_i$ in the domain of any variable $\mathbf{x}_i (\mathbf{v}_i \in \mathbf{D}_i)$ belongs to a solution of $\mathbf{P}$

Proof by induction on the number of variables.
Base Step ($n = 2$).
For 2 variables, imposing bounds-consistency results in narrowing the domain of the variables to

$$\mathbf{x}_1^+ = \min(\mathbf{b}_1, \mathbf{b}_2 - \mathbf{p}_1), \qquad \mathbf{x}_2^- = \max(\mathbf{a}_2, \mathbf{a}_1 + \mathbf{p}_1) \tag{A.1}$$

where $\mathbf{x}_1^+$ and $\mathbf{x}_2^-$ denote respectively the new upper bound of $\mathbf{x}_1$ and new lower bound of $\mathbf{x}_2$. Since the bounds for each variable are narrowed but not empty, otherwise it is not bounds-consistent, it can be easily checked that any value $\mathbf{v}_1$ (resp. $\mathbf{v}_2$) in the domain of $\mathbf{x}_1$ (resp. $\mathbf{x}_2$) has a support on variable $\mathbf{x}_2$ (resp. $\mathbf{x}_1$)
Induction Step ($n > 2$)
Let us assume that the result stands for problem $\mathbf{P}_n$ with n variables, i.e. for any sequence of difference constraints between consecutive variables that is bounds-consistent, it is the case that any value of any variable belongs to a solution. We may now analyse a problem $\mathbf{P}$ with $n+1$ variables.
If it is not possible to impose bounds-consistency on the whole problem then the problem is unsatisfiable, since imposing bounds-consistency does not eliminate any solution. In the case of interest, after making problem $\mathbf{P}_n$ bounds consistent, its variable $\mathbf{x}_n$ will have domain $\mathbf{a}_n \ldots \mathbf{b}_n$. Problem $\mathbf{P}$, resulting from $\mathbf{P}_n$ by the addition of a variable $\mathbf{x}_{n+1}$ with

initial domain $\mathbf{a}_{n+1} \ldots \mathbf{b}_{n+1}$ can be made bounds consistent and variables $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$ will have new bounds for their domains

$$\mathbf{x}_n^+ = \min(\mathbf{b}_n, \mathbf{b}_{n+1} - \mathbf{p}_n), \qquad \mathbf{x}_{n+1}^- = \max(\mathbf{a}_{n+1}, \mathbf{a}_n + \mathbf{p}_n) \tag{A.2}$$

where $\mathbf{x}_n^+$ and $\mathbf{x}_{n+1}^-$, respectively, denote the new upper bound of $\mathbf{x}_n$ and lower bound of $\mathbf{x}_{n+1}$. Note that $\mathbf{x}_n^+ \geq \mathbf{a}_n$ and $\mathbf{x}_{n+1}^- \leq \mathbf{b}_{n+1}$ since we consider bounds consistency. The new domain of $\mathbf{x}_n$ is included in the old domain since $\mathbf{x}_n^+ \leq \mathbf{b}_n$. Consequently any value $\mathbf{v}_{n+1}$ of $\mathbf{x}_{n+1}$ has a support in some value $\mathbf{v}_n$ of $\mathbf{x}_n$. On the other hand, any value $\mathbf{v}_n$ of $\mathbf{x}_n$ within reduced domain is a solution of problem $\mathbf{P}_n$, by the induction hypothesis, i.e. there is a tuple

$$< \mathbf{x}_1 = \mathbf{v}_1, \mathbf{x}_2 = \mathbf{v}_2, \ldots, \mathbf{x}_n = \mathbf{v}_n >,$$

which is a solution of problem $\mathbf{P}_n$. Hence, extending this tuple with the value for variable $\mathbf{x}_{n+1}$

$$< \mathbf{x}_1 = \mathbf{v}_1, \mathbf{x}_2 = \mathbf{v}_2, \ldots, \mathbf{x}_n = \mathbf{v}_n, \mathbf{x}_{n+1} = \mathbf{v}_{n+1} >,$$

guarantees that any value $\mathbf{v}_{n+1}$ of $\mathbf{x}_{n+1}$ is a solution of the problem $\mathbf{P}$ with $n+1$ variables.

