Erasmus University Rotterdam
Erasmus School of Economics
Econometric Institute

A Guided Dynamic Programming Framework and its Application to Routing Problems

A thesis submitted to the
Econometric Institute
in partial fulfilment of the requirements

for the degree

MASTER OF SCIENCE
in
ECONOMETRICS AND MANAGEMENT SCIENCE

with specialisation
OPERATIONS RESEARCH AND QUANTITATIVE LOGISTICS

by

R.B.O. Kerkkamp

Rotterdam, The Netherlands
August 2013

ERASMUS UNIVERSITEIT ROTTERDAM

MSc THESIS ECONOMETRICS AND MANAGEMENT SCIENCE

"A Guided Dynamic Programming Framework and its Application to Routing Problems"

R.B.O. Kerkkamp

Student number 348154

Erasmus University Rotterdam

Academic supervisor

Prof.dr. A.P.M. Wagelmans

External supervisor

Prof.dr. J.A.S. Gromicho

Co-reader

Dr. W. van den Heuvel

August 2013

Rotterdam, The Netherlands

Abstract

We present the Guided Dynamic Programming Framework: a framework for heuristics that are based on Restricted Dynamic Programming. New to these heuristics is the guidance of the Dynamic Programming search. We illustrate the framework by applying it to the Travelling Salesman Problem and the Vehicle Routing Problem.

This research is performed to conclude the Master in Econometrics and Management Science with the Operations Research and Quantitative Logistics specialisation at the Erasmus University Rotterdam in the Netherlands. It is a cooperative project with ORTEC in Zoetermeer (the Netherlands). ORTEC is a provider of (logistical) planning and optimisation services and has customers worldwide. Due to proprietary reasons, no implemented code is provided.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Research Overview

Our research focusses on solution methods to solve combinatorial optimisation problems. In particular, we consider the Travelling Salesman Problem and the Vehicle Routing Problem. Before we elaborate on the research approach, we introduce these routing problems.

1.1.1 Travelling Salesman Problem

Given the locations of multiple customers and the travel distance between each pair of customers (if possible), a salesman is tasked to sequentially visit all customers exactly once. The salesman must start and end its route in its depot, resulting in a tour. The Travelling Salesman Problem is to find such a tour that minimises the travelled distance.

Many solution approaches have been developed, ranging from exact methods to simplistic heuristics. Among exact methods are (Mixed) Integer Linear Programming formulations, Dynamic Programming methods and Branch and Bound methods combined with relaxations and additional bounds. Complex heuristics are Meta-heuristics such as Genetic Algorithms, Ant Colony Optimisation, Tabu Search, Greedy Randomised Adaptive Search (GRASP) and Adaptive Large Neighbourhood Search (ALNS). Basic heuristics include Nearest Neighbour procedures, Insertion algorithms and, of course, the well-known algorithm based on Shortest Spanning Tree combined with Minimum Cost Matching by Chistofides. Local Search methods can be used to further improve solutions. For more information on these approaches, see [4, 20, 25, 35, 40].

1.1.2 Vehicle Routing Problem

Consider the Travelling Salesman Problem, but suppose we have multiple salesmen available to visit the customers. Each salesmen has its own depot (possible at the same location) and can have different travel distances between customers. Customers must be visited exactly once by any salesman, so the different tours of the salesmen must be disjoint. This problem is called the Vehicle Routing Problem, where the salesmen correspond to vehicles.

Most of the algorithms for the Travelling Salesman Problem have been generalised to be applicable to the Vehicle Routing Problem. There are solution approaches based on Integer Programming formulations, Branch and Bound, Dynamic Programming, Set Partition formulations combined with Column Generation, Meta-heuristics, Local Search and more. Additionally, solution methods for the Travelling Salesman can be used directly in combination with Cluster-First Route-Second methods. Cluster-First Route-Second methods first assign customers to vehicles after which multiple Travelling Salesman Problems need to be solved. For an overview and further reading, see [7, 16, 26, 33, 35, 40].

1.1.3 Complexity

The Travelling Salesman Problem is a classic problem that is difficult to solve to optimality, since it is NP-hard (see [32]). Informally, an NP-hard problem can be solved exactly, but for large instances it would require an infeasible amount of time (it is intractable). The Vehicle Routing Problem is also NP-hard, since the Travelling Salesman Problem is a special case of the Vehicle Routing Problem.

Furthermore, these routing problems can be extended with additional (resource) constraints, such as capacity, time windows, precedence and pickup and delivery (see [33, 35]). An example is the Capacitated Vehicle Routing Problem with Time Windows: customers have a load demand and vehicles have a maximum load capacity. Also, the customers must be serviced within a predefined time window (so travel and service times need to be taken into account). The addition of such resource constraints often increases the difficulty of the problem.

Therefore, we need heuristics to construct reasonably good solutions in practice. These heuristics can be based on exact solution methods or use intuitively appealing construction steps. The downside of heuristics is that performance guarantees often cannot be given. In fact, usually there is no heuristic that outperforms all others. Thus, developing heuristics is still an active research area. Our research is a contribution to this research area, in particular for routing problems.

1.1.4 Research Objectives

The main objective of this research is to develop a new general heuristic that can be applied to routing problems. As seen in the introduction of the two classic routing problems, many different solution approaches have been developed. We restrict our research to Dynamic Programming based heuristics and their application to:

- the Travelling Salesman Problem,
- the Capacitated Vehicle Routing Problem with Time Windows.

A single depot and homogeneous vehicles are assumed. For the Vehicle Routing Problem we focus on the pricing problem of a Column Generation approach and the Giant Tour Representation. For both problems we discuss exact Dynamic Programming methods and existing Dynamic Programming heuristics. These allow us to develop new heuristics based on Dynamic Programming. We will evaluate the effect of the parameters and the scalability of these heuristics and benchmark their performance using literature instances.

The underlying thought behind these new heuristics is to apply a preprocessing phase to determine high potential expansions (good completions) of partial solutions. This knowledge is then used to guide the Dynamic Programming search. Both the preprocessing and the guidance can have many different implementations. These heuristics are therefore placed in a general framework: the Guided Dynamic Programming Framework.

1.1.5 Structure

In Chapter 2 we develop the Guided Dynamic Programming Framework, mainly in context of the Travelling Salesman Problem. Thereto, we first discuss an exact Dynamic Programming method for the Travelling Salesman Problem and existing heuristics based on this exact method.

Chapter 3 presents three Guided Dynamic Programming heuristics for the Travelling Salesman Problem. An evaluation of the effects of the parameters is performed and performance results are given for a selection of literature benchmark instances.

The application to the Vehicle Routing Problem is discussed in Chapter 4. Exact methods are treated first, from which we develop a Guided Dynamic Programming heuristic based on Column Generation. Again, literature benchmarks are used for an evaluation of the heuristic.

Our conclusions are presented in Chapter 5, including topics for future research. The appendix consists of examples, some theoretical properties and all numerical results.

We assume the reader is familiar to a certain extent with the Travelling Salesman Problem, the Vehicle Routing Problem, Dynamic Programming and Column Generation solution methods. These topics will be introduced accordingly, but perhaps not in sufficient detail. Of course, references are given for further reading. However, we do not treat general complexity theory (such as the definitions of NP-hard problems and pseudopolynomial-time algorithms). For an introduction into complexity theory, see [1, 32].

Chapter 2

Guided Dynamic Programming Framework

2.1 Introduction

We introduce the fundamentals of Dynamic Programming and the Dynamic Programming approach by Bellman, Held and Karp for the Travelling Salesman. There are several heuristics based on this exact method which we use to develop a Guided Dynamic Programming heuristic. We place this heuristic in a more general framework called the Guided Dynamic Programming Framework.

The exact Dynamic Programming method is described in Section 2.2, starting with the fundamentals, followed by an approach for solving the Travelling Salesman Problem and general improvements for this approach. In Section 2.3 we discuss Dynamic Programming heuristics from the literature and a local improvement step. Section 2.4 presents the Guided Dynamic Programming Framework based on information extraction, elite edge selection and guidance.

2.2 Dynamic Programming

Optimisation problems that satisfy the Principle of Optimality can be solved exactly by solving subproblems of increasing size. Dynamic Programming is an exact solutions method for such problems. In the following section we will give the basic fundamentals of Dynamic Programming, followed by its application to the Travelling Salesman Problem. This classic approach has several improvements that (usually) accelerate the method without losing the optimality guarantee. We discuss a selection of these improvements to give an indication of possible extensions for high quality solution methods.

2.2.1 Fundamentals

Dynamic Programming (DP) is an exact solution approach for an optimisation problem that has a certain (solution) structure. The problem can be seen as determining a sequence of decisions, each time in terms of its current state of the system. The sequence of decisions can be split into subproblems and can be solved to optimality by solving the subproblems to optimality. Such a problem satisfies the Principle of Optimality:

> "an optimal sequence of decisions has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal sequence of decisions with regard to the state resulting from the first decisions" [3].

