INFORMATICS: OPTIMIZATION

Master Thesis

Exact methods for solving the small scale single vehicle pickup and delivery problem with time windows.

Author: Yacine Lakel

Supervisor: Dag Haugland

August 15, 2016

Acknowledgement

I would first like to start off by thanking Dag Haugland for his guidance, commitment, and patience, the Department for Informatics for providing a good environment for students to thrive and my fellow colleagues who have provided years of friendship, inspiration and support. Finally, I would like to thank Silje Marie Smitt for the constant encouragement throughout this process. I have learned so much through this difficult journey.

Abstract

The single vehicle pickup and delivery problem with time windows (*1-PDPTW)* is a well-known problem in transportation and combinatorics. In practice, these routes are relatively small with less then 30 stops. In this thesis, we consider two exact methods to solve this problem for small instance sizes. The first is solving the corresponding MIP-model using commercial linear programming software and the second is a forward dynamic programming algorithm. The motivation for this thesis is to determine the usability of exact methods as a heuristic to solve the multi vehicle pickup and delivery problem.

Contents

List of Figures

Listings

1 Introduction

In the transportation sector, developing efficient routes is key in terms of reducing both the total company costs and route durations. A well known problem that is derived from optimizing these types of routes is the *General Pickup and Delivery Problem with Time Windows* (**PDPTW**). In this problem, a fleet of vehicles must service a set of transportation requests, consisting of a pickup and delivery location. Additionally, each location has a given time window in which a vehicle must have serviced the location. This problem consists of two major tasks: assigning subset of the transportation requests to each vehicle and finding a route to service their requests. The goal is to minimize the total cost of servicing all transportation requests. Multiple factors can effect this cost, such as the number of vehicles used, the total duration and total travelled distance.

In designing an algorithm to find minimal cost solutions to this problem, one can divide the algorithm into two problems. The first problem is a assigning transportation requests to each vehicle, while the second is a simplified version of our original problem, where there is only one vehicle available. This simplified version of the problem is named the *Single Vehicle Pickup and Delivery Problem with Time Windows* (**1-PDPTW**). An instance of a PDPTW can thus generate multiple instances of the 1-PDPTW. In practice, solutions to the PDPTW will usually generate instances of the 1-PDPTW with a size of less then 10-15 requests (20-30 stops). Given this relatively small size, exact methods can be considered to solve each 1-PDPTW instance.

The goal for this thesis is to analyze the practical and theoretical capabilities of solving small scale 1-PDPTW problems with exact methods. We will first give both a formal and mathematical formulation of the 1-PDPTW and then give a review of the literature related to this problem. For our experimental tests, we will consider two exact methods to solve this problem. The first method is by solving the underlying MIP-model using Gurobi, a commercial MIP-solver. The second method is a forward dynamic programming algorithm developed by Desoires et al.[2] to solve the *Single Vehicle Dial-a-Ride Problem*. By analysing the results from our experimental , we will determine how efficient the exact methods are in terms of computational time and how they perform as the route size increases.

2 Literature Review

The 1-PDPTW belongs to a class of problems named *Vehicle Routing Problems* (**VRP**). These types of problems have been studied for over 50 years, greatly influenced by the introduction to vehicle routing problems by Danzig and Ramser in 1959 [1]. In the general VRP, goods must be delivered from a depot by a fleet of vehicles to a set of customers. The problem is to construct vehicle routes that serve all customers while the sum of the total distances of all routes are minimized. A valid set of vehicle routes must satisfy the following three constraints: each vehicle starts and ends at the depot, each customer is visited exactly once by one vehicle, and the capacity of the vehicle must not be exceeded.

To solve the VRP, one must consider two sub problems. The first one is to assign a subset of customers to each vehicle and the second one is to find the minimal route that starts and ends at the depot and only visits a vehicle's assigned customer exactly once. The second sub problem is a constrained variation of one of the most famous NP-hard problems in combinatorial optimization, named the *Travelling Salesman Problem* (**TSP**). This is the simplest type of vehicle routing problem. In this problem you are given a set locations to visit and distances between each pair of location. The goal is to find shortest route that visits each location exactly once and returns to the city the route originated at. Thus, the VRP is a generalization of the TSP and is therefore NP-hard.[1] The VRP with *Backhauls* (**VRPB**) is a variation of the VRP where both delivery and pickup requests are considered. The customers are thus partitioned into two subsets: delivery requests and pickup requests. A delivery request requires a vehicle to deliver a given load from the depot to a customer, while a pickup request requires a vehicle to pickup a certain load at a customers location and deliver it to the depot. Toth and Vigo (2002) provide a comprehensive introduction on the VRP and many of its variations[11].

While the VRP and VRPB deals with the transportation of goods between the depot and customers, the *Pickup and Delivery* problem (**PDP**), an extension of the VRPB, considers the transportation of goods or persons between pickup and delivery locations without transshipment at intermediate locations. In particular, the PDP problem that deals with the transportation of persons is defined as the *Dial-a-Ride* problem (**DARP**)[10]. In the PDP (and DARP), each vehicle is assigned a set of transportation requests. Each request

[1] All variations of the VRP stated in this section are NP-hard due to the underlying TSP sub problem.

consists of a pickup location and its corresponding delivery location that must be serviced in a similar manner as in the VRP. Thus, the PDP includes an additional precedence constraint, where a vehicle must visit a pickup location before it can visit its corresponding delivery location.

The PDP, as well as the other problems stated previously, can be further extended with the addition of time windows. This extension is referred to the PDP with *time windows* (**PDPTW**). Time windows are time intervals assigned to each location (customers and depot) in which the service at the location must be started. In the general PDPTW, if a vehicle arrives at a location before the time window, the vehicle must wait until it is able to service the request. [2]

The PDPTW problem has a special instance where only one vehicle is available to service all the customers. In this special instance, the assigning of customers to vehicles is not needed, thus we only consider the problem of finding the minimal route that services all customers. This instance is defined as the *Single Vehicle* PDPTW (**1-PDPTW**). The general instance is often denoted as the **m-PDPTW**. [3] Note that the 1-PDPTW is a variation of the TSP, where precedence, capacity and time window constraints are added.

Solution approaches to PDP problems vary depending on the variation of that problem. Savelsbergh's et al. [10] survey on variations of PDP problems uses three distinguishing characteristics. Based on the combination of characteristics, a new solution approach might be needed. The first characteristic is if the problem is *static* or *dynamic*. In a static PDP, all requests are determined in advance of constructing the routes, while dynamic problems allow some requests to arrive during the execution of the route(s). Note that all problems stated previously have been static variations. The second characteristic is *time windows*. The inclusion of time windows adds an additional set of constraints to the PDP which makes the PDPTW more complex. The third characteristic is the *number of vehicles*, specifically if the problem is a 1-PDP or a m-PDP. Instances of the m-PDP require two sub problems to be solved, while instances of the (1-PDP) require only one sub problem to be solved (as stated in the previous paragraph). Furthermore, choosing an exact approach

[2] Wang [13] distinguishes between two types of time windows: hard and soft. Soft time windows allow a vehicle to violate the time window at a given penalty cost. Hard time windows, on the other hand, must be strictly held.

[3] The notation distinguishing between one and many vehicles is commonly used in vehicle routing problems.[10]

over an approximate approach can lead to a faster time complexity at the cost of higher cost routes.

In finding optimal solutions for the 1-PDPTW, the main approaches are algorithms using dynamic programming. Psaraftis et al. (1983) [9] and Desrosiers et al (1986) [2] both propose a dynamic programming approach to solve the 1-PDPTW. Psartif previously proposed a dynamic programming approach using backwards recursion to solve the 1-DARP which was then modified to use forward recursion as well as a state elimination procedure to remove infeasible states. The algorithm has a time complexity of $O(n^2 3^n)$.

Desrosiers presents a similar approach to solve the 1-DARP with a more elaborate elimination criteria due to the structure in which states are represented. This algorithm proved very effective when time windows are tight and vehicle capacity is small. Despite the problems complexity, when capacity constraints are considerably tight, the running time seems to increase only linearly with problem size. The algorithm has been tested successfully on problems with up to 40 transportation requests (80 customers). This algorithm was developed to solve the large-scale 1-DARP.

Many techniques have been used when developing approximate solutions. Bruggen et al. (1993) [12] developed a heuristic consisting of a 2-phase approach. The algorithm first creates an initial feasible solution and then tries to improve on this solution. This heuristic uses a variable depth arc exchange procedure in both phases, where the number of arc exchanges is dynamically calculated during the search. This algorithm can reach near optimal solutions for problem sizes of up to 38 customers. A problem that arises from this algorithm is that it does not guarantee a solution as well as producing low quality solutions. This occurs when the 2-phase procedure is trapped in local optima. The author proposes a meta heuristic that uses simulated annealing as an alternative approach, but at the cost of high processing time.

Jih and Hsu[6][7] have produced two algorithms that utilizes genetic algorithms. The algorithm produced in 1999 [6] combines dynamic programming and a genetic algorithm. This approach creates sets of sub routes which are then sent to a genetic algorithm which create an initial population. The genetic algorithm uses crossover operators to mutate solutions which are encoded as permutations of the locations. Jih and Hsu (2002) [7] later proposes a family competition genetic algorithm to improve on their previous work.

The work of Landrieu et al. (2001)[8] utilizes a tabu search based heuristic to solve the 1-PDPTW. First, an insertion heuristic is used to create a route respecting capacity and precedence constraints. Two tabu search methods are then used to create an initial minimal solution. This approach produced solutions for instances of sizes between 10-40 customers in a reasonable amount of time. When the instance size exceeded 60 customers, however, processing time was greatly increased (over one hour).

For further information on approximation algorithms used to solve the 1-PDPTW, Hosney and Mumford (2010)[5] provides detailed research on various approximation approaches, as well as providing experimental work and attributing with there own proposed heuristic.

3 Problem Formulation

In the 1-PDPTW we are given a single vehicle and a set of transportation requests. As stated previously, each transportation request consists of a pickup location, a delivery location and a load to be transported. In order for a vehicle to service a location it must arrive at the location within its respective time window. If the vehicle arrives before the time window it must wait before it can service the location.

The goal of the 1-PDPTW is to find a valid route that serves all requests. A route is valid if:

- The route starts at the origin depot and ends at the destination depot.
- Each location is visited and serviced exactly once.
- For each request, the pickup location is visited before its corresponding delivery location. This is named the precedence constraint.
- The vehicles capacity is never exceeded at any stop in the route.
- Each location is served within its time window.

Travelling from one location to another has two cost values: a time cost and a distance cost. The objective is to minimize the sum of the total time used and the total distance travelled. [4]

Mathematical Formulation

The mathematical formulation and notation for 1-PDPTW is based on X. Wang's [1] formulation of the PDPTW.

Lets define the set of transportation requests as follows:

[4] In the literature and in practice, the objective function can vary depending on the cost values assigned. Additionally it is often such in experimental tests that distance is equal to time.