Complexity

A problem $\mathbf{P}$ defined as above has polynomial complexity, since a solution of $\mathbf{P}$ can be obtained without deep backtracking. In fact, a solution can be found as follows.

First, an arbitrary variable and an arbitrary value in its domain can be selected (with no loss of generality let us make $\mathbf{x}_1 = \mathbf{v}_1$) and bounds consistency is imposed on the resulting network of binary constraints.

This procedure can be subsequently iterated over the other variables: pick an arbitrary variable and value in its domain (possibly narrowed by imposing bounds-consistency in the previous iteration) and impose bounds consistency on the remaining network.

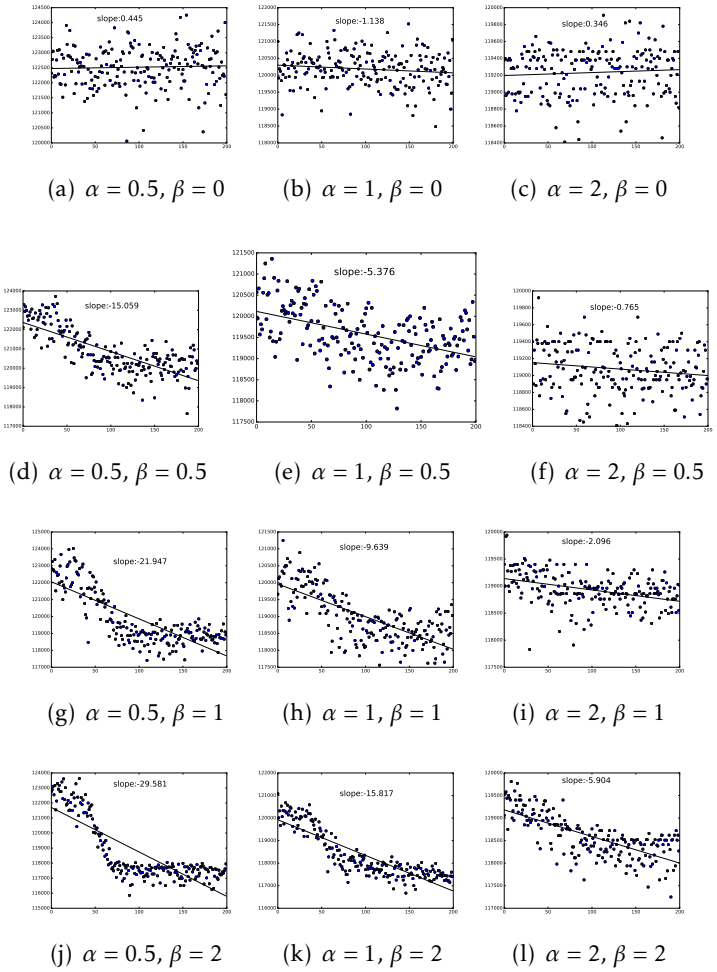
At each step, the value can be arbitrarily chosen, since any value of the selected variable belongs to some solution when the network is bounds consistent.

To solve the problem, one value for each of the n variables must be selected, and these n selections might be interleaved while maintaining bounds-consistency on the remaining network.

Maintaining bound-consistency in a network of binary constraints over n variables with domains of size d, has worst case complexity $\mathcal{O}(ad^2)$. In this case, $a = n-1$ and hence the worst case complexity is $\mathcal{O}(nd^2)$, resulting in an overall worst-case complexity for finding a solution of $\mathcal{O}(n^2d^2)$

APPENDIX B

Scatter plot on different α and β



(a) $\alpha = 0.5, \beta = 0$ (b) $\alpha = 1, \beta = 0$ (c) $\alpha = 2, \beta = 0$

(d) $\alpha = 0.5, \beta = 0.5$ (e) $\alpha = 1, \beta = 0.5$ (f) $\alpha = 2, \beta = 0.5$

(g) $\alpha = 0.5, \beta = 1$ (h) $\alpha = 1, \beta = 1$ (i) $\alpha = 2, \beta = 1$

(j) $\alpha = 0.5, \beta = 2$ (k) $\alpha = 1, \beta = 2$ (l) $\alpha = 2, \beta = 2$

Figure B.1: Overview scatter plot and linear regression on different α and β value.