Based on this principle, a recursive equation describes the relation between the optimal objective values of the subproblems. Given the optimal solutions of previous (smaller) subproblems, the optimal solution of the current subproblem can be determined from the recursive equation. For minimisation a typical form of this equation is

$$F_{k+1}(j) = \min_{i \in D_{k+1}(j)} \{F_k(i) + c_k(i,j)\}.$$

Here $F_k(i)$ is the optimal objective value of the k-th smallest subproblem with state i of the subproblem state space. The set $D_{k+1}(j)$ is the set of possible states of the k-th subproblem as origin for the $(k+1)$-th subproblem with state j. The objective cost to expand a k-th subproblem from state i to state j is defined by $c_k(i,j)$. Of course, the smallest subproblems have suitable objective values and can be seen as initial values for the recursive relation.

Intuitively, the recursive relation allows us to solve subproblems, iteratively increasing in size, until the overall problem is solved. In case of the example relation given above, the origin with the lowest objective value after expanding is chosen and gives the optimal solution for the larger subproblem.

Often an optimisation problem allows a set of dominance relationships between subproblems. A dominance relationship is a set of conditions, based on properties of feasible solutions for the compared subproblems, such that (solutions to) a subproblem can be discarded and ignored in all recursive relations without losing the optimality guarantee for the overall problem. Simply put, a solution dominates another if it is better in all aspects (e.g. feasibility and objective value). The dominated solution can be forgotten without compromising the ability to find the optimal overall solution. Dominance relationships often allow a significant reduction in the number of evaluated subproblems, thus reducing the effort to solve the overall problem.

The following section describes the Dynamic Programming solution method for the Travelling Salesman Problem. This classic and fundamental optimisation problem will serve as a leading example to make the Dynamic Programming methods of this section more tangible. In fact, most if not all (heuristic) methods are presented based on the DP for the Travelling Salesman Problem. These often have a natural way to be applied to other optimisation problems.

2.2.2 DP for the Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a classic problem which is easy to state but difficult to solve. Given an origin, a set of cities to be visited and their pairwise distances, which tour starting and ending in the origin and visiting each city exactly once minimises the total distance travelled? It is known that this problem is NP-hard (see for instance [32] for a reduction from the Hamiltonian Cycle problem). Let us introduce the notation for the Travelling Salesman Problem.

Consider a directed graph $G = (V, E)$, a starting node $s \in V$ and a distance function $d : E \to \mathbb{R}$. The starting node is also called the depot. Let the number of nodes be $|V| = N$. We will often need the set of nodes excluding the starting node, defined by $\hat{V} = V \setminus \{s\}$ with $|\hat{V}| = \hat{N} = N - 1$. Furthermore, the edges of the graph G restricted to $\hat{V}$ are $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$.

The Travelling Salesman Problem is equivalent to finding a Hamiltonian cycle with minimal travelled distance. A Hamiltonian cycle T in graph G is represented by $(s, v_{T(1)}, \ldots, v_{T(\hat{N})}, s)$, where

$$\left\{v_{T(i)} : i \in \{1, \ldots, \hat{N}\}\right\} = \hat{V}.$$

For feasibility it must hold that all used edges are allowed:

$$\begin{aligned} (s, v_{T(1)}) &\in E, \\ (v_{T(i)}, v_{T(i+1)}) &\in \hat{E} \qquad\qquad \forall\, i \in \{1, \ldots, \hat{N} - 1\}, \\ (v_{T(\hat{N})}, s) &\in E. \end{aligned}$$

The goal is to minimise the total travelled distance $d(T)$ of a feasible Hamiltonian tour T, where the total travelled distance is given by

$$d(T) = d(s, v_{T(1)}) + \sum_{i=1}^{\hat{N}-1} d(v_{T(i)}, v_{T(i+1)}) + d(v_{T(\hat{N})}, s).$$

Since there are $\hat{N}!$ possible tours when the graph is complete ($E = V \times V$), a Brute Force approach of evaluating all tours is infeasible for even small complete graphs. Bellman [4] and Held and Karp [20] independently derived a Dynamic Programming solution method with running time $\mathcal{O}(N^2 2^N)$. Currently, this is still the best running time for all known exact methods for the TSP.

In contrast to Brute Force and many heuristics, Dynamic Programming has (unordered) sets instead of (ordered) tuples as state space. The DP method considers subsets of all nodes, starting with subsets of cardinality one, and expands these subsets until all nodes are included.

These subsets correspond to all nodes that have already been visited. As each subset can end in one of the visited nodes, all these possibilities are evaluated. The expansion of subsets is based on the Principle of Optimality.

The states of the Dynamic Programming state space are defined by a set $S \subseteq \hat{V}$ of visited nodes (customers) and an ending node $v \in S$. Furthermore, a final state (V, s) corresponds to a feasible tour. Hence, a state can be depicted by (S, v) in general. Each state (S, v) has an objective label $\mathcal{L}_o(S, v) \in \mathbb{R}$ and a path label $\mathcal{L}_p(S, v) \in V$.

The objective label $\mathcal{L}_o(S, v)$ stores the travelled distance of the shortest path starting in s and ending in v that visits all nodes in S exactly once (and no other nodes). An exception is state (V, s) where the starting node is visited twice (a tour). The path label $\mathcal{L}_p(S, v)$ stores the node in S that precedes v in the corresponding path. Therefore, we can reconstruct the optimal path by backtracking from state (V, s).

The DP method has $|V|$ stages, corresponding to the cardinality of the set $S \subseteq V$ of visited nodes (excluding the first starting node). These stages are considered in increasing order and during each stage all corresponding states are expanded. The first states are initialised to the distance between s and the visited node $v \in \hat{V}$, where $(s, v) \in E$:

$$\begin{aligned} \mathcal{L}_o(\{v\}, v) &= d(s, v), \\ \mathcal{L}_p(\{v\}, v) &= s. \end{aligned}$$

States that cannot be reached are set to infeasible: an infinite travelled distance and an empty path. In stage $\sigma \in \{1, \ldots, |\hat{V}| - 1\}$ the following recursive equations are used for each $S \subseteq \hat{V}$ with $|S| = \sigma$ and $v_j \in \hat{V} \setminus S$:

$$\mathcal{L}_o(S \cup \{v_j\}, v_j) = \min \{\mathcal{L}_o(S, v_i) + d(v_i, v_j) : v_i \in S, (v_i, v_j) \in E\}, \tag{2.2.1}$$

$$\mathcal{L}_p(S \cup \{v_j\}, v_j) = \operatorname{argmin} \{\mathcal{L}_o(S, v_i) + d(v_i, v_j) : v_i \in S, (v_i, v_j) \in E\}. \tag{2.2.2}$$

The optimal travelled distance of a tour (if it exists) is found by

$$\begin{aligned} \mathcal{L}_o(V, s) &= \min \left\{\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) : v_i \in \hat{V}, (v_i, s) \in E\right\}, \\ \mathcal{L}_p(V, s) &= \operatorname{argmin} \left\{\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) : v_i \in \hat{V}, (v_i, s) \in E\right\}. \end{aligned}$$

The Dynamic Programming method is depicted in Algorithm 2.2.1. For clarity, we have omitted the state path label $\mathcal{L}_p$ in the algorithm. Its running time is $\mathcal{O}(N^2 2^N)$, since there are $\mathcal{O}(N 2^N)$ states and to determine the label of each state we need $\mathcal{O}(N)$ time (see Equation (2.2.1)). Appendix A.2 contains an example of DP applied to the Travelling Salesman Problem.

We introduce some terminology which we will often use. We call the described exact DP method the Full Dynamic Programming method, in order to discern it from heuristics based on DP. A state can be expanded by an edge in E if it maintains feasibility of completing a tour. For instance, expanding with an edge ending in s is only allowed for states of the last stage ($|\hat{V}|$). These allowed edges are called the out-links of that state. Each node v_i in (2.2.2) corresponds to edge (v_i, v_j). These edges are the in-links of state $(S \cup \{v_j\}, v_j)$. If $\mathcal{L}_p(S \cup \{v_j\}, v_j) = v_i$ we call v_i the origin of state $(S \cup \{v_j\}, v_j)$.

Often we abuse terminology related to the recursive relation. If we expand (the path of) state (S, v_i) to v_j, we say that edge (v_i, v_j) is ‘expanded’. If it is also the minimiser in (2.2.1), we say that the edge is ‘used’.

2.2.3 Improvements to DP

There are several ways to improve the performance of DP without losing optimality. A common approach is to combine DP with Branch and Bound techniques, that is, bounding completion costs of states and reducing the state space (see [30]). For each state in DP a lower bound on any feasible solution obtainable from that state can be calculated. If the lower bound is higher than an upper bound on the optimal solution this state can be removed from consideration.