- $R = \{1, \ldots, n\}$ is the set of requests.
- $P = \{1, \ldots, n\}$ is the set of pickup locations.
- $D = \{n+1, \ldots, 2n\}$ is the set of delivery locations.
- o and o' is the starting and ending depots. In other literature (Furtado [4]), the origin and destination depot are also denoted as 0 and $2n+1$, respectively.

For each request $r \in R$ we specify its corresponding pickup and delivery location as $(u, u+n)$ where $r = u \in P$ and $u+n \in D$. In the literature (X. Wang [1]), this pair is called an origin-destination pair (OD-pair). The load to be transported is denoted as ℓ_r for each request r. This specifies the request's pickup locations load is $\ell_u = \ell_r$, and the delivery locations load is $\ell_{u+n} = -\ell_r$ where $u = r$. At each location $u \in P \cup D$ we denote the time window as $[b_u, e_u]$, $u \in P \cup D$, where b_u and e_u is the earliest and latest time the vehicle can start its service, respectively. It is assumed that both the service time and load for the depots o and o' are defined as 0.

Let us define the problem as a directed graph $G = (V, A)$. The set of nodes $V = P \cup D \cup \{o, o'\}$ represent the locations and the set of arcs $A \subset V \times V$ represent roads connecting the locations. By the constraints given by the problem, certain edges will never be traversed. Let $(u, v) \in V \times V$ be an arc from u to v. We define the complement of A, denoted $\overline{A}$, as

$$\begin{aligned}
\overline{A} = &\{(u,u) | u \in V\} \cup && (1)\\
&\{(u,o) | u \in V\} \cup && (2)\\
&\{(o',v) | v \in V\} \cup && (3)\\
&\{(o,v) | v \in D \cup \{o,o'\}\} \cup && (4)\\
&\{(u,o) | u \in P \cup \{o,o'\}\} \cup && (5)\\
&\{(u+n,u) | u \in P, u+n \in D\} && (6)
\end{aligned}$$

such that $A = V \times V / \overline{A}$. A has no looping arcs , as this has no relevance to the problem (1). By definition of a valid route, each route must start at the starting depot o and end

at the ending depot o', thus, no edges go to the starting depot (2) and no edges go from the ending depot (3). By the precedence constraint, the first location visited after the starting depot must be a pickup location (4) and the location visited before the arriving at the ending depot must be a delivery location (5). Additionally, for each pickup location $u \in P$, the edge $(u+n, u)$, $u+n \in D$ is never traversed as this violates the precedence constraint (6).

Let n be the number of transportation requests. The number of arcs $|A|$ is equal to the sum of the arcs from the starting depot (n arcs), the arcs to the ending depot (n arcs) and the arcs that go between the remaining locations. The arcs that go between the remaining locations equate to a complete directed graph consisting of the pickup and delivery locations with the removal of the arcs that go from a delivery location to its corresponding pickup location ($2n(2n-1) - n$ arcs). Thus, the number of arcs $|A| = n + n + 4n^2 - 2n - n = 4n^2 - n$.

Figure 1 shows an example of a graph representing a 1-PDPTW instance with two transportation requests where $R = \{1, 2\}$,$P = \{1, 2\}$ and $D = \{3, 4\}$.

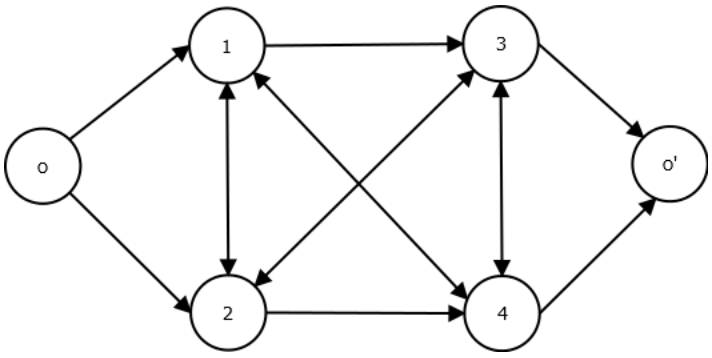


Figure 1: An example of a 1-PDPTW graph with two transportation requests

Each arc $(u, v) \in A$ has two weights: a travel distance d_{uv} and a travel time d'_{uv}. Additionally, the set of arcs satisfy the triangle inequality. Furthermore, we include two cost rates, β and β', for distance and time, respectively. The cost rates can be used for generating different types of minimum cost routes.

In order to solve the problem, three variables are needed. The first variable, x_{uv}, is a binary variable that equals 1 if the vehicle traversed the arc $(u, v) \in A$, and 0 otherwise.

The second variable, $l_u, u \in V$, is the load after the service at node u is finished. The last variable, $t_u, u \in V$, is the time that service is started by the vehicle at node u.

MIP Model

The 1-PDPTW can be modelled as the following:

$$\min \sum_{(u,v)\in A} \beta d_{uv} x_{uv} + \beta'(t_{o'} - b_o) \tag{1}$$

subject to:

$$\sum_{\substack{u\in P\cup D\cup\{o\}\\(u,v)\in A}} x_{uv} = 1 \qquad \forall v \in P \cup D \tag{2}$$

$$\sum_{\substack{v\in P\cup D\cup\{o'\}\\(u,v)\in A}} x_{uv} = 1 \qquad \forall u \in P \cup D \tag{3}$$

$$\sum_{v\in P} x_{o,v} = 1 \tag{4}$$

$$\sum_{u\in D} x_{u,o'} = 1 \tag{5}$$

$$x_{uv}(t_u + s_u + d'_{uv} - t_v) \leq 0 \qquad \forall (u,v) \in A \tag{6}$$

$$b_u \leq t_u \leq e_u \qquad \forall u \in V \tag{7}$$

$$t_u + s_u + d'_{u,n+u} \leq t_{n+u} \qquad \forall u \in P \tag{8}$$

$$x_{uv}(l_u + \ell_v - l_v) = 0 \qquad \forall (u,v) \in A \tag{9}$$

$$\ell_u \leq l_u \leq Q \qquad \forall u \in P \tag{10}$$

$$0 \leq l_{n+u} \leq Q - \ell_u \qquad \forall u \in P \tag{11}$$

$$l_o = l_{o'} = 0 \tag{12}$$

$$t_o = b_o \tag{13}$$

$$t_u \geq 0 \qquad \forall u \in V \tag{14}$$

$$l_u \geq 0 \qquad \forall u \in V \tag{15}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{16}$$

(1) The objective function minimizes a weighted sum of total cost of all traversed edges

and the total time it takes from the origin depot to the destination depot.

(2)-(3) These constraints guarantee that each location, besides the depots, is visited exactly once. In particular, constraint (2) specifies that for every node v, the sum over all arcs x_{uv} going into node v, is equal to 1. Since x_{uv} is a binary variable, then in order for the sum over all the arcs into v to be equal to 1, exactly one arc can have an x-variable with value of 1. This represents entering a node only once. Constraint (3) is similar, but instead of entering a node it guarantees that the vehicle leaves every node exactly once.

(4) A route must start at the origin depot and thereafter visit a pickup node, since no delivery node has yet been visited.

(5) A route must end at the destination depot and must have previously been visited by a delivery node.

(6-7) These constraints define the restrictions on travel and arrival time. Constraint (6) states that for any travelled arc $x_{u,v} = 1$, the arrival time at v is the earliest time you can leave u, denoted as $t_u + s_u$, plus the travel time from u to v. The arrival time at t_v must also be within the time window at node v, described in (7). Thus, idle time is allowed, i.e. arrival time at v can be greater then the actual time it takes to travel from u to v. Note that constraint (6) only holds if the set of arcs adhere to the triangle inequality.

(8) Constraint (8) describes the precedence constraint (pickup node is visited before delivery node).

(9)-(11) These constraints describe and bound the vehicles load at any node. Constraint (9) describes how the load changes from a node u to a node v. For any travelled arc $x_{u,v}$, the load l_v at node v is the load at l_u plus the demand ℓ_v at v. Constraints (10) and (11) bound the load at a pickup and delivery node respectively.

(12)-(14) The last constraints set the load for the depots (12) and the start time at the origin depot (13)

Linearising constraints (6) and (9)

It order to linearise the model, constraints (6) and (9) must be changed. This can be done in the following way:

$$(6) : x_{uv}(t_u + s_u + d'_{uv} - t_v) \leq 0 \Rightarrow t_u + s_u + d'_{uv} - t_v \leq (1 - x_{uv})M_1$$

$$(9) : x_{uv}(l_u + \ell_v - l_v) = 0 \Rightarrow l_u + \ell_v - l_v \leq (1 - x_{uv})M_2 \text{ and } l_u + \ell_v - l_v \geq (1 - x_{uv})M_3$$

where M_1,M_2 and M_3 are sufficiently large integers. If $x_{uv} = 1$, then constraint (6) and (9) produce the desired constraints:

$$t_u + s_u + d'_{uv} - t_v \leq 0$$

and

$$l_u + \ell_v - l_v \leq 0 \text{ and } l_u + \ell_v - l_v \geq 0 \Rightarrow l_u + \ell_v - l_v = 0$$

In order to further constrict our model, lower bounds can be found for M_1, M_2 and M_3. This can be done by considering the lower or upper bounds of the variables in the constraints. Constraint (6) contains three variables and two constants. In the case that $x_{uv} = 0$, the constraint evaluates to $t_u + s_u + d'_{uv} - t_v \leq M_1$. Let $M_1 = M'_1 - (d'_{uv} + s_u)$, thus creating the following in-equality :

$$t_u - t_v \leq M'_1$$

The maximum value that t_u - t_v can evaluate to is attained when t_u is at its upper bound and t_v is at its lower bound. The upper and lower bounds for these variables are defined in constraint (7). Thus, $e_u - b_v = M'_1$ which yields the following:

$$M_1 = e_u - b_v - (d'_{uv} + s_u)$$

For constraint (9), there are two variables to consider given that $x_{uv} = 0$. These are the load variable l_u and l_v. Constraint (10) and (11) define the lower and upper bounds for

these variables. To determine M_2 and M_3 in

$$l_u + \ell_v - l_v \leq M_2$$

and

$$l_u + \ell_v - l_v \geq M_3$$

we must determine when $l_u - l_v$ is at its upper and lower bound respectively. The bounds for the load variables are:

$$\begin{cases} l_u = 0 & \text{if } u \in \{o, o'\} \\ \ell_u \leq l_u \leq Q & \text{if } u \in P \\ 0 \leq l_u \leq Q - \ell_{u-n} & \text{if } u \in D \end{cases}$$

Given the type of location u and v are, the values for M_2 and M_3 are the following:

$$M_2 = U(l_u) + \ell_v + L(l_v)$$

$$M_3 = L(l_u) + \ell_v + U(l_v)$$

where $U(\ell_u)$ and $L(\ell_u)$ are the upper and lower bounds on ℓ_u respectively.