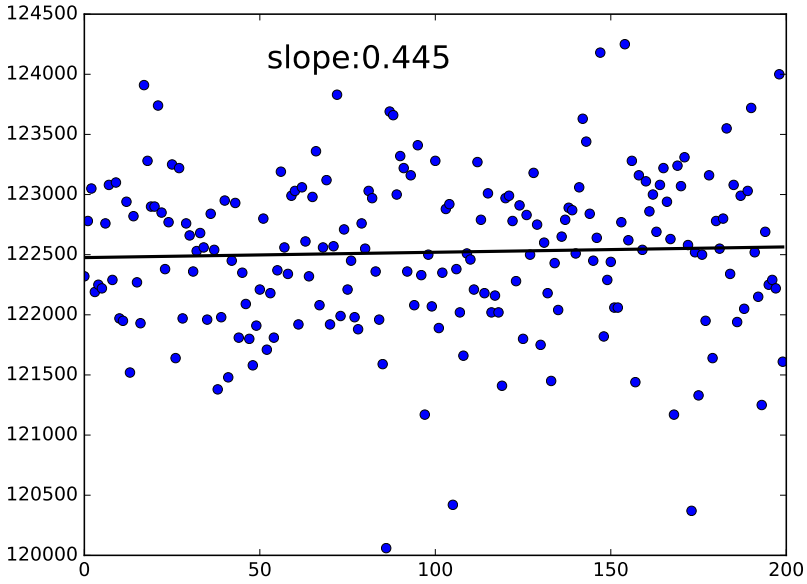


Figure B.2: Scatter plot and linear regression on $\alpha = 0.5$ and $\beta = 0$

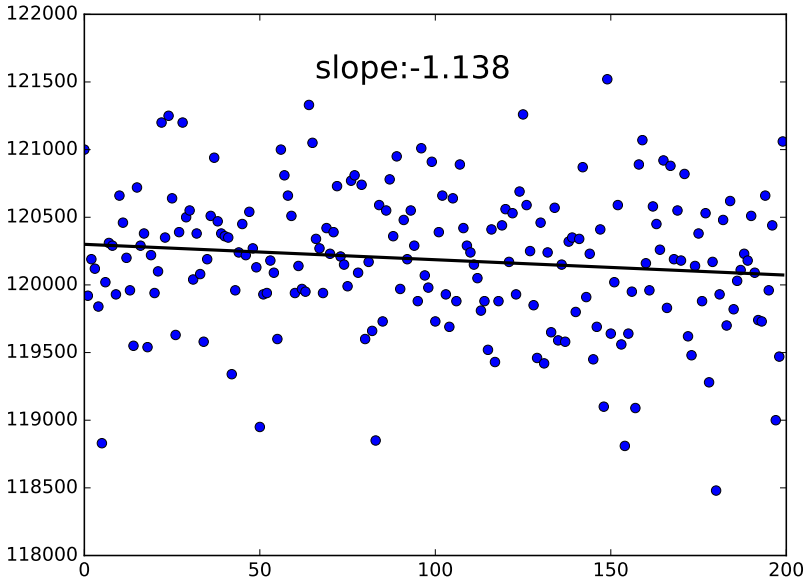


Figure B.3: Scatter plot and linear regression on $\alpha = 1$ and $\beta = 0$