For instance, lower bounds can be obtained by solving relaxations of the original problem with certain variables fixed (e.g. for the TSP the nodes that have already been visited in the current state). General relaxations are Linear Programming and Lagrangian relaxations. Examples of relaxations for the TSP are the Linear Assignment Problem and the Shortest Spanning (Anti-) Arborescence Problem. An important requirement for these relaxations is that they must be able to be solved fast and efficiently, as a lower bound is needed for each state.

These lower bounds can be combined leading to the additive bounding procedure by Fischetti and Toth [15]. See for instance Van den Hoeven [21] for an application to the TSP of this technique in combination with DP and bounding. In fact, this bounding procedure was used to perform a DP alteration of the well-known A* algorithm.

Mingozzi et al. [31] and Van Hoorn [22] consider performing DP for the TSP in a forward and a backward manner, that is, in the normal and reversed graph respectively. By doing so, the completion costs from a stage onwards can be estimated with increasing accuracy as the iterative forward and backward DP proceeds. Also, when a connection can be made between the forward and backward DP a feasible solution is found and the upper bound can be updated. This technique can be incorporated in other applications of Dynamic Programming.

Also, improvements for special cases of the TSP exist. For instance, Bloemendal [5] shows some of these cases for the TSP (and VRP), including a doubling-up procedure for TSP instances with certain symmetries and a way to split an instance into smaller instances in case of identical nodes. Furthermore, for symmetric instances only half of the DP state space is required.

We will not consider these general improvements for Dynamic Programming. Incorporating these methods can greatly change the performance and behaviour of the DP variations that follow in the next sections. Therefore, care must be taken if one wants to include these improvements.

Algorithm 2.2.1 Full Dynamic Programming for TSP

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$
Output: minimum travelled distance of Hamiltonian cycle if one exists

```
 1: procedure TSP-Full-DP(input)
 2:     initialise objective state space labels: $\mathcal{L}_o(S, v) = \infty$ for all $S \subseteq V$ and $v \in S$
 3:     set: $\hat{V} = V \setminus \{s\}$

 4:     for all nodes $v_j \in \{v \in \hat{V} : (s, v) \in E\}$ do
 5:         new objective state space label: $\mathcal{L}_o(\{v_j\}, v_j) = d(s, v_j)$

 6:     for all stages $\sigma = 1, \ldots, |\hat{V}| - 1$ do
 7:         for all subsets $S_i \in \{S \subset \hat{V} : |S| = \sigma\}$ do
 8:             for all nodes $v_i \in S_i$ do
 9:                 for all nodes $v_j \in \{v \in \hat{V} \setminus S_i : (v_i, v) \in E\}$ do
10:                     if $\mathcal{L}_o(S_i, v_i) + d(v_i, v_j) < \mathcal{L}_o(S_i \cup \{v_j\}, v_j)$ then
11:                         new objective state space label: $\mathcal{L}_o(S_i \cup \{v_j\}, v_j) = \mathcal{L}_o(S_i, v_i) + d(v_i, v_j)$

12:     for all nodes $v_i \in \{v \in \hat{V} : (v, s) \in E\}$ do
13:         if $\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) < \mathcal{L}_o(V, s)$ then
14:             new objective state space label: $\mathcal{L}_o(V, s) = \mathcal{L}_o(\hat{V}, v_i) + d(v_i, s)$

15:     return $\mathcal{L}_o(V, s)$
```

2.3 Restricted Dynamic Programming

Although Dynamic Programming can be a very efficient exact method for several optimisation problems, it has its limitations for larger Travelling Salesman Problem instances due to its exponential running time. Restricted Dynamic Programming is a broad class of heuristics that are based on Full DP, but restrict the search in the DP state space in some way. The optimality guarantee is lost in favour for (greatly) improved running times, allowing us to find relative good solutions for large instances.

We will give two versions of Restricted DP from the literature, where one is an extension of the other. Furthermore, we present a local improvement step for these methods. Although these approaches can be used for other optimisation problems as well, we only consider their application to the TSP.

2.3.1 State Restriction

The disadvantage of the Bellman-Held-Karp DP method is the required exponential space and running time, making it practically impossible to solve very large instances. Malandraki and Dial [29] truncate the state space to at most B states per stage. At each stage only the B most promising states are expanded. For instance, B states with the smallest travelled distance so far can be selected. This approach looks only locally when selecting and suffers from near-sightedness. This heuristic can be seen as a beam search restricting the number of states and we will refer to it as the State Restricted DP (Algorithm 2.3.1). An example is shown in Appendix A.2.

Of course, different selection criteria can be used (the ranking of states in Algorithm 2.3.1). For instance, if more constraints are added to the TSP, e.g. time windows, we should carefully determine suitable selection criteria. Note that the first couple of stages are exact (depending on the beam width) and we get the Full DP method if we set the beam to (or larger than)

$$B = \begin{cases} \frac{|\hat{V}|}{2}\binom{|\hat{V}|}{|\hat{V}|/2} & \text{if } |\hat{V}| \text{ even} \\ \frac{|\hat{V}|+1}{2}\binom{|\hat{V}|}{(|\hat{V}|+1)/2} & \text{if } |\hat{V}| \text{ odd} \end{cases},$$

as this is the maximum number of states among all stages.

2.3.2 State and Out-link Restriction

Gromicho et al. [19] restrict the state space even further by limiting the number of out-links from each state. Instead of expanding a state to all nodes not yet visited by the state, only L (feasible) expansions are performed. Of course, if less than L expansions are possible, all are used. For instance, we could restrict the expansion to the L nearest unvisited neighbours. Limiting the out-links can be seen in context with Granular Tabu Search such as in [39], where long edges are removed from the graph. The State and Out-link Restricted Dynamic Programming method is shown in Algorithm 2.3.2.

In general, the running time is $\mathcal{O}(N^2B\log{(N)} + NBL\log{(BL)})$. Each stage we need to sort the corresponding states ($\mathcal{O}(BL)$ in total), requiring $\mathcal{O}(BL\log{(BL)})$ time. Furthermore, for each stage and each state to be expanded the out-links need to be sorted, taking $\mathcal{O}(N^2B\log{(N)})$ time in total. The running time can be improved with a more efficient implementation. If we keep a sorted list (structure) of up to B states that will be expanded, the running time improves to $\mathcal{O}(N^2B\log{(N)} + NBL\log{(B)})$.

If the ranking of the out-links is fixed through all stages, the sorting only needs to be performed once. This leads to a running time of $\mathcal{O}(NBL\log{(B)})$. Note that we can regain the State Restricted DP by selecting $L = \hat{N} = |\hat{V}|$. Thus, for the State Restricted DP we have a running time of $\mathcal{O}(N^2B\log{(B)})$.

2.3.3 Local Improvement

A local improvement step can be included in the State and Out-link Restricted DP method. We can check for each of the B selected states if any previously unevaluated in-link can improve the current path. Since the out-links are restricted, not all feasible in-links of a state are evaluated.

Checking for local improvements for all states before pruning to B states is equivalent to using all out-links. Therefore, only a selected number of states must be checked. For instance, only the B states that are expanded can be re-evaluated. Usually, these B states are the states with the smallest travelled distance so far (the best objective values). It is therefore likely that only a few improvements are found, as these states are already good. Instead, one can prune to a multiple of B states (say $3B$) and search for local improvements. Then prune again to the B best states, but using the updated objective values. This way, it is more likely that improvements are found. However, it is not clear a priori if local improvement is beneficial due to the extra computational time.

Algorithm 2.3.1 State Restricted Dynamic Programming Heuristic for TSP

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$, state beam width $B \in \mathbb{N}_{>0}$
Output: travelled distance of Hamiltonian cycle if one exists

```
1:  procedure TSP-State Restricted-DP(input)
2:      initialise objective state space labels: $\mathcal{L}_o(S, v) = \infty$ for all $S \subseteq V$ and $v \in S$
3:      set: $\hat{V} = V \setminus \{s\}$

4:      for all nodes $v_j \in \{v \in \hat{V} : (s, v) \in E\}$ do
5:          new objective state space label: $\mathcal{L}_o(\{v_j\}, v_j) = d(s, v_j)$

6:      for all stages $\sigma = 1, \ldots, |\hat{V}| - 1$ do
7:          for all states $(S_i, v_i)$ in States to Expand$(\sigma, B)$ do
8:              for all nodes $v_j \in \{v \in \hat{V} \setminus S_i : (v_i, v) \in E\}$ do
9:                  if $\mathcal{L}_o(S_i, v_i) + d(v_i, v_j) < \mathcal{L}_o(S_i \cup \{v_j\}, v_j)$ then
10:                     new objective state space label: $\mathcal{L}_o(S_i \cup \{v_j\}, v_j) = \mathcal{L}_o(S_i, v_i) + d(v_i, v_j)$

11:     for all states $(\hat{V}, v_i)$ in States to Expand$(|\hat{V}|, B)$ do
12:         if $\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) < \mathcal{L}_o(V, s)$ then
13:             new objective state space label: $\mathcal{L}_o(V, s) = \mathcal{L}_o(\hat{V}, v_i) + d(v_i, s)$

14:     return $\mathcal{L}_o(V, s)$


15: procedure States to Expand(stage $\sigma$, state beam $B$)
16:     for all subsets $S_i \in \{S \subseteq \hat{V} : |S| = \sigma\}$ do
17:         for all nodes $v_i \in S_i$ do
18:             rank state $(S_i, v_i)$
19:             insert state in list according to rank of $(S_i, v_i)$

20:     return up to $B$ highest ranked elements from list
```