4 A Forward Dynamic Programming Approach

Desrosiers et al. (1986) [2] presents a forward dynamic programming approach to solve the single vehicle dial-a-ride problem with time windows for large scale. The objective was to find the route which minimizes the total distance travelled. The idea behind the algorithm is to reduce the number of states generated by removing dominating state. Desrosiers proposes a two dimensional labelling of states (time, cost), which was introduced in thier previous paper [3] to solve the shortest path problem with time windows. This algorithm produced promising results and was able to solve problems with 40 transportation requests (80 nodes). This was due to efficient elimination criteria which eliminated states which were infeasible.

Solution method

Desrosiers et al. solves the problem using a forward dynamic programming method with $1 \leq k \leq 2n$ iterations, where n is the number of transportation requests. The set of nodes are denoted as $\{0, \ldots, 2n+1\}$, where node 0 and $2n+1$ are the origin and destination nodes, respectively, and nodes $\{1, \ldots, n\}$ and $\{n+1, \ldots, 2n\}$ are the pickup and delivery nodes, respectively, such that every delivery node $n+i \in \{n+1, ..2n\}$ corresponds to the pickup node $i \in \{1, ..., n\}$. The vehicle is initially located at the origin node 0. At the first iteration, the states are made up of routes visiting a single pickup node. At each subsequent iteration $2 \leq k \leq 2n$, the states are constructed from the states of the previous iteration and are made up of routes visiting one additional node from among the pickup and delivery nodes. After $2n$ iterations, the vehicle must return to the origin node (destination node) $2n+1$. For each iteration, states are considered for elimination based on feasibility.

Definition of States

Let $S \subseteq \{1, \ldots, 2n\}$ be the non-ordered set of visited nodes at iteration k with cardinality k. If there exists a feasible route starting at origin node 0 that visits all nodes in S and terminates at node $i \in S$, we denote the corresponding *state* as (S, i).

Definition of Labels and Label Elimination

There can be multiple feasible routes that start from the origin node to a node i for each state (S, i). We denote a route that corresponds with the state (S, i) as (S_α, i), where S_α describes the order of visitation for the route. Regardless to the order of which the nodes in S are visited, every route has the same vehicle load at node i denoted $y(S)$. On the other hand, routes differ in terms of distance travelled and arrival time at node i depending on the order of which the nodes were visited. For each route, a cost *label* is defined as $(t(S_\alpha, i), z(S_\alpha, i))$, where $t(S_\alpha, i)$ represents the arrival time at node i and $z(S_\alpha, i)$ represents the distance travelled to reach node i.

Not all labels (i.e routes) are considered for a given state. These labels are eliminated if they cannot be part of the *minimal cost route* from node 0 to node $2n + 1$. We denote the set of minimal cost routes as $\Omega(S, i)$ and denote the minimal cost labels as:

$$H(S, i) = \{(t(S_\alpha, i), z(S_\alpha, i)) | S_\alpha \in \Omega(S, i)\}$$

Labels and routes are eliminated using the following partial relation in Lemma 1.

Lemma 1 *A route (S_α, i) and its associated label $(t(S_\alpha, i), z(S_\alpha, i))$ is eliminated from the set of minimal cost routes and labels if there exists a route with both a lesser or equal time and a lesser or equal distance.*

Proof. We show that a label cannot be part of the minimal cost route from 0 to $2n + 1$ using the following optimality principle:
Given a state (S, i) at iteration k , let $(R, j), R = S \cup \{j\}$ be the subsequent state at iteration $k + 1$ and $(t(R_\alpha, j), z(R_\alpha, j))$ be the label associated with a route from 0 to j visiting the nodes in R. If R_α is a minimum cost route from among all routes (R'_α, j) arriving at node j at times $t(R_{\alpha'}, j) \leq t(R_\alpha, j)$ and $z(R'_\alpha, j) \leq z(R_\alpha, j)$ and arc (i, j) is the last in this route, then the sub route from 0 to i is a minimal cost route from among all routes ending in i. The corresponding costs for the sub routes are $t(R_{\alpha'}, j) - s_i - d'_{i,j} \leq t(R_\alpha, j) - s_i - d'_{i,j}$ and $z(R_{\alpha'}, j) - d_{i,j} \leq z(R_\alpha, j) - d_{i,j}$. Thus the label $(t(R_\alpha, j) - s_i - d'_{i,j}, z(R_\alpha, j) - d_{i,j})$ is not part of the minimal cost labels from 0 to i.

Forward Recursion

At the first iteration, the set of states are:

$$\{(\{j\}, j) | j \in \{1, \ldots, n\}\}$$

For each of the states, the load and label are easily calculated:

$$y(\{j\}, j) = \ell_0 + \ell_j \quad j \in \{1, \ldots, n\}$$

$$H(\{j\}, j) = \{(\max[b_0 + d'_{0,j}, b_j], d_{0,j})\} \quad j \in \{1, \ldots, n\}$$

These states correspond to visiting a single pickup node from the origin depot. Each states load consists of the initial load at the origin node plus the load at the pickup node. As there is only one possible route for each state, there is only one label associated with the state. The time cost is calculated as the maximum of the sum of the initial start time and the time from the origin node to the pickup node, and the earliest arrival time at the pickup node. The distance cost is set to the distance from the origin to the destination.

At each subsequent iteration $2 \leq k \leq 2n$, new states are created by adding a single node to the total visited nodes at the preceding iteration. Let $(S \cup \{j\}, j), j \in \overline{S}$, be such a state. The load for this state is found by:

$$y(S \cup \{j\}) = y(S) + \ell_j$$

The set of labels $H(S \cup \{j\}, j)$ consist of the labels of all states (S, i) used to construct the state $(S \cup \{j\}, j)$ when node j is added to the state (S, i). These time and distance costs are adjusted by the following function:

$$f_{ij}(t(S_\alpha, i), z(S_\alpha, i)) = \begin{cases} (\max[t(S_\alpha, i) + s_i + d'_{i,j}, b_j], z(S_\alpha, i) + d_{i,j}) & \text{if } t(S_\alpha, i) + s_i + d'_{i,j} \leq e_j \\ \varnothing & \text{otherwise} \end{cases}$$

The function creates a new label iff starting at node i, it is possible to visit node j before the latest arrival time for the time window at node j. Using this function, for each state (S, i) we have the following set of adjusted labels:

$$F_{ij}(H(S,i)) = \bigcup_{S_\alpha \in \Omega(S,i)} f(ij)(t(S_\alpha, i), z(S_\alpha, i))$$

The new set of labels for the state $(S \cup \{j\}, j)$ can now be created :

$$H^*(S \cup \{j\}, j) = \bigcup F_{ij}(H(S,i))$$

The * symbol represents the list of new labels remaining after the elimination of labels according to Lemma 1 to create the set of minimal routes.

State Elimination Criteria

It is useful to limit the number of states created when using dynamic programming. States are eliminated based on the set S and the state (S, i).[5] A state is feasible iff it is both *ante-feasible* and *post-feasible*. The state (S, i) is ante-feasible if there exists a route which visits the nodes in S and terminates in i which abides by the constraints of the 1-PDPTW (vehicle capacity, precedence and time windows). The state is post-feasible if there exists a route starting at i that visits all nodes in $\overline{S} = V \backslash S$ and abides by the same constraints. All states which are not ante-feasible are eliminated by the algorithm, yet not all states that are post-feasible are eliminated as some post-feasible states would be more costly to identify than the savings that could be achieved.

[5] Desrosier et al. [2] mentions a third elimination bases if multiple nodes are in the same geographical location. We choose to ignore this elimination as it is highly instance dependent.

Elimination criteria based on the set `S`

Let S be the set at iteration $k-1$ where we wish to add a node j to create the state $(S \cup \{j\}, j)$ at iteration k. A state is eliminated if one the criteria is not respected. The following four criteria are completely independent of the terminal node i.

(1) : Node j must not be previously visited.

(2) : If node j is a delivery node, then $j-n$ must be previously visited ($j-n \in S$).

(3) : If node j is a pickup node, then the vehicle capacity constraint must be respected:

$$y(S) + \ell_j < Q$$

(4) : The earliest time at which node j can be visited is $t_j = b_j$. Using this property, we can eliminate states where time window constraints are not respected :

$$b_j + s_j + d'_{jv} \leq e_v \quad \forall v \in \overline{S \cup \{j\}}$$

where v is an unvisited node. If any node cannot be visited after j, then the state is eliminated.

Criteria **(1)**, **(2)** and **(3)** are ante-feasible criteria, while state **(4)** is a post-feasible criteria. Note that in criteria **(4)**, checking every unvisited node for each state is very costly. To overcome this, let the nodes be pre-ordered in decreasing order of their latest arrival time e_i. It is reasonable to presume that this ordering will be relatively similar to the optimal solution. Desrosiers proposes to carry out criteria **(4)** on the unvisited nodes ranked from $k-2$ to $k+4$ as these are the most probable to be visited. This can be altered at the potential cost of computational time.

Elimination criteria based on the state `(S,i)`

Let (S, i) be a state at iteration $k-1$, let $(S \cup \{j\}, j)$ after adding j, and let t_i be the earliest arrival time at node i given by the first label in $H(S, i)$. The following four criteria are

based on the time at j, the time after visiting a subsequent node, the time after visiting two subsequent pickup nodes, and the time after visiting two subsequent delivery nodes, respectively.

(5): The time constraint must be respected: $t_i + s_i + d'_{\mathtt{ij}} \leq e_j$

(6): Suppose that node j is visited at time $t_j = \max\{b_j, t_i + s_i + d'_{ij}\}$, it must be possible to visit each unvisited node $\nu \in \overline{S \cup \{j\}}$ while respecting the time constraints: $t_j + s_j + d'_{j\nu} \leq e_\nu$, for all $\nu \in n\overline{S \cup \{j\}}$. This criterion tightens criterion **(4)**.

(7): Suppose that node j is visited at time $t_j = \max\{b_j, t_i + s_i + d'_{ij}\}$, for every pair of unvisited delivery nodes $n + \nu_1$ and $n + \nu_2$ whose corresponding pickup nodes ν_1 and ν_2 have been previously visited while respecting the time constraints. With two visiting permutations for $n + \nu_1$ and $n + \nu_2$, the following must be satisfied:

$$\max\{b_{n+\nu_1}, t_j + s_j + d'_{j,n+\nu_1}\} + s_{n+\nu_1} + d'_{n+\nu_1,n+\nu_2} \leq e_{n+\nu_2}$$

or

$$\max\{b_{n+\nu_2}, t_j + s_j + d'_{j,n+\nu_2}\} + s_{n+\nu_2} + d'_{n+\nu_2,n+\nu_1} \leq e_{n+\nu_1}$$

with $\nu_1, \nu_2 \in S \cup \{j\}$ and $n + \nu_1, n + \nu_2 \in \overline{S \cup \{j\}}$.