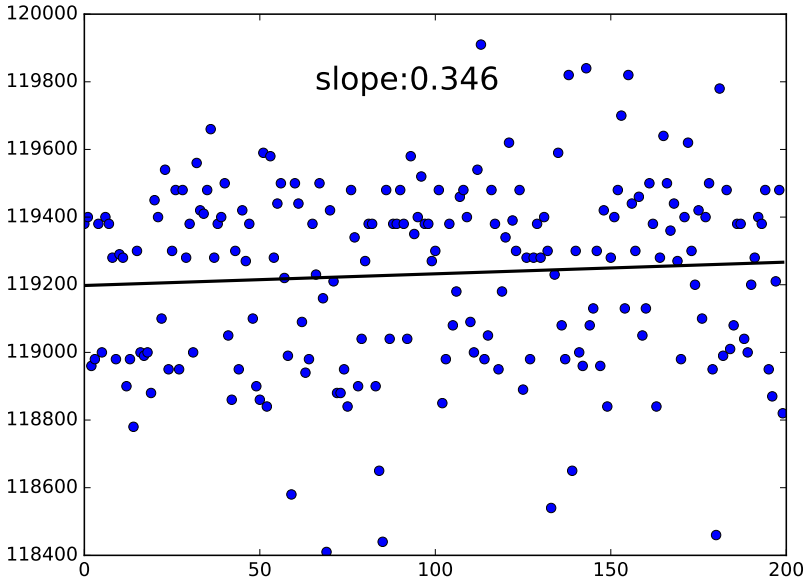


Figure B.4: Scatter plot and linear regression on $\alpha = 2$ and $\beta = 0$

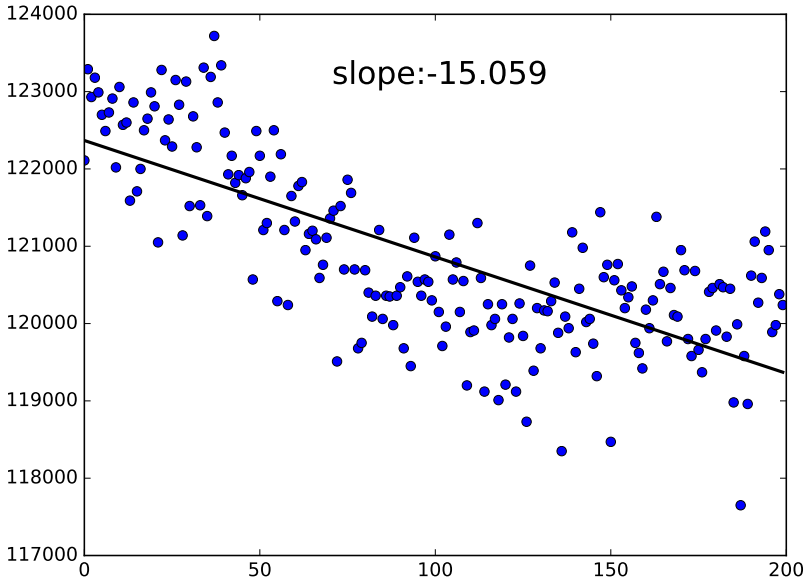


Figure B.5: Scatter plot and linear regression on $\alpha = 0.5$ and $\beta = 0.5$