Algorithm 2.3.2 State and Out-link Restricted Dynamic Programming Heuristic for TSP

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$, state beam width $B \in \mathbb{N}_{>0}$, out-link number $L \in \mathbb{N}_{>0}$

Output: travelled distance of Hamiltonian cycle if one exists

```
1:  procedure TSP-State and Out-link Restricted-DP(input)
2:      initialise objective state space labels: $\mathcal{L}_o(S, v) = \infty$ for all $S \subseteq V$ and $v \in S$
3:      set: $\hat{V} = V \setminus \{s\}$

4:      for all nodes $v_j$ in Nodes to Visit$((\emptyset, s), L)$ do
5:          new objective state space label: $\mathcal{L}_o(\{v_j\}, v_j) = d(s, v_j)$

6:      for all stages $\sigma = 1, \ldots, |\hat{V}| - 1$ do
7:          for all states $(S_i, v_i)$ in States to Expand$(\sigma, B)$ do
8:              for all nodes $v_j$ in Nodes to Visit$((S_i, v_i), L)$ do
9:                  if $\mathcal{L}_o(S_i, v_i) + d(v_i, v_j) < \mathcal{L}_o(S_i \cup \{v_j\}, v_j)$ then
10:                     new objective state space label: $\mathcal{L}_o(S_i \cup \{v_j\}, v_j) = \mathcal{L}_o(S_i, v_i) + d(v_i, v_j)$

11:     for all states $(\hat{V}, v_i)$ in States to Expand$(|\hat{V}|, B)$ do
12:         if $\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) < \mathcal{L}_o(V, s)$ then
13:             new objective state space label: $\mathcal{L}_o(V, s) = \mathcal{L}_o(\hat{V}, v_i) + d(v_i, s)$

14:     return $\mathcal{L}_o(V, s)$


15: procedure Nodes to Visit(state $(S_i, v_i)$, out-link number $L$)
16:     for all nodes $v_j \in \{v \in \hat{V} \setminus S_i : (v_i, v) \in E\}$ do
17:         rank edge $(v_i, v_j)$
18:         insert node $v_j$ in list according to rank of $(v_i, v_j)$

19:     return up to $L$ highest ranked elements from list


20: procedure States to Expand(stage $\sigma$, state beam $B$)
21:     for all subsets $S_i \in \{S \subseteq \hat{V} : |S| = \sigma\}$ do
22:         for all nodes $v_i \in S_i$ do
23:             rank state $(S_i, v_i)$
24:             insert state in list according to rank of $(S_i, v_i)$

25:     return up to $B$ highest ranked elements from list
```

2.4 Guided Dynamic Programming

Restricted Dynamic Programming suffers from the need to select states (and out-links) as most promising for further expansion. This is often done in a greedy way by selecting the states with the best objective value or with the best lower bounds on the completion cost of states. Therefore, it can suffer from short-sightedness. We propose a general Dynamic Programming framework that allows the use of information on the optimisation problem at hand to guide the Restricted DP method. That is, we give preference to certain state expansions, hopefully counteracting some negative features of Restricted DP.

We call this framework the Guided Dynamic Programming Framework. The two fundamental questions underlying this method are: how can we detect weak and/or strong expansions and can we guide (Restricted) Dynamic Programming using this information? The framework consists of three parts:

- information extraction,
- elite expansion selection,
- Dynamic Programming guidance.

The information extraction part collects useful information on the problem instance, ranging from simple data as the distance matrix to more complex data gained from meta-heuristics. Based on this information, elite expansions are predefined for each state possible in the Dynamic Programming method. For instance, the shortest out-link (edge) of a state is selected as an elite expansion of that state. Finally, during execution of (Restricted) Dynamic Programming we give preference to elite expansions. Thus, the DP is guided to use these selected expansions. A possible approach is to only allow elite expansions to be made.

In the following sections we will elaborate on each of these three parts. The framework allows many choices in the kind of information extracted, the way elite expansions are selected and the way the DP is guided. See also Figure 2.4.1 for an overview of the framework and the presented choices of each part.

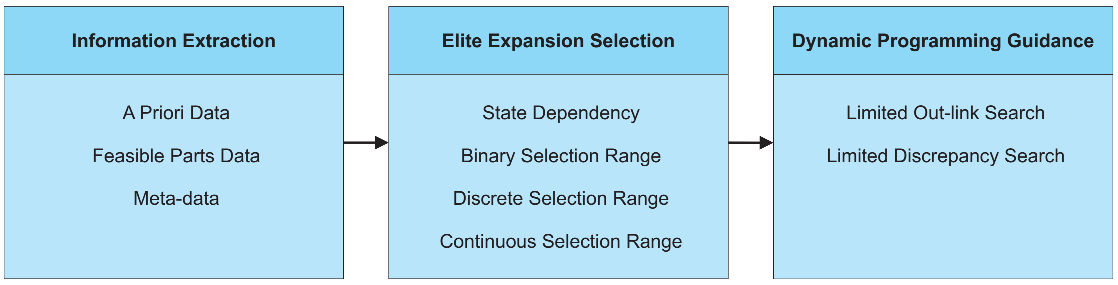


Figure 2.4.1: The Guided Dynamic Programming Framework.

2.4.1 Information Extraction

The extraction of information on the problem instance has the most diverse approaches. All try to evaluate the attractiveness of expansions (e.g. edges) of DP states. We have divided these approaches in three types:

- a priori data,
- feasible parts data,
- meta-data.

Do note that this list does not cover all ways for information extraction, but it gives an indication of the diversity. We will discuss these information sources and give several implementation examples.

2.4.1.1 A Priori Data

A priori data is information available from the definition of the problem instance. One can think of the distance matrix and resource consumption in combination with resource constraints (e.g. capacity or time windows). The trivial choice would be to use no a priori data, in which case we need to extract other information.

2.4.1.2 Feasible Parts Data

Another source of information are the edges in (parts of) feasible solutions (see [6] for a connection with automated learning). If an edge is used frequently among a set of available feasible solutions, it can be interpreted that this edge is highly preferred. Of course, there is no guarantee that this edge is indeed in the optimal (or in any near-optimal) solution. If we have feasible solutions, we can use the objective value to filter out bad solutions or use it in combination with some kind of weight (e.g. similar to Ant Colony Optimisation pheromone levels, see [10]).

For feasible parts data we need (parts of) feasible solutions. Any construction method will suffice, but the interpretation of the value of the feasible solutions would be different. Using a greedy construction method says more about the used edges in the found solutions than using a random construction method. The greedy solutions will most likely use short edges more often than random solutions. Also, having one feasible solution requires a different approach as to when many feasible solutions are available.

To conclude the feasible parts data, we describe a few possible approaches. As mentioned, greedy construction methods can be used. We can create $|V|$ greedy solutions, each one starting in a certain node in V. Most likely, the used edges are very similar between the solutions. The other extreme is to generate random solutions. Here, the used edges will be very diverse (depending on the number of solutions and the size of the instance). An added benefit is that random solutions can be combined with Ordinal Optimisation techniques to give ordinal performance estimates of the final solution (see [37]). Another approach would be to perform a Restricted DP, usually giving multiple feasible solutions. Although DP can be computationally expensive, it can also serve to provide more data than feasible solutions only: meta-data.

2.4.1.3 Meta-Data

With meta-data we mean information that surpasses straightforward data like the distance matrix or feasible solution data. Where a priori data and feasible solutions are easy to interpret, meta-data arises from construction methods as a by-product. An illustrative example is the pheromone weight of edges in Ant Colony Optimisation ([10]).

In Ant Colony Optimisation the pheromone weights are initialised to a uniform value for all edges. Random feasible solutions are constructed, guided by these weights. Then the weights of the edges are updated according to the resulting objective value of the solutions, giving preference to edges used by good solutions. This process repeats itself until convergence. As can be seen, there is a strong resemblance between Ant Colony Optimisation and the proposed Guided DP Framework. After a certain number of iterations, the pheromone weights give an indication of preferred edges, which can be used as information to guide the DP method.

A different approach is to use Restricted Dynamic Programming as a source of meta-data. A common way is to look at edges used by solutions, that is, at the solution space. Although a similar approach can be done for feasible solutions of DP, we can consider the DP state space as information source. Instead of looking at edges used by solutions, we look at edges used for state expansions. We did not encounter similar approaches in the literature, therefore we elaborate this aspect in the next section.