(8): Suppose that node j is visited at time $t_j = \max\{b_j, t_i + s_i + d'_{ij}\}$, it must be possible to visit all pairs of unvisited pickup node ν_1 and ν_2 while respecting the time constraints. With two visiting permutations for $n + \nu_1$ and $n + \nu_2$, we must satisfy:

$$\max\{b_{\nu_1}, t_j + s_j + d'_{j,\nu_1}\} + s_{\nu_1} + d'_{\nu_1,\nu_2} \leq e_{\nu_2}$$

or

$$\max\{b_{\nu_2}, t_j + s_j + d'_{j,\nu_2}\} + s_{\nu_2} + d'_{\nu_2,\nu_1} \leq e_{\nu_1}$$

with $\nu_1, \nu_2 \in \{1, \ldots, n\} \cap \overline{S \cup \{j\}}$.

Criteria **(6)**, **(7)** and **(8)** checks if a state is post-feasible while criteria **(5)** checks if a state is ante-feasible. Criteria **(6)** is checked in the same way as criteria **(4)**. Criteria **7** and **(8)** are only check for one pair of delivery and pickup nodes, respectively.

Previous Results

The tests done by Desrosier showed promising results. They investigated 93 problems with up to 40 requests, where the problems were derived from actual routes from cities in Canada. Almost all of the tests done with a problem size less than 25 requests were solved in less than two seconds, and was able to easily solve problems with 40 requests. The relationship between computation time and problem size was almost linear, with the exception of a few "hard" problems. For a more detailed explanation of the test results and algorithm, see [2].

5 Implementation Details

In this section, we will give an overview of certain implementation details used in implementing the FDP algorithm and the MIP model. Additionally, we discuss the slight alterations to the instance data used in the experimental analysis.

Data

The Li & Lim benchmark [4] provides a vast amount of PDPTW instances ranging from 100 to 1000 locations. An instance is represented as a text file, where the first line contains three integers representing the number of available vehicles, the max capacity for each vehicle, and the speed at which a vehicle travels, respectively. We choose to ignore the speed as it is irrelevant to our problem. The following $n + 1$ lines, n is the number of locations, describe the necessary information about the depot and each location. Since only one depot is given, we set the starting and ending depot to be in the same location. It should be noted that no time or distance costs between locations were given. In both implementations, the distance cost between locations was set to the straight-line distance between the locations position (given in xy-coordinates). We also set the time cost between locations to be equivalent to their respective distance cost. This requires no alteration to our algorithm or problem, as the triangle inequality still holds.

Using Gurobi's Java API

The Gurobi Optimizer provides extensive APIs for multiple languages and frameworks. For coherence with the FDP implementation, we chose to implement the model found in Section 3 using the Java API. For a more detailed explanation of how to use Gurobi Optimizer and it's tools, see the Gurobi documentation for the Java API.[6] Although Gurobi provides tools to import models written in many types of modelling languages, we have chosen to implement our model purely using the Java API. The API's modelling syntax is very intuitive and allowed us to use previously written code to import instance data. The

[6]Gurobi documentaion is available at `http://www.gurobi.com/documentation/`

implemented model is shown in Listing 1.

Implementing the FDP in Java

We have chosen a similar implementation method as described in [2]. The data structure is split into three levels. The first level retains information of the feasible set S, current load, and a reference to the terminal nodes. The second level contains the terminal nodes, which form the states (S, i). Each terminal node has a reference to the set of labels $H(S, i)$, which are stored in the third level. Each label consists of the time cost and distance cost for that particular route.

As new states are generated at each iteration, the DP table can become quite large. For large instances, certain implementation techniques may be needed to limit the memory requirement. Additionally, at each iteration, only the previous and current set of states are required. At the end of iteration k, only the labels from iteration $k-1$ which were used to generate the labels of k need to be stored in order for backtracking. For theoretical purposes, we choose to store the entire DP table.

Certain elimination criteria are checked at different levels in the data structure. Each state holds information on which nodes it can visit. Let $N = \{1, ..., n\}$ be the possible nodes that can be visited at the initial state (starting depot). This set comprises of all pickup nodes. When visiting a node $j \in N$, j is removed from N. If $j \in \{1, ..., n\}$ (a pickup node), we add it's corresponding delivery node $j + n$ to N. By using this process, criteria **(1)** and **(2)** are respected. In the first level, criteria **(3)** and **(4)** are checked when a node j is considered to be added to a state. Criteria **(6)**, **(7)** and **(8)** are checked in the second level after the state $(S \cup \{j\}, j)$ is created. Finally, criteria **(5)** is checked before creating each label. Once all feasible labels are created, all labels that are not part of the set of minimum cost labels are removed.

6 Experimental Analysis

In this section we will describe the experiments done on both the MIP model using Gurobi and Desrosier's[2] FDP algorithm . The data to be solved was retrieved from the Li & Lim benchmark[4] for PDPTW problems. Each instance of a PDPTW problem used has been previously solved. These solutions were used to extract multiple SPDPTW problems from a single instance. This generated ca 4500 SWPDPTW instances varying from 3 - 121 total nodes (1 - 60 requests). The majority of instances range from a size of 11 to 21 nodes. The graph in figure 2 shows the number of instances per instance size.[7] Both algorithms were run on a computer with an Intel Core i-7-4700HQ processor with a speed of 2.4GHz with and 16GB of memory.

The goal of this experiment was to compare the number of solutions found and how running time changes with respect to the instance size. Additionally, as our major focus is solving relatively small instances (20 - 30 nodes) in a short period of time, we will restrict all running time to 10 seconds. If a solution is not found with this time limit, it is terminated. A special case is given to the MIP model, where a feasible solution might be found.

MIP Model Results

The MIP model is implemented in Java using the Gurobi API and solved with Gurobi. Gurobi allows to alter the solution strategy of the solver with a given parameter. There are four strategies available: a balanced strategy between finding feasible solutions and proving optimality (MIP 0), priority on finding feasible solutions (MIP 1), priority on proving optimality (MIP 2) and priority on finding better bounds for the objective function (MIP 3). By choosing the correct strategy, one can potentially improve the overall runtime. Our first experiment is comparing the four strategies in order to decide on which to use when comparing this solution method with the FDP algorithm.

[7] All figures presented in this section can be found in the Figures section.

Strategy Analysis

To compare these strategies, we will first observe how each individual strategy preforms. Afterwards, we will compare the strategies based on the amount of instances solved and the runtime for each instance.

Figures 3 and 4 show a comparison of the results produced from each strategy. Figure 3 shows the optimal solutions found while figure 4 shows the total solutions, either optimal or feasible, that were found. From these results, we observe that all strategies, except for MIP 3, managed to find optimal solutions for all instances with a size less then 15. The amount of optimal solutions found for instance sizes greater then 15 starts to decrease gradually. For instance sizes greater than 63, none of the strategies produced optimal or feasible solutions. MIP 3 generally produced the worst results, only producing equal results for 11 out of the 31 instance sizes. MIP 0 produced the best overall results, with either equal or more optimal solutions for 23 out of the 31 instance sizes.

This strategy also produced on average more solutions (21 out of 31). It can also be observed that MIP 1 provides consistently more solutions than MIP 2. The total overall difference of both optimal solutions found and total solutions found between all strategies is on average around $\approx$ 7%. There are also an average of $\approx$ 9% feasible solutions found for each instance size. A more detailed visualization of the average amount of optimal and total solutions can be found in figure 5 . Although this difference is significant, it is not yet clear which strategy to choose. When the instance size is greater than 35 the best performing strategy varies mainly between MIP 0 and MIP 1.

To confirm our findings from the results analysis, we will now analyse the runtime results for each instance. Figure 6 shows comparisons of runtime results for each strategy. The charts visualize how many more instances a strategy performed better than the compared strategy. Charts (6a), (6e) and (6f) show MIP 3 compared to MIP 0, MIP 1 and MIP 2 respectively. This confirms our previous assumption that MIP 3 is a less adequate strategy to solve our problem. The results from chart (6d) show that MIP 1 and MIP 2 perform similarly, with MIP 1 performing slightly better (ca. 5% better). Charts (6a), (6b), and (6c) show that the balanced strategy, MIP 0, performs significantly better than the other strategies. This confirms our previous assumption concerning MIP 0.

From both the results analysis and the runtime analysis, we can conclude that a balanced strategy is better than optimizing a single aspect (feasibility, optimality or bounds). Thus, we will be using this strategy when comparing the MIP model with Gurobi and the FDP implementation. Although MIP 0 seemed a superior strategy, further analysis could be done in determining which types of instances each strategy performs better. It would also be interesting to see how each strategy performs given a larger time limit.

FDP Algorithm Results

In this section we will analyse the results produced from the FDP. Figure 7 shows the result overview for the FDP. The FDP implementation managed to solve all instances with instance size less then 25. After this point, the number of solutions found decreases in a similar manner as with the MIP implementation. The largest solved instance had a size of 73 (36 requests). For our goal sizes (20-30 nodes), the FDP implementation managed to solve the majority of the instances, with the largest instance size of 31 being the lowest percent solved (60%).

Figure 8 shows the average depth for each instance size compared to the route size. When the instance size is greater than 25, the difference between average depth reached and the actual route size increases.

Comparison

To compare these two implementations, we will first compare the amount of solutions produced and then compare the runtime for each instance as we did in section 1. Figure 9 shows a comparison of solutions found for instances sizes of 19 to 37 (9a) and instance sizes of 37-73 (9b). For all instances, except for instance sizes of 31, the FDP implementation provided more solutions than the MIP implementation. On average, the FDP provided ca 10% more solutions.

Our runtime analysis in Figure 10 also shows an overwhelming dominance of the FDP implementation. Out of the 4695 instances tested, both implementations either did not

find a solution or had the same runtime for 2334 instances (50%). The FDP managed to find a solution faster for 2279 instances (48%), while the MIP implementation only managed to solve 82 instances (2%) faster.

Observing our goal region (20-30 nodes), we can see that the FDP finds more optimal solutions than the MIP implementation. Not only that, but the FDP finds almost all of these solutions faster as well.

Experimental Conclusions

From the comparisons shown in the previous section, it leads us to believe that the FDP implementation is superior to the MIP implementation given a time limit of 10 seconds. An interesting finding to be noted is that when running both implementations, the FDP needed far more memory than the MIP-model. The MIP model on the other hand used far more processing power than the FDP. This makes sense, since the FDP algorithm needs to save previous states in a table, while Gurobi (and other MIP-solvers) utilize far more processing power to manipulate constraints. From this observation, one can conclude that managing and optimizing memory can lead to better results for the FDP and increased processing capabilities can lean to better results for the MIP-model.