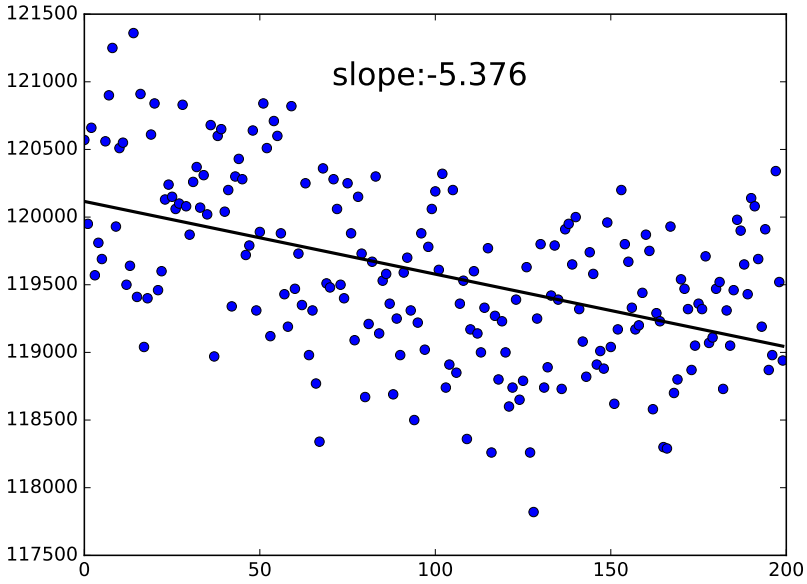


Figure B.6: Scatter plot and linear regression on $\alpha = 1$ and $\beta = 0.5$

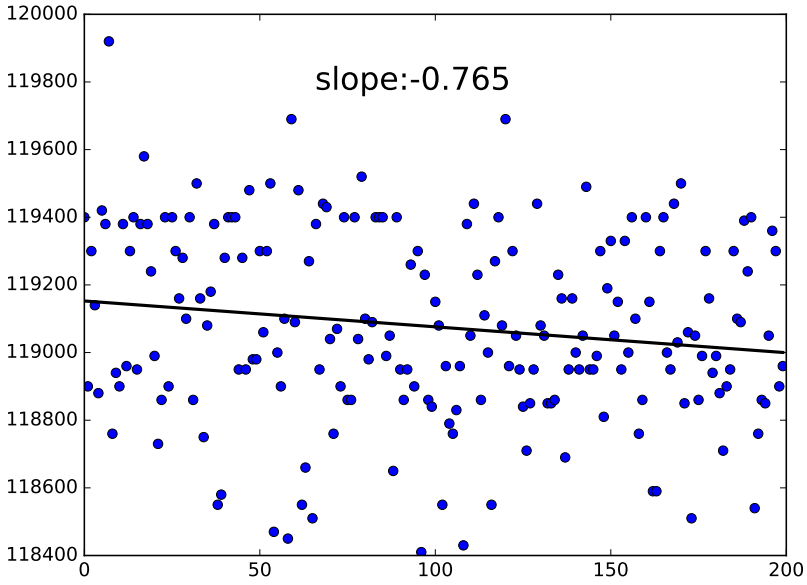


Figure B.7: Scatter plot and linear regression on $\alpha = 2$ and $\beta = 0.5$

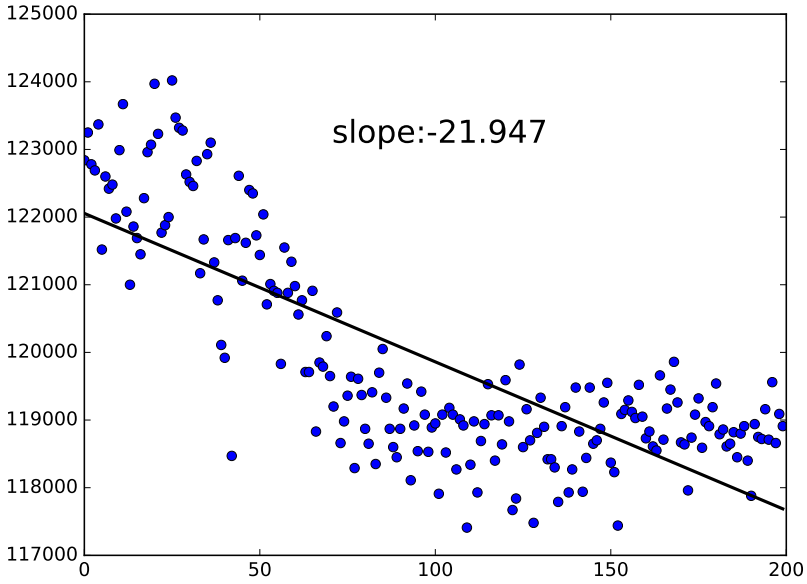