Meta-data is not restricted to the two described methods. Other heuristics could turn out to be useful in this aspect, resulting possibly in a fascinating synergy with Dynamic Programming. If DP is used to solve subproblems connected to a master problem (e.g. with Column Generation), one can think of extracting information from the master problem. For instance, we can keep track of edges that are repeatedly used by columns selected over several iterations of the Column Generation master problem.

2.4.1.4 Edge Counters from the DP State Space

The Dynamic Programming approach requires the construction of the full state space. This is computationally intensive, but our idea is that it also gives a lot of information about the problem at hand. This is based on the optimality criterion that each subset of adjacent nodes in the tour is ordered optimally in the optimal tour. DP constructs many paths, each optimally ordered when restricted to the subset of visited nodes (and starting and ending node of the path).

Suppose a common ordering of some nodes exists among these paths. If we would be able to detect these, we can limit the DP search to using these common orderings. This idea is similar to the elite parts seen in [38], but there common parts in feasible tours are used. We propose to look at feasible paths from the DP state space, which are all optimal in a certain 'neighbourhood' (the subset of visited nodes with fixed starting and ending nodes). In a sense, we investigate if we can detect weak and/or strong expansions (out-links) from the DP state space.

In practice, we would use Restricted DP and extract elite parts from the incomplete state space. We can then perform another Restricted DP, but this time using only elite parts. Before we can develop such a method, we need to look at the Full DP method. The state space of Full DP can be seen as the best possible information source we could ever get from Restricted DP. Hence, we first need to determine which information can be extracted from the full state space.

The most straightforward information is to see which edges are part of optimal paths in the DP state space. This information is aggregated for each edge, that is, we only look locally and forget any connection with the corresponding state or stage. We track the following occurrences for each edge (see Algorithm 2.4.1):

C_{exp} **Edge Expansion Counter**
The number of times a state is expanded by this edge, possibly improving an earlier found path, i.e. the edge is evaluated.

C_{use} **Edge Usage Counter**
The number of times a state is expanded by this edge as part of an optimal path, i.e. the edge is used.

Appendix A.2 contains an example how to determine the edge counters.

Our reasoning for using these counters is as follows: edges that are often used as part of optimal paths (minimiser in-links in Equation (2.2.1)) are clearly important to get exact state space labels. In fact, if an edge is never used, it can be removed from the instance without losing optimality. Similarly, giving preference to frequently used edges makes more sense than doing so for rarely used edges, though one must be careful with this intuition without any proof.

The edge expansion counters keep track of the number of times an edge was eligible to be an in-link of a state. It has no value when applying Full DP, as these are equal for all edges. The proof is given in Appendix B, in addition to general bounds and properties of both edge counters derived from the Full DP state space. Furthermore, we give some remarks on counters derived from the Restricted DP state space.

When applying Restricted DP the edge expansion counter differs between edges and can be used for normalisation. Suppose we have two edges which are both used ten times. However, the first edge has an expansion counter of 10, the second 100. Thus, the first edge was always used if expanded, the second edge on average only once per 10 expansions. With this normalisation, the first edge shows more potential. Such normalisation leads to edge scores, a measure of attractiveness of edge expansions.

2.4.1.5 Edge Scores

We propose the following approach to use the edge counters. Since the expansion counter shows how often a sequence is evaluated and the usage counter how often it is actually used, dividing the usage counter by the expansion counter gives an indication of the attractiveness of that sequence. For an edge (v_i, v_j) we define the edge attractiveness score as follows:

$$A(v_i, v_j) = \frac{C_{use}(v_i, v_j)}{C_{exp}(v_i, v_j)} \in [0, 1].$$

That is, we normalise the edge usage counter by dividing it by the edge expansion counter. Often, we call A the score of an edge. If an edge has a high edge score (close to one), it indicates that it is often optimal to expand a state with that edge. In the extreme case that the score is one, that state should always be expanded by that edge. Note that for a fixed node there can be multiple edges from that node with high scores. Therefore, multiple out-links can have high potential.

Likewise, if an edge has a low score (close to zero), the edge is rarely used and can be removed from the graph, potentially without losing optimality. If the score is zero, one can indeed remove that edge from the graph without losing optimality. Otherwise, it is not clear a priori.

Restricting to edges in $\hat{E}$ (without the starting node), we know the average value of the usage counter after completing Full DP (see Equation (B.2.1) for the derivation):

$$\mathbb{E}_{\hat{E}}\left[C_{use}\right] = \frac{2^{|V|-2}-1}{|V|-2}.$$

Therefore, the average edge score (still restricted to $\hat{E}$) is:

$$\mathbb{E}_{\hat{E}}\left[A\right] = \mathbb{E}_{\hat{E}}\left[\frac{C_{use}}{C_{exp}}\right] = \left(\frac{1}{2^{|V|-3}}\right)\mathbb{E}_{\hat{E}}\left[C_{use}\right] = \frac{2^{|V|-2}-1}{(|V|-2)2^{|V|-3}} \approx \frac{2}{|V|-2}, \tag{2.4.1}$$

which goes to zero if the number of nodes $|V|$ increases. We can use this average edge score as an indication for the extremity of the score of an individual edge under Full DP. Note that we do not have similar results for the edge counters under Restricted DP (see also Appendix B.3). Hence, we cannot provide an estimate for the extremity of scores under Restricted DP.

Under Restricted DP some caution is advised when comparing edge scores as this comparison has some potential downsides. For instance, it could be that under Full DP two edges actually have equal counters, whereas this is not the case under Restricted DP. Another downside, is that counters under Restricted DP with relatively small beam are biased, which we discuss next.

2.4.1.6 Biased Counters under Restricted DP

Consider applying the Restricted DP method. If a state (S, v_j) can only be reached by exactly one state $(S \setminus \{v_j\}, v_i)$ from the previous stage, its contribution to the counters is biased. Edge (v_i, v_j) is expanded and used, so both counters would be incremented. However the expansion does not provide any information about the attractiveness of edge (v_i, v_j) with respect to the other in-links of state (S, v_j) (available in Full DP). A relative high usage counter of an edge indicates a high potential for it being a good expansion. Therefore, incrementing the counters in the case of no competition results in a bias (increasing the attractiveness or potential).

In order to reduce this bias we can only update the counters if there were multiple choices (in-links). So only if a state can be reached by more than one previous state, we increment the counters appropriately. Algorithm 2.4.2 shows the Restricted Dynamic Programming method with edge counters where this bias adjustment is applied. In order to do so, we have to keep track if a state can be reached by multiple in-links (see the in-link counter label $\mathcal{L}_{in}$). Most of the derived results for the counters when applying Full DP (in Appendix B.2) should be decreased by one if we perform this bias correction step. All of our numerical results use this small bias adjustment for the counters.

Similarly, if we restrict the number of out-links during State and Out-link Restricted DP, less connections are evaluated. This greatly affects the reliability of the edge counters, because of limited competition among in-links. Therefore, we will not limit the number of out-links when we want to extract edge counters.

A different way to deal with the counter bias is to incorporate weights. Say there are k in-links to a state when performing Restricted DP and edge (v_i, v_j) is the in-link used. Instead of incrementing the usage counter of edge (v_i, v_j) by one, we can increment it by $(k-1)$ (or use similar approaches). The more competing in-links there are, the higher the reliability that edge (v_i, v_j) is also used in the exact case (Full DP). Similar theoretical results for the usage counters under Full DP can be derived. We will not use weights in our methods.

We can also apply the local improvement method from Section 2.3.3, where the number of evaluated in-links is (most likely) increased. This results in more competing in-links and more reliable counters. Note that the state space objective values do not need to be improved in order to get more reliable counters: if a new in-link does not lead to improvements, its attractiveness decreases.

2.4.1.7 Other Counters

The edge expansion and usage counters are in a sense the smallest quantifiable information elements one can extract from the DP state space. A straightforward generalisation is to consider the expansion and usage counters for larger elements, that is, to consider a number of edges at the same time. We will call this generalisation 'sequence counting'.

The sequence counting approach looks at the order in which nodes are visited in (optimal) paths in DP. Instead of counting usage of sequences of two nodes (v_i, v_j) (corresponding to edges), we count sequences of more nodes. For instance, sequences (v_i, v_j, v_k) of three nodes, which can be depicted by three-dimensional matrices. Of course, counting larger sequences would result in sparser counter matrices (more sequences will not be used by DP).

As mentioned, the idea behind these counters is to detect elite expansions from the DP state space. Given the elite parts, we can limit DP to these elite sequences. The edge expansion and usage counters (sequences of two nodes) can be seen to provide short-sighted elite parts (elite edges): practically only the ending node v of a state (S, v) determines which edges can be used to expand this state. Sequences of three nodes give elite parts that depend on the last two nodes visited in a state. Therefore, larger elite sequences should suffer less from this short-sightedness.

Another variation to the edge counters is to make snapshots during selected stages of DP. In a certain stage we can backtrack all current states to get (optimal) paths. From these paths we can count the used edges (or used sequences in general). If we do this in the final stage, we are determining elite parts from a complete tour, more similar to the approach in [38].