7 Conclusion

In this thesis we have considered two exact methods to solve the small scale 1-PDPTW. Desrosier's et al.[2] FDP algorithm proved to be more efficient at solving the problem given the time constraint on the runtime. Both algorithms managed to find an optimal solution for over 50% of all instance sizes with less then 30 stops. The FDP algorithm solved all instances with less then 25 stops while the MIP-solver began to drop off in the number of optimal solutions for instance sizes of less then 15. The FDP algorithm provided overall more solutions as well as finding the solutions faster.

Based on the experimental results, the FDP algorithm proved to be the most promising choice for small-scale instances. It should be noted that FDP algorithm utilizes far more memory, and thus must be taken into consideration when implementing this algorithm. With larger instance sizes, this can become an issue and can be thus advantageous to solve the MIP model as MIP-solver mainly are dependent on processing power.

Future research should be directed towards the application of exact methods as a sub-routine to solve the PDPTW. As implementation is key in the FDP algorithm, designing techniques should be further researched in order to decrease processing time. Finally, as we were able to solve instances with up to 73 locations, it would be interesting to research methods of preprocessing instances to determine their complexity. This would provide a powerful tool to determine which algorithm to choose based on the difficulty of a particular instance.

References

[1] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Manage. Sci.*, 6(1):80–91, oct 1959.

[2] Jacques Desrosiers, Yvan Dumas, and François Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325, 1986.

[3] Jacques Desrosiers, Paul Pelletier, and François Soumis. Plus court chemin avec contraintes d'horaires. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 17(4):357–377, 1983.

[4] H.Li and A.Lim. Pickup and delivery problem with time windows instances. `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/m`.

[5] Manar I. Hosny and Christine L. Mumford. The single vehicle pickup and delivery problem with time windows: intelligent operators for heuristic and metaheuristic algorithms. *Journal of Heuristics*, 16(3):417–439, 2010.

[6] Wan-Rong Jih and J Yung-Jen Hsu. Dynamic vehicle routing using hybrid genetic algorithms. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 453–458. IEEE, 1999.

[7] Wan-rong Jih, Cheng-Yen Kao, and Jane Yung-jen Hsu. Using family competition genetic algorithm in pickup and delivery problem with time window constraints. In *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, pages 496–501. IEEE, 2002.

[8] Antoine Landrieu, Yazid Mati, and Zdenek Binder. A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. *Journal of Intelligent Manufacturing*, 12(5-6):497–508, 2001.

[9] Harilaos N Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[10] Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

[11] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2002.

[12] LJJ Van der Bruggen, Jan Karel Lenstra, and PC Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, 27(3):298–311, 1993.

[13] Xin Wang. Vehicle routing. In *Operational Transportation Planning of Modern Freight Forwarding Companies: Vehicle Routing under Consideration of Subcontracting and Request Exchange*, pages 7–23. Springer Fachmedien Wiesbaden, Wiesbaden, 2015.

Listings

Listing 1: MIP Model Snippet

```
/**
 * This programs is an implementation of a MIP model for a 1-PDPTW instance
 * using Gurobi Optimizer as a solver.
 * For Gurobi documentation see : {@link http://www.gurobi.com/documentation/6.5/refman/java_api_overview.html}
 * @author Yacine Lakel
 * @date 15.08.2016
 *
 */

public class Mip {


  public Mip(String file){
    DATA.createData(file);
  }


  /**
   * Solves an instance of the 1-PDPTW
   * @param MIP_FOCUS
   * an integer representing the strategy the user wants the mip solver to use.
   * Min value: 0
   * Max value: 3
   * See Gurobi Java API documentation for more detail.
   * @param TIME_LIMIT
   * a double the represents the time limit for the solver.
   * Min Value: 1.0
   * @return a String representing the solution found, if any.
```

```
29     */
    public String solve(int MIP_FOCUS, double TIME_LIMIT){
31
      assert(MIP_FOCUS >= 0 && MIP_FOCUS <= 3);
33    assert(TIME_LIMIT >= 1.0);

35    String solution ="";
      try {
37
        // Create Model
39      GRBEnv env   = new GRBEnv();
        env.set(GRB.IntParam.MIPFocus,MIP_FOCUS);
41      env.set(GRB.DoubleParam.TimeLimit, TIME_LIMIT);;
        env.set(GRB.IntParam.OutputFlag,0);
43      GRBModel  model = new GRBModel(env);

45      // Create Flow Variables (x_ij)
        GRBVar[][] flow_vars = new GRBVar[DATA.getNumberOfLocations
           ()][DATA.getNumberOfLocations()];
47      createFlowVariables(model,flow_vars);

49      // Create Load Variables (l_ij)
        GRBVar[] load_vars = new GRBVar[DATA.getNumberOfLocations()
           ];
51      createLoadVariables(model,load_vars);


53
        // Create Arrival Time Variables (t_ij)
55      GRBVar[] time_vars = new GRBVar[DATA.getNumberOfLocations()
           ];
        createTimeVariables(model,time_vars);
57
        // Integrate Variables
59      model.update();
```

```
61      // Create Objective expression (sum { x_ij * d_ij + e_o')
        GRBLinExpr expr = new GRBLinExpr();
63            // Sum all edges travelled (x_ij * d_ij)
        for(int i = 0; i < DATA.getNumberOfLocations(); i++){
65        for(int j = 0; j < DATA.getNumberOfLocations(); j++){
            if(flow_vars[i][j] != null)
67            expr.addTerm(DATA.getDist(i, j), flow_vars[i][j]);
          }
69      }
            // Add total time used (e_o')
71      expr.addTerm(1, time_vars[DATA.getNumberOfLocations()-1]);
        model.setObjective(expr,GRB.MINIMIZE);
73


75      // ----- CONSTRAINTS --------

77      // C0: STARTING CONSTRAINT
        // A route must start from the origin depot and visit a
           pickup node
79
        expr = new GRBLinExpr();
81      for(int i = 1; i <= DATA.getNumberOfRequests(); i++){
          expr.addTerm(1, flow_vars[0][i]);
83      }
        model.addConstr(expr, GRB.EQUAL, 1, "C0");
85
        // C1: ENDING CONSTRAINT
87      // A route must end at a delivery node and then visit the
           destination node

89      expr = new GRBLinExpr();
        for(int i = DATA.getNumberOfRequests() ; i <= 2*DATA.
           getNumberOfRequests(); i++){
```

```
91          expr.addTerm(1, flow_vars[i][DATA.getNumberOfLocations()
               -1]);
          }
93        model.addConstr(expr, GRB.EQUAL, 1, "C1");

95        // C2_(E/L)i: ENTERED AND LEFT ONLY ONCE CONSTRAINT
          // Every node is visited exactly once
97
          for(int u = 1; u < DATA.getNumberOfLocations()-1; u++){
99          expr = new GRBLinExpr();
            for(int i = 0; i < DATA.getNumberOfLocations()-1; i++){
101           if(flow_vars[i][u] != null){
                expr.addTerm(1, flow_vars[i][u]);
103           }
            }
105         model.addConstr(expr, GRB.EQUAL, 1, "C2_L"+u);
            expr = new GRBLinExpr();
107         for(int j = 1; j < DATA.getNumberOfLocations(); j++){
              if(flow_vars[u][j] != null){
109             expr.addTerm(1, flow_vars[u][j]);
              }
111         }
            model.addConstr(expr, GRB.EQUAL, 1, "C2_E"+u);
113       }

115       // C3_i: TIME PRECEDENSE CONSTRAINT
          // a pickup node must be visited before its corresponding
             delivery node
117

119       for(int i = 1; i <= DATA.getNumberOfRequests();i++){
            expr = new GRBLinExpr();
121         expr.addTerm(1, time_vars[i]);
            expr.addConstant(DATA.getTime(i, i+DATA.
```

```
            getNumberOfRequests()) + DATA.getServiceTime(i));
123       model.addConstr(expr, GRB.LESS_EQUAL, time_vars[i+DATA.
            getNumberOfRequests()], "C3"+i);
        }
125

127     // C4_ij: LOAD FLOW
        // Constraints that represents the load flow
129     GRBLinExpr expr1, expr2, expr3;
        for(int i = 0; i < DATA.getNumberOfLocations()-1; i++){
131       for(int j = 1; j < DATA.getNumberOfLocations(); j++){
            if(flow_vars[i][j] != null){
133           expr1 = new GRBLinExpr();
              expr1.addTerm(1, load_vars[i]);
135           expr1.addTerm(-1, load_vars[j]);
              expr1.addConstant(DATA.getLocation(j).getDemand());
137
              expr2 = new GRBLinExpr();
139           expr2.addConstant(-2*DATA.getVehicleCapacity());
              expr2.addTerm(2*DATA.getVehicleCapacity(),flow_vars[i
                ][j]);
141
              expr3 = new GRBLinExpr();
143           expr3.addConstant(DATA.getVehicleCapacity());
              expr3.addTerm(-DATA.getVehicleCapacity(),flow_vars[i
                ][j]);
145
              expr2.add(expr1);
147           expr3.add(expr1);

149           model.addConstr(expr2, GRB.LESS_EQUAL, 0, "C4U_"+i+"_
                "+j);
              model.addConstr(expr3, GRB.GREATER_EQUAL, 0, "C4L_"+i
                +"_"+j);
```

```
151             }
              }
153         }

155         //C5_ij: TIME FLOW
            //Constraints that represents the load flow
157         double bound = DATA.getLatestTime(0);
            for(int i = 0; i < DATA.getNumberOfLocations()-1; i++){
159           for(int j = 1; j < DATA.getNumberOfLocations(); j++){
                if(flow_vars[i][j] != null){
161               expr = new GRBLinExpr();
                  expr1 = new GRBLinExpr();
163               double cost = DATA.getServiceTime(i) + DATA.getTime(i
                     , j);
                  // Left hand side
165               expr.addTerm(1,time_vars[i]);
                  expr.addTerm(-1,time_vars[j]);
167               expr.addConstant(cost);

169               expr1.addConstant(bound);
                  expr1.addTerm(-bound, flow_vars[i][j]);
171
                  model.addConstr(expr, GRB.LESS_EQUAL, expr1, "C5L"+i+
                     ""+j);
173           }

175       }
          }
177


179       // Optimize model
          model.optimize();
181       try{
            String dist = String.format("%.2f", model.get(GRB.
```

```
            DoubleAttr.ObjVal));
183         String runtime = String.format("%.2f",model.get(GRB.
            DoubleAttr.Runtime));
            solution = runtime + " " + getTime(flow_vars) + " " +
            dist;
185       }
          catch(GRBException e){
187         solution = "- - -";
          }
189
          // Dispose of model and environment
191       model.dispose();
          env.dispose();
193
        } catch (GRBException e) {
195       System.out.println("Error code: " + e.getErrorCode() + ". "
              +
              e.getMessage());
197     }
        return solution;
199   }
```