Figure B.8: Scatter plot and linear regression on $\alpha = 0.5$ and $\beta = 1$

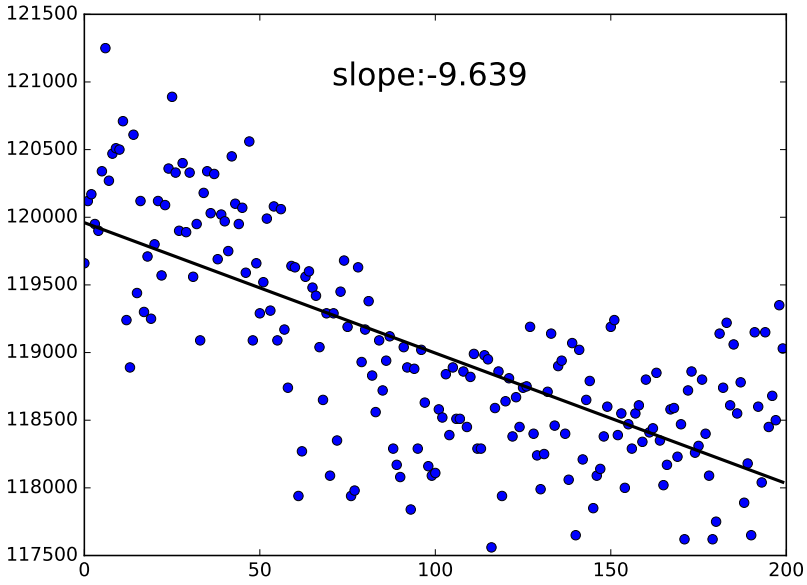


Figure B.9: Scatter plot and linear regression on $\alpha = 1$ and $\beta = 1$

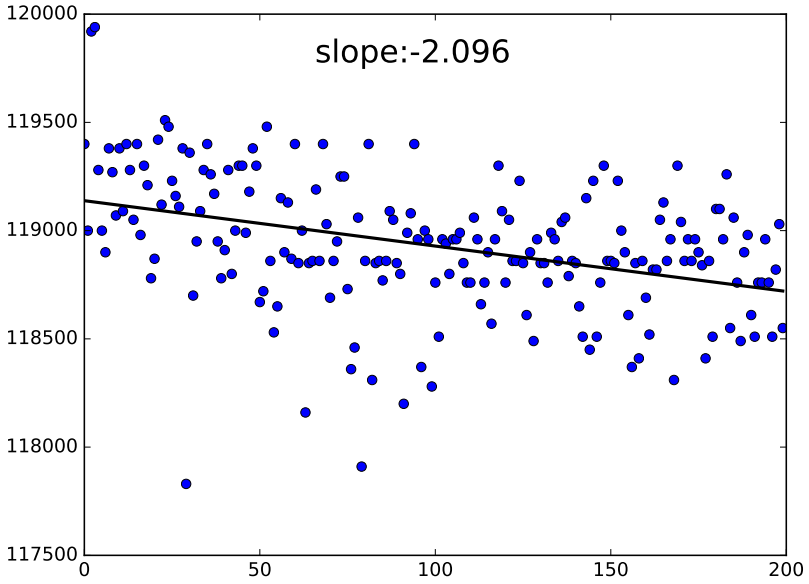


Figure B.10: Scatter plot and linear regression on $\alpha = 2$ and $\beta = 1$

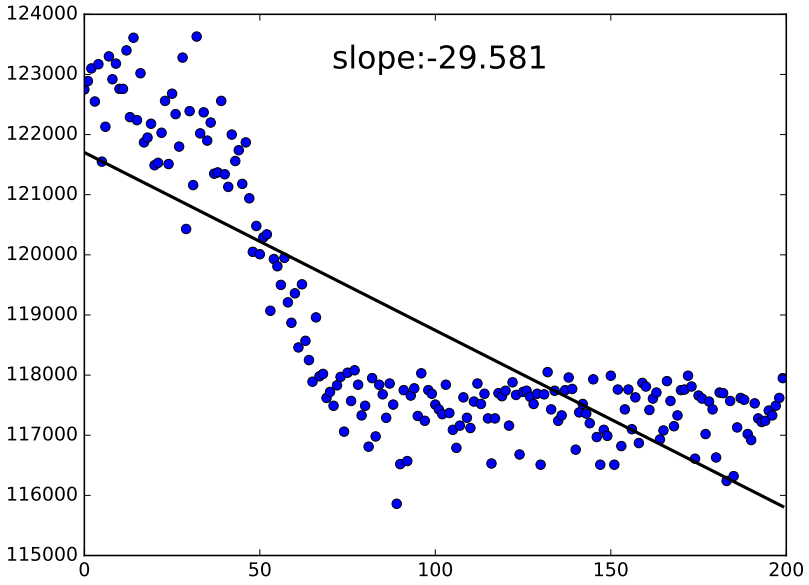