A different way of counting would be to consider (unordered) sets of nodes visited in optimal paths in DP. Instead of counting usage of sequences (v_i, v_j) and (v_j, v_i), we only count usage of subset $\{v_i, v_j\}$. In this case, the sequence counters of (v_i, v_j) and (v_j, v_i) should be added together to give the counter of $\{v_i, v_j\}$. This can easily be generalised to larger subsets.

This set counting approach has a connection to the classic Cluster-First Route-Second method (and hypergraphs). If we limit DP to elite subsets we are scheduling clusters (corresponding to the subsets) in a particular order. A difference is that the subsets can overlap (they can be non-disjunct), which can be seen as uncertainty to which cluster a node belongs.

To conclude, there is a large variety of counters to extract from the (Restricted) DP state space. All these counters can be treated in a similar way. Under Restricted Dynamic Programming it is important to normalise the usage counter, for instance with the expansion counter. For a sequence $p = (v_{p(1)}, \dots, v_{p(k)})$ of size $k \geq 2$ we can define

$$A(p) = \frac{C_{use}(p)}{C_{exp}(p)} \in [0, 1],$$

as the attractiveness score of that sequence (and similarly for the set counting approach).

2.4.2 Elite Expansion Selection

Given information on the potential or attractiveness of expansions, we have to select which expansions we classify as elite. The Guided DP method will be encouraged to primarily use these elite expansions. As there are many different sources of information, we limit ourselves to the following types: distance matrix as a priori data, feasible parts data from random solutions and edge scores meta-data.

We discern the following classes of elite expansion selection:

- state independent selection,
- state dependent selection,

combined with one of the following:

- binary selection range,
- discrete selection range,
- continuous selection range.

We will go into the details of each of these classes, giving examples for the different data types. Note that for all classes it is generally advisable to select all edges from and to the starting node as elite. This is beneficial for feasibility and hardly has any effect on running times.

2.4.2.1 State Dependency

The selected elite expansions can be fixed during all stages, resulting a state independent selection. This kind of selection is easier to manage and requires less data storage. It is the more logical choice when using aggregated data with no sense of order, such as edge scores. Using fixed elite edges in combination with distance as information is similar to the granular approach of [39].

Allowing elite expansions to be state (and thus stage) dependent generally gives more flexibility, but can become difficult to manage. We have already seen an example which is easy to manage: Out-link Restricted DP limits the DP search to the L shortest out-links. That is, this method uses the distance matrix as information and selects the shortest edges as elite, depending on the current state. One can also opt to use no guidance during the first (or last) couple of DP stages by setting all edges during those stages to elite.

Feasible parts data can be used in both approaches. For stage n we can set state dependent elite edges equal to the n-th used edge from all solutions. This would result in very similar solutions. For fixed elite edges we can use all used edges from all solutions. This allows more combinations to be made.

Thus, depending on the information source we can introduce more flexibility in the search space by selecting the appropriate state dependency. To simplify matters, we will only consider state independent elite expansions in our methods (except for Out-link Restricted DP). Besides state dependency, we also have three different ways to classify elite edges: using binary, discrete and continuous selection ranges.

2.4.2.2 Binary Selection Range

A binary selection range classifies two types of edges: normal and elite, with no middle ground. This approach is the most transparent, but is highly dependent on good parameter settings. We illustrate binary selection with the three selected data sources.

As previously mentioned, the distance matrix can be used to limit the DP search to short edges (or filter out long edges). With state dependent elite edges we can set the L shortest edges to elite. For state independent elites it makes more sense to only filter out very long edges (making the rest elite), in order to maintain feasibility. One can also work with selection based on distance percentiles.

For feasible parts from random solutions we can select the most common edges as elite. Thereto, we need to set a threshold when an edge is common. This threshold could be a predefined value (e.g. at least 5 times) or be based on percentiles (e.g. at least used by 5% of the solutions). Since we know the objective values of these solutions, we can use only the best 5% solutions and mark all used edges by those solutions as elite.

As a final example, we treat the edge scores as source. As a heuristic we can remove all edges having a score lower than a certain threshold value $\rho \in [0, 1]$. With $\rho = 0$ all edges are classified as elite, whereas with $\rho = 1$ only the edges that are always used are selected. Do note that this threshold value should depend on the size of the instance (the number of nodes), the beam width B and number of out-links L. See for instance Equation (2.4.1) for the case when edge counters are derived from the Full DP state space. In this case, we can use the average edge score as an indication for the extremity of the score of an individual edge and select an appropriate threshold.

Instead of using the threshold value as a predetermined scalar which needs to be adjusted to the instance size and settings, we can also remove all edges that have a score below a certain percentile value. This percentile value ρ is determined by the percentile level $\tau \in [0, 1]$ in the following way:

$$\rho = \min\{\hat{\rho} \in [0, 1] : |\{e \in E : A(e) \leq \hat{\rho}\}|/|E| \geq \tau\},$$

where $A(e)$ is the score of edge $e \in E$. For example, with $\tau = 0.9$ approximately 10% of the edges are selected as elites. This approach seems more suitable for a heuristic as the effect of changing the percentile level is more predictable. Both approaches give us a threshold value $\rho \in [0, 1]$ to make the binary selection.

2.4.2.3 Discrete Selection Range

The discrete selection range is similar to the binary range, except that it allows multiple levels of 'eliteness'. For instance, very short edges (0-20% shortest) can be classified as highly preferable. Medium sized edges (20-50%) as moderately preferable, followed by the rest as normal edges. The penalty for deviating from elite edges can be based on the eliteness level (applicable to the Limited Discrepancy Search DP guidance, to be discussed later). Similar approaches can be done for the frequency of feasible parts and edge scores. We will not implement discrete selection ranges.

2.4.2.4 Continuous Selection Range

If we let the number of discrete eliteness level grow to infinity, we get a continuous spectrum of elite types of edges. With an appropriate continuous penalty function we can guide the DP search with these penalties. A continuous selection range does not have the difficulty of selecting good boundaries for the levels of eliteness. However, the interpretation of the maximum allowed deviation (of the Limited Discrepancy Search DP guidance) is less clear. Therefore, DP guidance becomes more complex. Our methods will not apply a continuous selection range.

2.4.3 DP Guidance

After extracting information on the attractiveness of expansions and classifying these expansions into several levels of eliteness, it remains to use these elite expansions to guide the Dynamic Programming method. We present two ways to guide the DP method:

- Limited Out-link Search,
- Limited Discrepancy Search.

We will start with the most basic guidance called Limited Out-link Search, which is strongly related to a granular graph approach as elite edges must be used. This method is only applicable with a binary elite selection range.

The second guidance, Limited Discrepancy Search, is based on [13, 17]. It allows a limited deviation from elite edges and can be seen as a generalisation of Limited Out-link Search. Feillet, Gendreau and Rousseau [13] use Limited Discrepancy Search in combination with distance matrix a priori data and binary elite selection range.

2.4.3.1 Limited Out-link Search

As mentioned before, Limited Out-link Search (LOS) guidance can only be used in combination with binary elite selection. For simplicity we will assume that the elite expansions are edges. Given the elite edges of a state, LOS limits the DP search to only use these elite edges for expansion. It is closely related to removing edges from the instance. However, LOS removes out-links per state, instead of edges of the instance (which would remove them for all states). In general this removes any guarantee for optimality. Out-link Restricted DP (Section 2.3.2) fits in this framework, where the elite edges are the L shortest out-links of that state.

Consider using edge scores to determine elite edges for the Limited Out-link Search. Suppose we have obtained the edge counters from the Restricted DP state space and determined the edge scores. The selected percentile level τ in turn determines the threshold $\rho \in [0, 1]$. All edges with a score equal to or higher than the threshold ρ are classified as elite. Although we have already found a solution, performing another Restricted DP guided by these elite edges can result in better solutions. We can guide DP by limiting the allowed state expansions (using LOS): only elite edges can be used (which are fixed for all stages).

In a way, this is a two phase DP approach as two sequential DP searches are performed. The first Restricted DP extracts information on elite edges and the second Restricted DP is guided by these elite parts. We call this method the Two-Phase DP and apply it to the TSP in Chapter 3.

One problem that can arise from limiting the out-links is that a state cannot be expanded by any elite edge. Two obvious solutions are: do not expand that state or expand the state as if all edges are elites.

The first solution is more strict and can lead to infeasibility. We suggest to use the second solution. When a state has no feasible elite expansions, we do not have a preferred way of expanding that state. It does not necessarily mean that the state is bad in quality, as the counters are state independent values aggregated from the DP state space. Similar problems can occur when using feasible parts as data. Thus, expanding that state as if we are not guiding the DP method seems to be a logical approach. An added benefit to this approach is that this approach always returns a feasible tour (if one exists). See Algorithm 2.4.3 for details of the Guided Restricted Dynamic Programming, which we have just described.