Figures

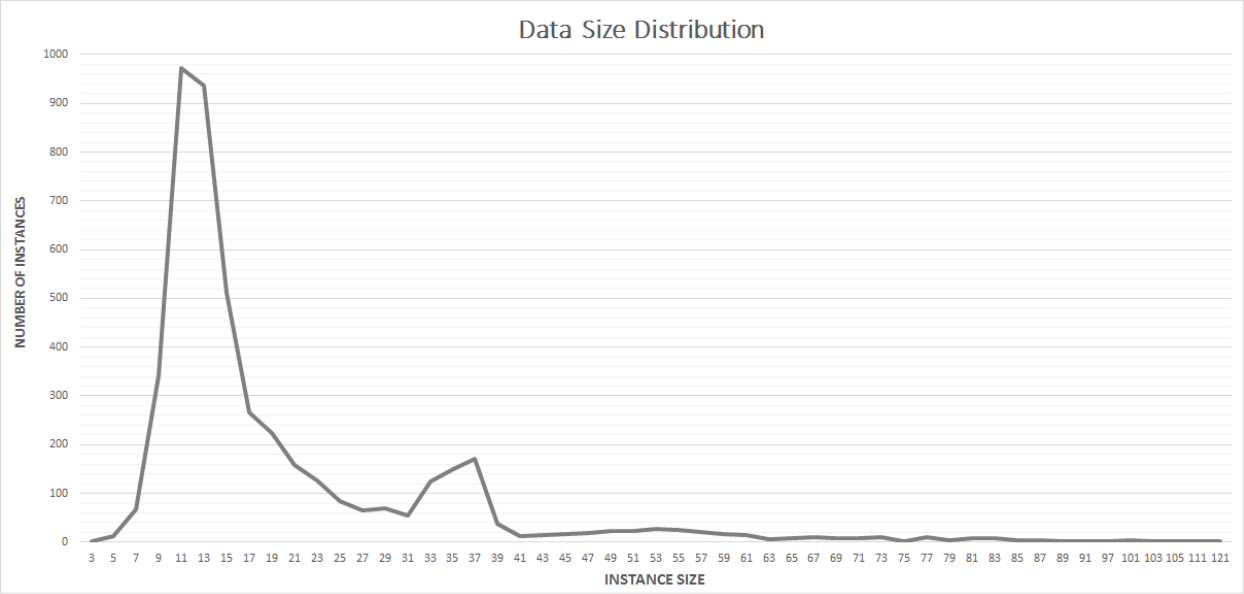
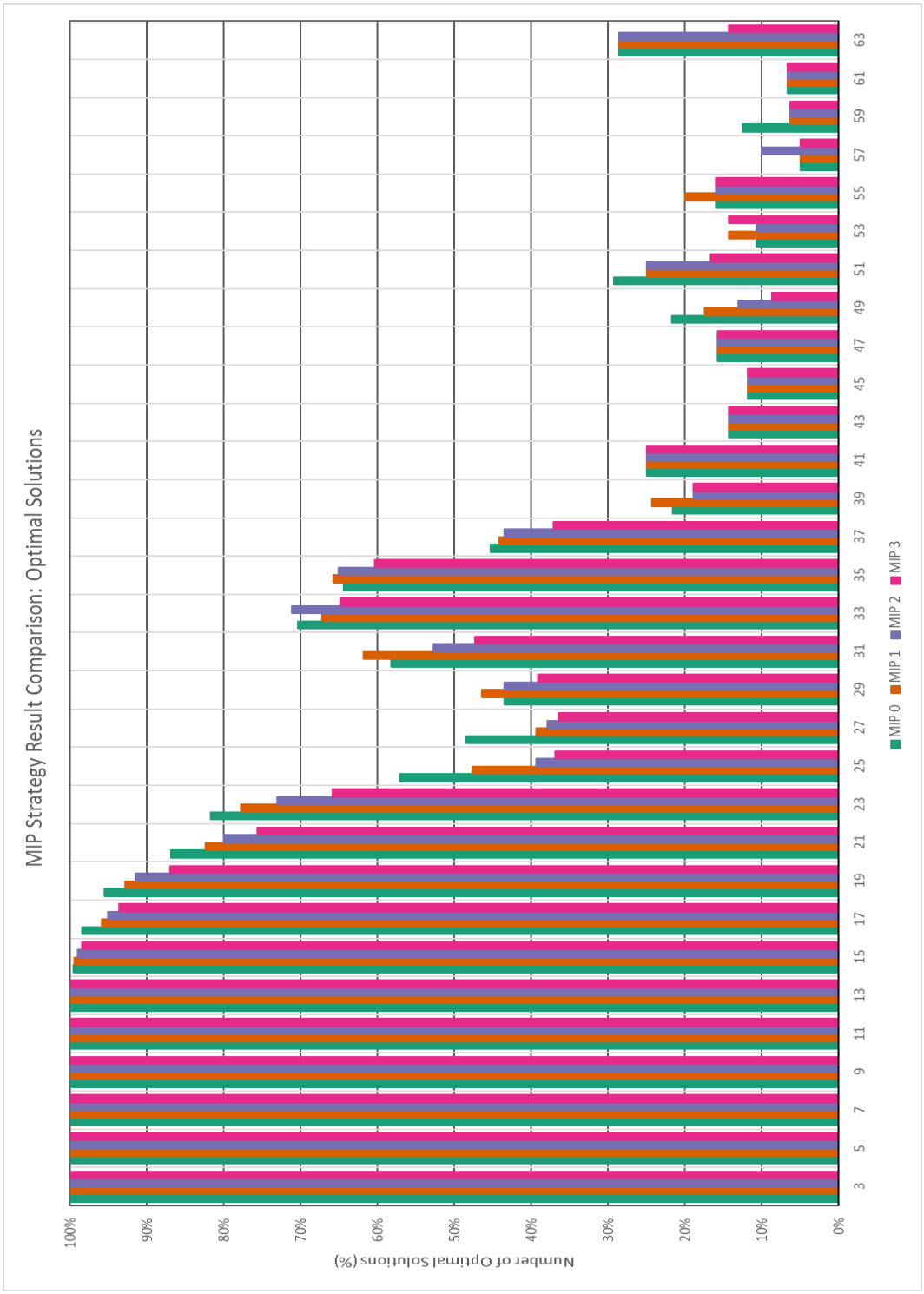
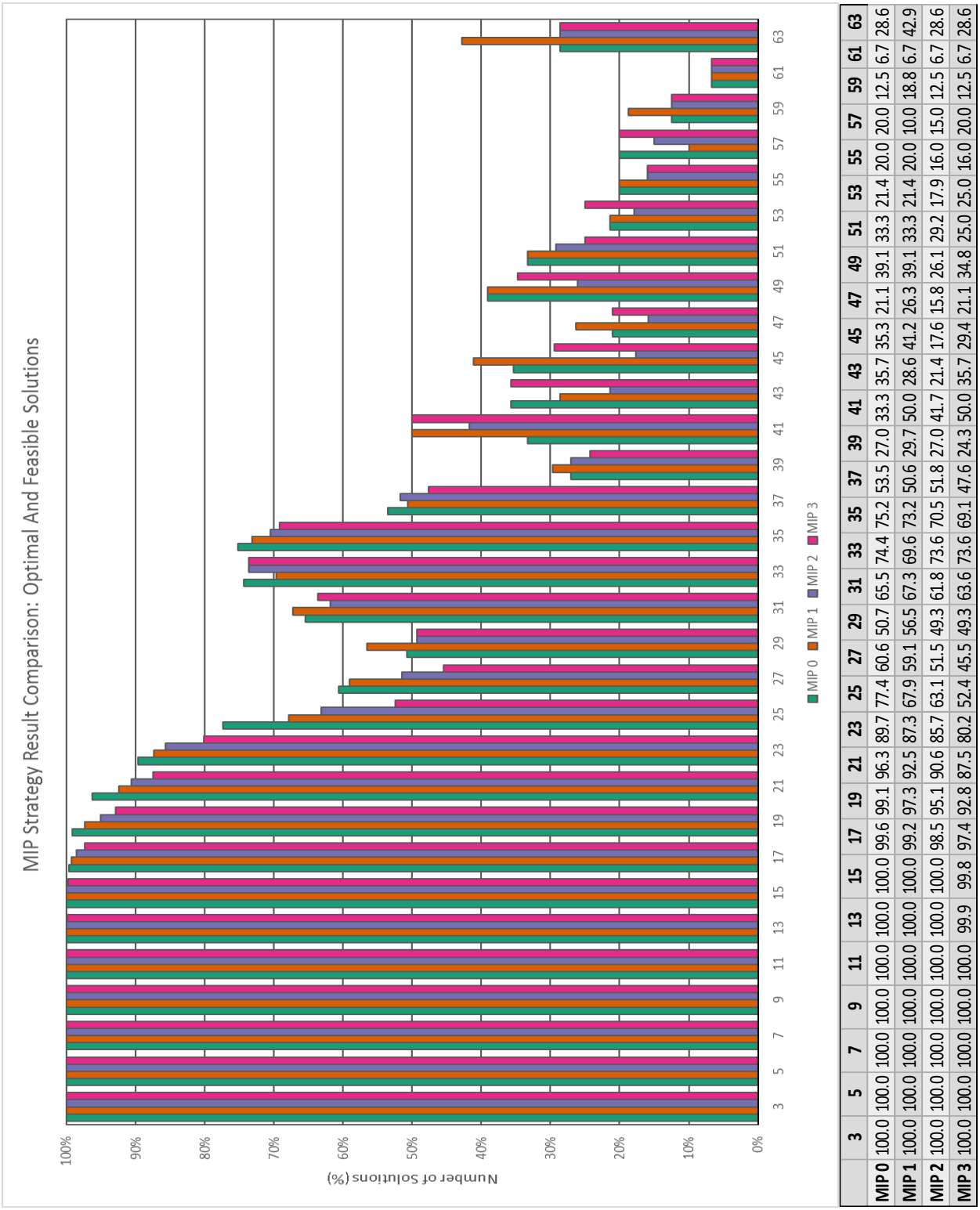


Figure 2: Data distribution over instance size



	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63
MIP 0	100.0	100.0	100.0	100.0	100.0	100.0	99.6	98.5	95.5	86.9	81.7	57.1	48.5	43.5	58.2	70.4	64.4	45.3	21.6	25.0	14.3	11.8	15.8	21.7	29.2	10.7	16.0	5.0	12.5	6.7	28.6
MIP 1	100.0	100.0	100.0	100.0	100.0	100.0	99.4	95.9	92.8	82.4	77.8	47.6	39.4	46.4	61.8	67.2	65.8	44.1	24.3	25.0	14.3	11.8	15.8	17.4	25.0	14.3	20.0	5.0	6.3	6.7	28.6
MIP 2	100.0	100.0	100.0	100.0	100.0	100.0	99.0	95.1	91.5	80.0	73.0	39.3	37.9	43.5	52.7	71.2	65.1	43.5	18.9	25.0	14.3	11.8	15.8	13.0	25.0	10.7	16.0	10.0	6.3	6.7	28.6
MIP 3	100.0	100.0	100.0	100.0	100.0	99.9	98.4	93.6	87.0	75.6	65.9	36.9	36.4	39.1	47.3	64.8	60.4	37.1	18.9	25.0	14.3	11.8	15.8	8.7	16.7	14.3	16.0	5.0	6.3	6.7	14.3