Figure B.11: Scatter plot and linear regression on $\alpha = 0.5$ and $\beta = 2$

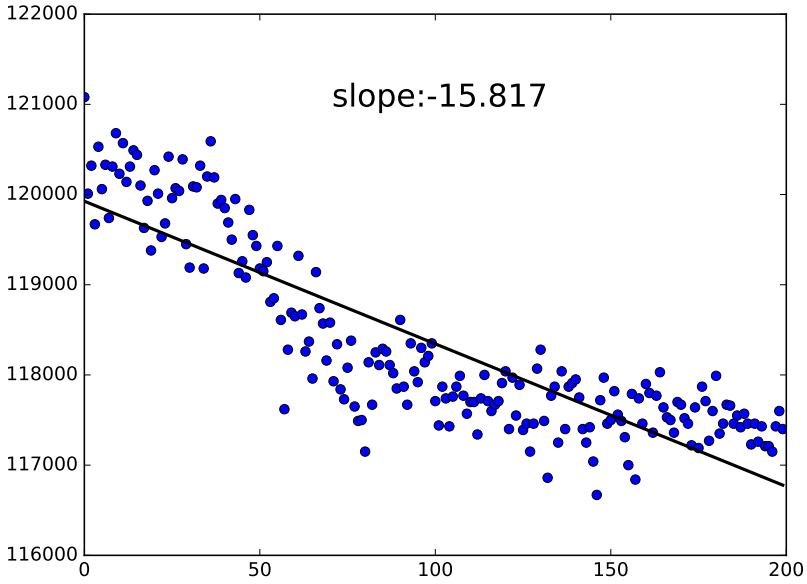


Figure B.12: Scatter plot and linear regression on $\alpha = 1$ and $\beta = 2$

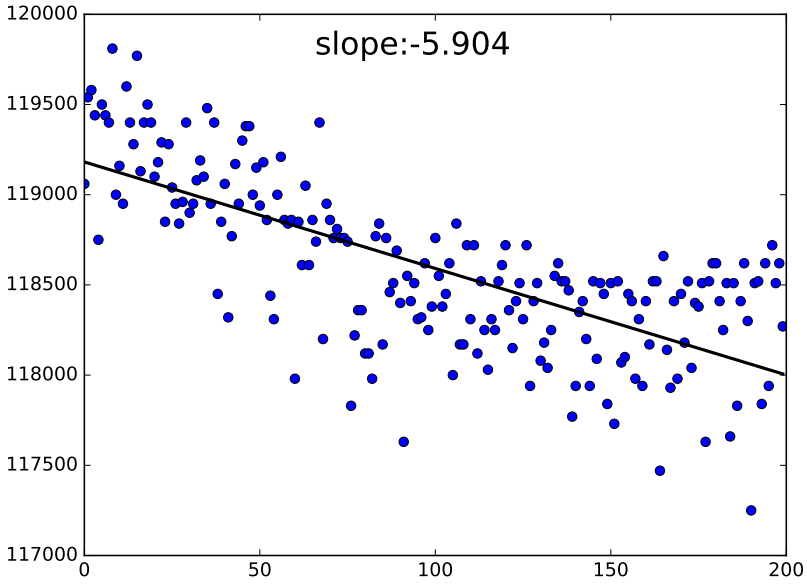


Figure B.13: Scatter plot and linear regression on $\alpha = 2$ and $\beta = 2$