2.4.3.2 Limited Discrepancy Search

Restricting the DP search purely to elite edges can lead to infeasibility issues and can be too strict. Since elite edges are determined approximately, an edge classified as normal can turn out to be critical for good solutions. Limited Discrepancy Search (LDS) allows the search to deviate slightly from elite edges. We present the method for the binary elite selection range.

At each state expansion we have normal and elite edges. We allow DP to use both, but keep track of the number of deviations from elite edges. If a partial solution (a path) exceeds a maximum allowed discrepancy, it is discarded. Thus, setting the maximum discrepancy to zero results in the Limited Out-link Search without feasibility correction. Setting it equal to the maximum size of a solution ($|V|$ suffices) results in an unrestricted search: the elite edges have no effect.

LDS works in a similar way for discrete and continuous selection ranges. Instead of monitoring the number of deviations from elite edges, we keep track of discrepancy penalties. Each expansion has a discrepancy penalty based on a predefined function and the eliteness level of the edge. Every expansion changes (e.g. increments) the total discrepancy penalty of the path so far. If the total exceeds the maximum allowed discrepancy the path is discarded. Usually one would not penalise the most elite class of expansions (see for instance the binary selection case).

Again, feasibility issues do arise when applying LDS to the Travelling Salesman Problem. In [13] this is solved by increasing the maximum discrepancy whenever infeasibility occurs. A similar alteration as with the Limited Out-link Search contradicts the whole idea of LDS, unless this is done as a final resort (e.g. reset the total discrepancy of all paths in the last reached stage).

Since Restricted DP must select up to B states (or solutions) to expand in each stage, we can give preference to solutions with the lowest discrepancy followed by smallest travelled distance. Hence, another way to maintain feasibility is to ignore the maximum allowed discrepancy (allow all partial solutions), but give preference to lowest discrepancy states when selecting B states to expand. Multiple solutions per state can also be implemented with dominance rules, similarly to DP for other resource constrained optimisation problems.

We have not implemented Limited Discrepancy Search for the Travelling Salesman Problem. Further research must be done to see which feasibility correction works well.

2.4.3.3 Iterative Guidance

Since Restricted Dynamic Programming can serve as information source (e.g. feasible parts or edge counters), one can implement an iterative guidance method, where the elite edges are updated after each Restricted DP execution. This can be seen as a generalisation of the Two-Phase DP method. However, care must be taken to prevent self-fulfilling prophecies: since the DP search is guided to use elite edges, these often appear to be the best edges from the Guided DP information. Therefore, any iterative guidance would most likely stagnate and lead to little gain. Perhaps ideas from Tabu Search can strengthen iterative guidance. Nevertheless, we will not consider applying iterative guidance.

Algorithm 2.4.1 Full Dynamic Programming for TSP with Edge Counters

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$
Output: minimum travelled distance of Hamiltonian cycle if one exists

```
 1: procedure TSP-FULL-DP(input)
 2:     initialise objective state space labels: $\mathcal{L}_o(S, v) = \infty$ for all $S \subseteq V$ and $v \in S$
 3:     initialise path state space labels: $\mathcal{L}_p(S, v) = \emptyset$ for all $S \subseteq V$ and $v \in S$
 4:     initialise edge counters to zero: $C_{exp}(v_i, v_j) = C_{use}(v_i, v_j) = 0$ for all $(v_i, v_j) \in E$
 5:     set: $\hat{V} = V \setminus \{s\}$

 6:     for all nodes $v_j \in \{v \in \hat{V} : (s, v) \in E\}$ do
 7:         increment edge expansion counter: $++C_{exp}(s, v_j)$
 8:         increment edge usage counter: $++C_{use}(s, v_j)$
 9:         new objective state space label: $\mathcal{L}_o(\{v_j\}, v_j) = d(s, v_j)$
10:         new path state space label: $\mathcal{L}_p(\{v_j\}, v_j) = s$

11:     for all stages $\sigma = 1, \dots, |\hat{V}| - 1$ do
12:         for all subsets $S_i \in \{S \subset \hat{V} : |S| = \sigma\}$ do
13:             for all nodes $v_i \in S_i$ do
14:                 for all nodes $v_j \in \{v \in \hat{V} \setminus S_i : (v_i, v) \in E\}$ do
15:                     increment edge expansion counter: $++C_{exp}(v_i, v_j)$
16:                     if $\mathcal{L}_o(S_i, v_i) + d(v_i, v_j) < \mathcal{L}_o(S_i \cup \{v_j\}, v_j)$ then
17:                         if $\mathcal{L}_p(S_i \cup \{v_j\}, v_j) \neq \emptyset$ then
18:                             decrement old edge usage counter: $--C_{use}(\mathcal{L}_p(S_i \cup \{v_j\}, v_j), v_j)$
19:                         increment new edge usage counter: $++C_{use}(v_i, v_j)$
20:                         new objective state space label: $\mathcal{L}_o(S_i \cup \{v_j\}, v_j) = \mathcal{L}_o(S_i, v_i) + d(v_i, v_j)$
21:                         new path state space label: $\mathcal{L}_p(S_i \cup \{v_j\}, v_j) = v_i$

22:     for all nodes $v_i \in \{v \in \hat{V} : (v, s) \in E\}$ do
23:         increment edge expansion counter: $++C_{exp}(v_i, s)$
24:         if $\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) < \mathcal{L}_o(V, s)$ then
25:             if $\mathcal{L}_p(V, s) \neq \emptyset$ then
26:                 decrement old edge usage counter: $--C_{use}(\mathcal{L}_p(V, s), s)$
27:             increment new edge usage counter: $++C_{use}(v_i, s)$
28:             new objective state space label: $\mathcal{L}_o(V, s) = \mathcal{L}_o(\hat{V}, v_i) + d(v_i, s)$
29:             new path state space label: $\mathcal{L}_p(V, s) = v_i$

30:     return $\mathcal{L}_o(V, s)$
```

Algorithm 2.4.2 Restricted Dynamic Programming for TSP with Bias Adjusted Edge Counters

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$, state beam width $B \in \mathbb{N}_{>0}$, out-link number $L \in \mathbb{N}_{>0}$
Output: travelled distance of Hamiltonian cycle if one exists

```
1: procedure TSP-State and Out-link Restricted-DP(input)
2:     initialise objective state space labels: $\mathcal{L}_o(S, v) = \infty$ for all $S \subseteq V$ and $v \in S$
3:     initialise path state space labels: $\mathcal{L}_p(S, v) = \emptyset$ for all $S \subseteq V$ and $v \in S$
4:     initialise in-link state space labels: $\mathcal{L}_{in}(S, v) = 0$ for all $S \subseteq V$ and $v \in S$
5:     initialise edge counters to zero: $C_{exp}(v_i, v_j) = C_{use}(v_i, v_j) = 0$ for all $(v_i, v_j) \in E$
6:     set: $\hat{V} = V \setminus \{s\}$

7:     for all nodes $v_j$ in Nodes to Visit$((\emptyset, s), L)$ do
8:         new objective state space label: $\mathcal{L}_o(\{v_j\}, v_j) = d(s, v_j)$
9:         new path state space label: $\mathcal{L}_p(\{v_j\}, v_j) = s$

10:    for all stages $\sigma = 1, \dots, |\hat{V}| - 1$ do
11:        for all states $(S_i, v_i)$ in States to Expand$(\sigma, B)$ do
12:            for all nodes $v_j$ in Nodes to Visit$((S_i, v_i), L)$ do
13:                increment in-link label: $++\,\mathcal{L}_{in}(S_i \cup \{v_j\}, v_j)$
14:                if $\mathcal{L}_{in}(S_i \cup \{v_j\}, v_j) == 2$ then
15:                    increment old edge expansion counter: $++\,C_{exp}(\mathcal{L}_p(S_i \cup \{v_j\}, v_j), v_j)$
16:                    increment old edge usage counter: $++\,C_{use}(\mathcal{L}_p(S_i \cup \{v_j\}, v_j), v_j)$
17:                if $\mathcal{L}_{in}(S_i \cup \{v_j\}, v_j) \geq 2$ then
18:                    increment edge expansion counter: $++\,C_{exp}(v_i, v_j)$
19:                if $\mathcal{L}_o(S_i, v_i) + d(v_i, v_j) < \mathcal{L}_o(S_i \cup \{v_j\}, v_j)$ then
20:                    if $\mathcal{L}_p(S_i \cup \{v_j\}, v_j) \neq \emptyset$ then
21:                        decrement old edge usage counter: $--\,C_{use}(\mathcal{L}_p(S_i \cup \{v_j\}, v_j), v_j)$
22:                        increment new edge usage counter: $++\,C_{use}(v_i, v_j)$
23:                    new objective state space label: $\mathcal{L}_o(S_i \cup \{v_j\}, v_j) = \mathcal{L}_o(S_i, v_i) + d(v_i, v_j)$
24:                    new path state space label: $\mathcal{L}_p(S_i \cup \{v_j\}, v_j) = v_i$

25:    for all states $(\hat{V}, v_i)$ in States to Expand$(|\hat{V}|, B)$ do
26:        increment in-link label: $++\,\mathcal{L}_{in}(V, s)$
27:        if $\mathcal{L}_{in}(V, s) == 2$ then
28:            increment old edge expansion counter: $++\,C_{exp}(\mathcal{L}_p(V, s), s)$
29:            increment old edge usage counter: $++\,C_{use}(\mathcal{L}_p(V, s), s)$
30:        if $\mathcal{L}_{in}(V, s) \geq 2$ then
31:            increment edge expansion counter: $++\,C_{exp}(v_i, s)$
32:        if $\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) < \mathcal{L}_o(V, s)$ then
33:            if $\mathcal{L}_p(V, s) \neq \emptyset$ then
34:                decrement old edge usage counter: $--\,C_{use}(\mathcal{L}_p(V, s), s)$
35:                increment new edge usage counter: $++\,C_{use}(v_i, s)$
36:            new objective state space label: $\mathcal{L}_o(V, s) = \mathcal{L}_o(\hat{V}, v_i) + d(v_i, s)$
37:            new path state space label: $\mathcal{L}_p(V, s) = v_i$

38:    return $\mathcal{L}_o(V, s)$
```