Figure 3: MIP Strategy Comparison of the Optimal Solutions Found



	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63
MIP 0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.6	99.1	96.3	89.7	77.4	60.6	50.7	65.5	74.4	75.2	53.5	27.0	33.3	35.7	35.3	21.1	39.1	33.3	21.4	20.0	20.0	12.5	6.7	28.6
MIP 1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.2	97.3	92.5	87.3	67.9	59.1	56.5	67.3	69.6	73.2	50.6	29.7	50.0	28.6	41.2	26.3	39.1	33.3	21.4	20.0	10.0	18.8	6.7	42.9
MIP 2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	98.5	95.1	90.6	85.7	63.1	51.5	49.3	61.8	73.6	70.5	51.8	27.0	41.7	21.4	17.6	15.8	26.1	29.2	17.9	16.0	15.0	12.5	6.7	28.6
MIP 3	100.0	100.0	100.0	100.0	100.0	99.9	99.8	97.4	92.8	87.5	80.2	52.4	45.5	49.3	63.6	73.6	69.1	47.6	24.3	50.0	35.7	29.4	21.1	34.8	25.0	25.0	16.0	20.0	12.5	6.7	28.6

Figure 4: MIP Strategy Comparison of Optimal and Feasible Solutions Found

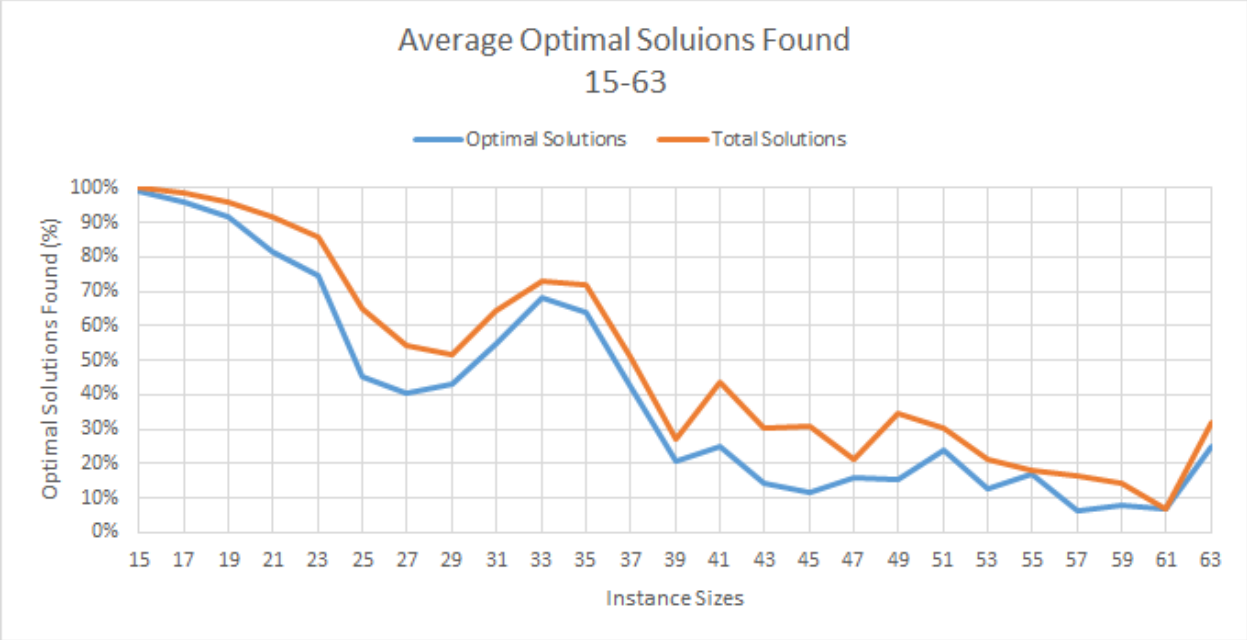
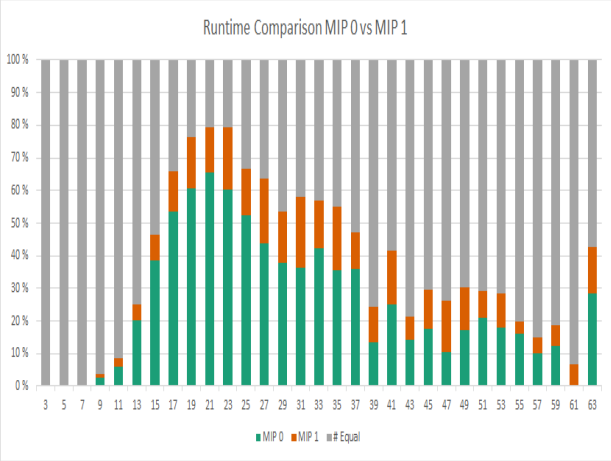
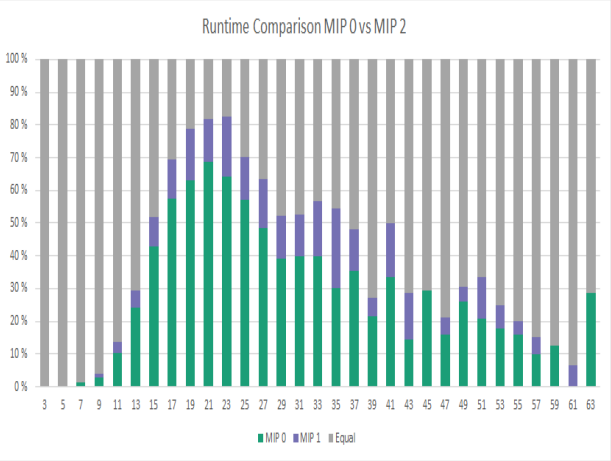


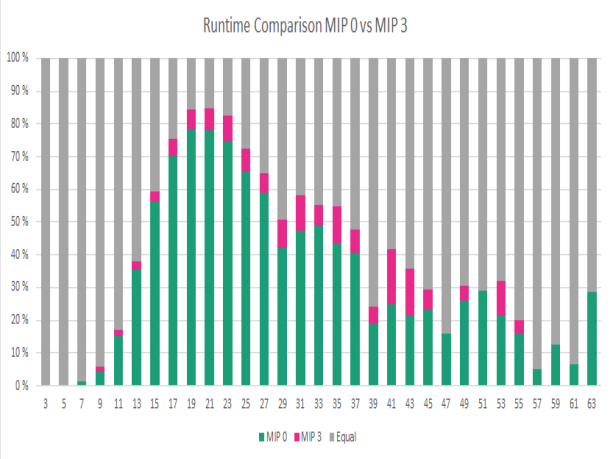
Figure 5: MIP Average Solutions Found



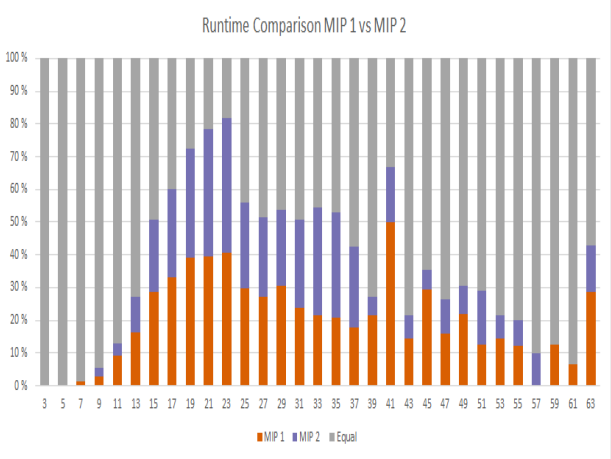
(a) 0 vs 1



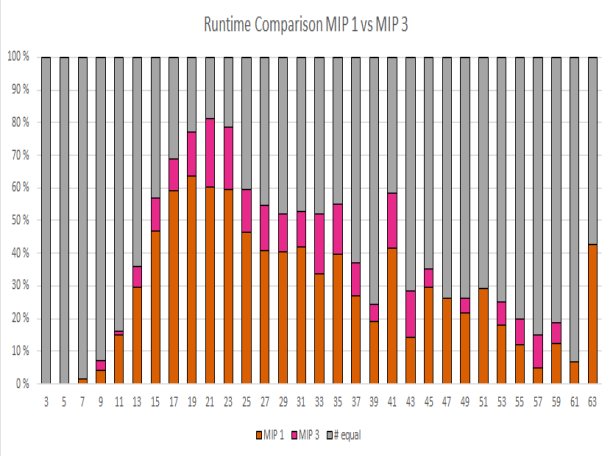
(b) 0 vs 2



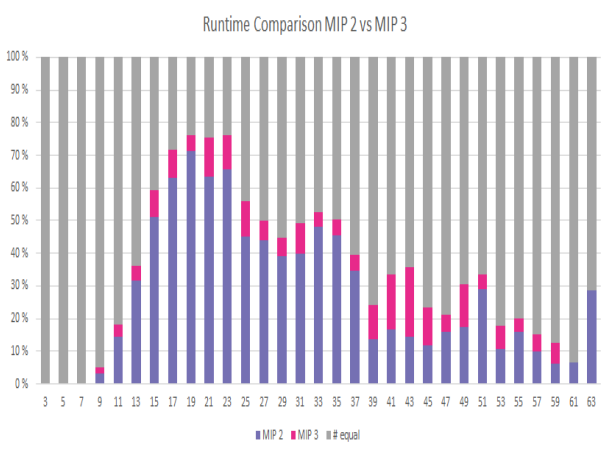
(c) 0 vs 3



(d) 1 vs 2



(e) 1 vs 3



(f) 2 vs 3

Figure 6: MIP Strategy Runtime Comparison

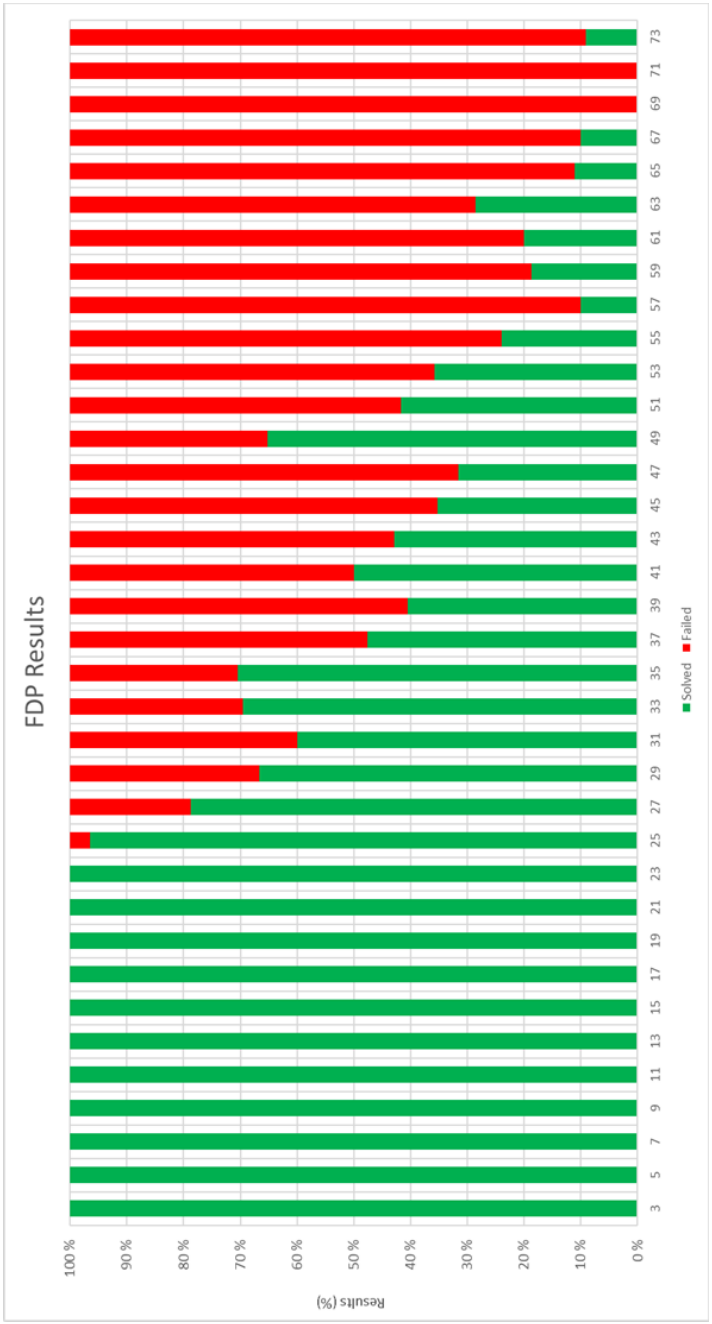


Figure 7: FDP Result Overview

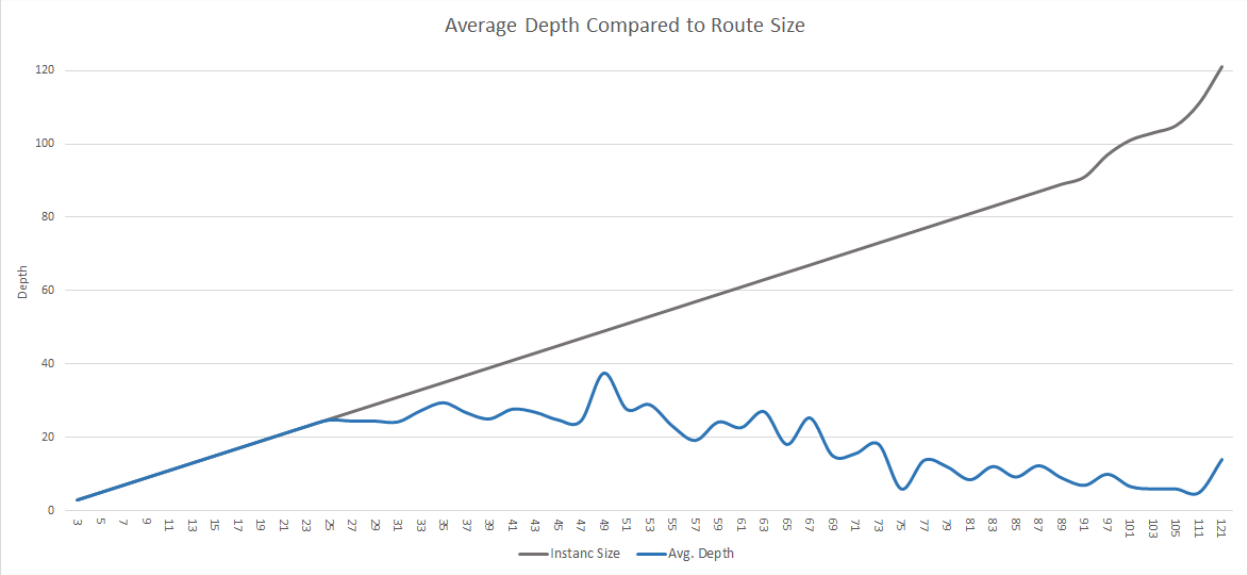
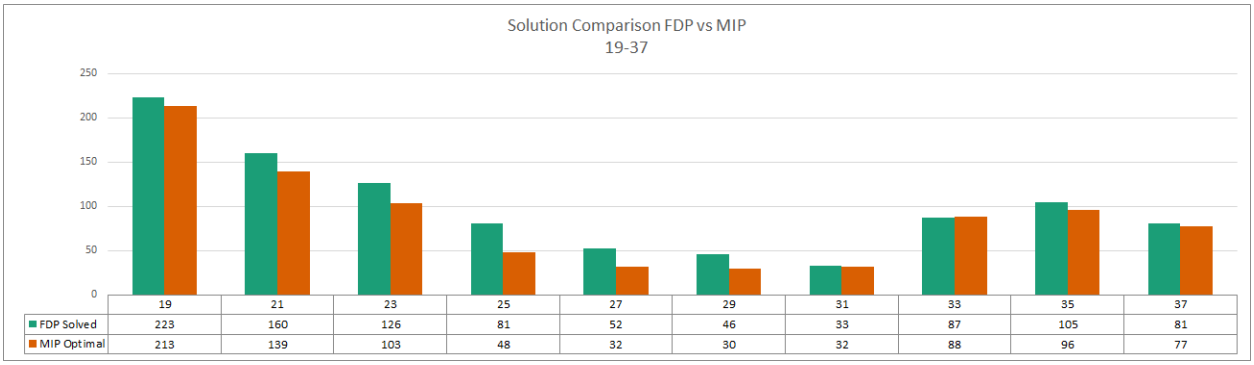
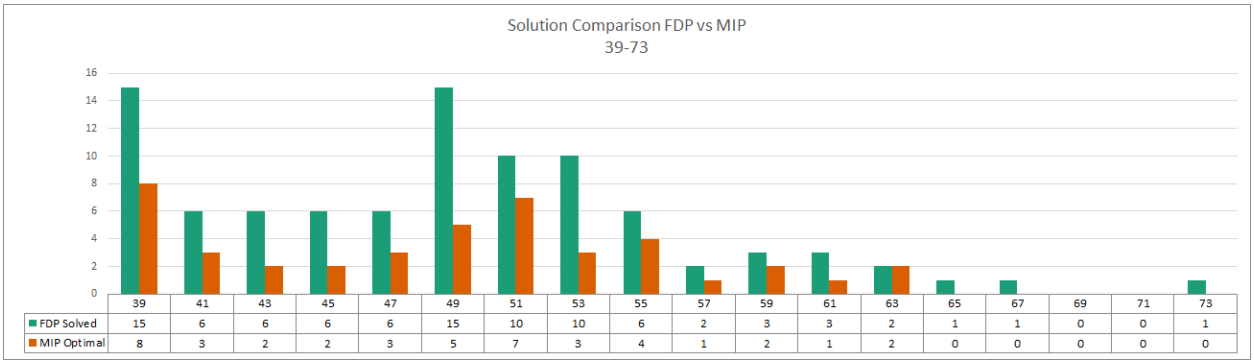


Figure 8: FDP Average Depth Reached Compared to Route Size



	19	21	23	25	27	29	31	33	35	37
FDP Solved	223	160	126	81	52	46	33	87	105	81
MIP Optimal	213	139	103	48	32	30	32	88	96	77

(a) 19-37



	39	41	43	45	47	49	51	53	55	57	59	61	63	65	67	69	71	73
FDP Solved	15	6	6	6	6	15	10	10	6	2	3	3	2	1	1	0	0	1
MIP Optimal	8	3	2	2	3	5	7	3	4	1	2	1	2	0	0	0	0	0

(b) 39-73

Figure 9: FDP vs MIP Number of Solutions Comparison

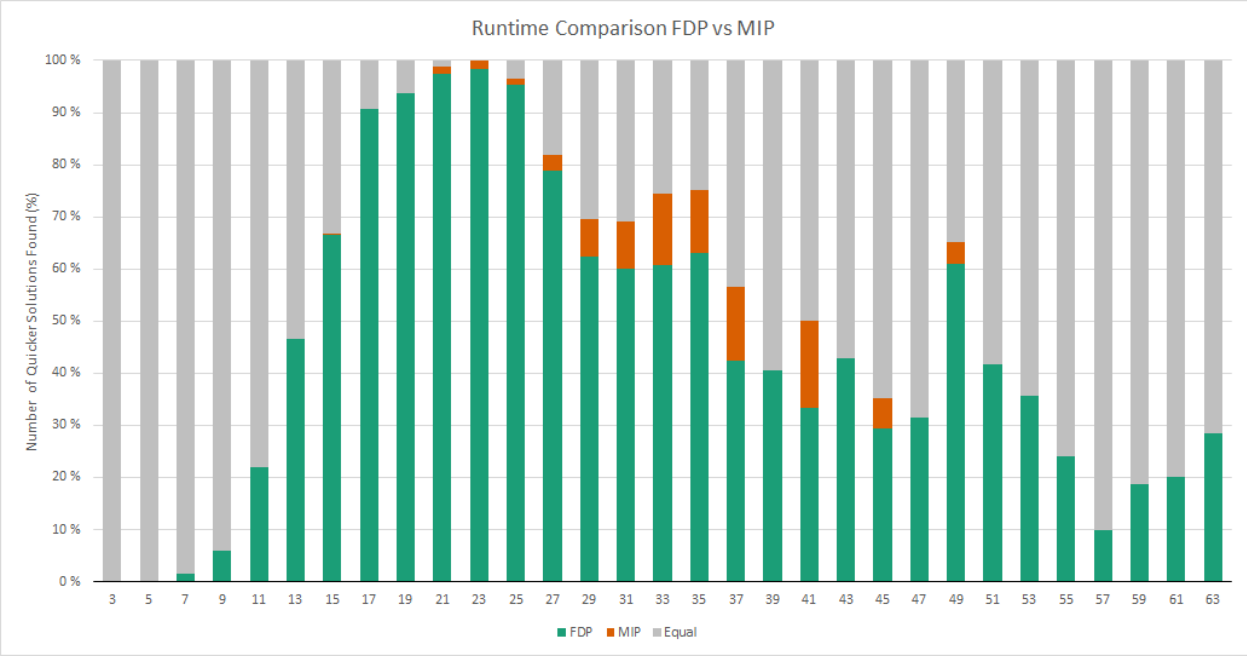


Figure 10: FDP vs MIP Best Runtime Comparison