Algorithm 2.4.3 LOS Guided Restricted Dynamic Programming Heuristic for TSP

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$, state beam width $B \in \mathbb{N}_{>0}$, out-link number $L \in \mathbb{N}_{>0}$, subset of elite edges $H \subseteq E$

Output: travelled distance of Hamiltonian cycle if one exists

```
1:  procedure TSP-LOS Guided Restricted-DP(input)
2:      initialise objective state space labels: $\mathcal{L}_o(S, v) = \infty$ for all $S \subseteq V$ and $v \in S$
3:      set: $\hat{V} = V \setminus \{s\}$

4:      for all nodes $v_j$ in Elite Nodes to Visit$((\emptyset, s), L, H)$ do
5:          new objective state space label: $\mathcal{L}_o(\{v_j\}, v_j) = d(s, v_j)$

6:      for all stages $\sigma = 1, \dots, |\hat{V}| - 1$ do
7:          for all states $(S_i, v_i)$ in States to Expand$(\sigma, B)$ do
8:              for all nodes $v_j$ in Elite Nodes to Visit$((S_i, v_i), L, H)$ do
9:                  if $\mathcal{L}_o(S_i, v_i) + d(v_i, v_j) < \mathcal{L}_o(S_i \cup \{v_j\}, v_j)$ then
10:                     new objective state space label: $\mathcal{L}_o(S_i \cup \{v_j\}, v_j) = \mathcal{L}_o(S_i, v_i) + d(v_i, v_j)$

11:     for all states $(\hat{V}, v_i)$ in States to Expand$(|\hat{V}|, B)$ do
12:         if $\mathcal{L}_o(\hat{V}, v_i) + d(v_i, s) < \mathcal{L}_o(V, s)$ then
13:             new objective state space label: $\mathcal{L}_o(V, s) = \mathcal{L}_o(\hat{V}, v_i) + d(v_i, s)$

14:     return $\mathcal{L}_o(V, s)$


15: procedure Elite Nodes to Visit(state $(S_i, v_i)$, out-link number $L$, elite edges $H$)
16:     if any adjacent edge is elite, $\{v \in \hat{V} \setminus S_i : (v_i, v) \in H\} \neq \emptyset$, then
17:         for all nodes $v_j \in \{v \in \hat{V} \setminus S_i : (v_i, v) \in H\}$ do
18:             rank edge $(v_i, v_j)$
19:             insert node $v_j$ in list according to rank of $(v_i, v_j)$
20:     else
21:         for all nodes $v_j \in \{v \in \hat{V} \setminus S_i : (v_i, v) \in E\}$ do
22:             rank edge $(v_i, v_j)$
23:             insert node $v_j$ in list according to rank of $(v_i, v_j)$

24:     return up to $L$ highest ranked elements from list


25: procedure States to Expand(stage $\sigma$, state beam $B$)
26:     for all subsets $S_i \in \{S \subseteq \hat{V} : |S| = \sigma\}$ do
27:         for all nodes $v_i \in S_i$ do
28:             rank state $(S_i, v_i)$
29:             insert state in list according to rank of $(S_i, v_i)$

30:     return up to $B$ highest ranked elements from list
```

2.5 Conclusion

To be able to develop a Guided Dynamic Programming method we have first discussed the fundamental properties of Dynamic Programming and the exact DP method for the Travelling Salesman Problem. In addition, several improvements for the exact DP method for the TSP found in the literature are mentioned.

The disadvantage of the (exact) Bellman-Held-Karp DP method is its exponential running time. By restricting the number of states that are expanded and the number of out-links used when expanding, we obtain a pseudopolynomial heuristic referred to as Restricted Dynamic Programming in the literature. We have suggested a local improvement step for the Restricted DP method, but (numerical) studies have to be performed to investigate the effectiveness of this improvement step.

The Restricted DP method suffers from near-sightedness caused by the (often) greedy selection of states and out-links. To counteract this shortcoming (or attempt to), we guide the DP search by limiting the allowed out-links to a predefined set of edges per state. This leads to the Guided Restricted Dynamic Programming method, based on Restricted DP.

The Guided DP method fits in a general framework called the Guided Dynamic Programming Framework. It presents a method for Guided DP consisting of three steps:

- information extraction,
- elite expansion selection,
- Dynamic Programming guidance,

which are discussed in detail.

The information extraction step, the first step, collects information which will be used to classify elite edges. We discern three sources of information:

- a priori data,
- feasible parts data,
- meta-data.

A priori data is information on the instance, for example, the distance matrix. Feasible parts data is extracted from (parts of) feasible solutions, for instance, randomly constructed solutions. Finally, meta-data is information obtained from solution approaches and is often a by-product. Examples are pheromone weights of edges in Ant Colony Optimisation and the introduced edge counters from the DP state space.

The edge counters from the DP state space count the number of times an edge is evaluated for an expansion and the number of times an edge is used in locally optimal paths in the DP state space. By dividing the usage counter by the expansion counter, we get an indication of the attractiveness of the edge. Several theoretical properties for these counters for the exact DP state space are derived in the appendix.

The second step is the selection of elite edges. We have illustrated the following options:

- state independent selection,
- state dependent selection,

combined with one of the following:

- binary selection range,
- discrete selection range,
- continuous selection range.

The state dependency determines if elite edges are fixed for all states or are allowed to vary. The selection ranges correspond to quantifying the level of eliteness of an edge. We focus on (fixed) state independent binary elite edges: edges are either elite or normal for all states.

The final step is guiding the DP search. Two types of guidance are presented:

- Limited Out-link Search (LOS),
- Limited Discrepancy Search (LDS).

LOS only allows expansions with elite edges and is more restrictive than LDS, since LDS allows a slight deviation from elite edges. LDS has been used before in the literature. Due to its simplicity, we focus on LOS. Depending on the optimisation problem at hand, both guidance methods need a feasibility correction to guarantee the construction of a feasible solution.

To conclude, we have presented a general framework to guide (Restricted) Dynamic Programming. Restricted Dynamic Programming fits in the Guided DP Framework: the distance matrix is used as a priori data, combined with a state dependent binary elite edge selection and Limited Out-link Search guidance. When discussing the framework, we have presented alternative methods. In the following two chapters we apply the framework to solve the Travelling Salesman Problem and the Vehicle Routing Problem using several types of Guided DP. Restricted DP will be used as a benchmark reference.

Chapter 3

Application to the Travelling Salesman Problem

3.1 Introduction

In this chapter we investigate the effectiveness of the Guided Dynamic Programming Framework when applied to the Travelling Salesman Problem. We present three methods based on different information extraction approaches. To evaluate the performance and the effect of parameter settings we use two data sets of randomly generated instances. Based on those results we select good parameters and solve a selection of literature benchmark instances.

In Section 3.2 we describe the general methodology and the used instances. The first method, based on DP edge counters, is presented for optimal and realistic conditions in Sections 3.3 and 3.4, respectively. The second method uses parts of random solutions and is shown in Section 3.5. The final method corresponds to the State and Out-link Restricted DP from the literature and is discussed in Section 3.6. All three solution methods are applied to the selected literature benchmarks in Section 3.7.

3.2 General Methodology

The Restricted DP method for the Travelling Salesman Problem can be guided in several ways. We will evaluate the following information extraction methods separately:

- a priori data from the distance matrix,
- feasible parts data from random solutions,
- meta-data from DP edge counters.

These extraction methods are combined with a binary elite selection range and Limited Out-link Search guidance. The details of each approach are mentioned in their separate sections. Note that meta-data is treated first and a priori data last (in decreasing order of complexity of information extraction). We call these solution methods the prototypes. Our reference solution method will be the State Restricted DP (see Section 2.3.1) and corresponds to no guidance. Any difference between the reference and the prototypes is therefore caused by the applied guidance. We do not compare the results with other approaches from the literature.

The (Guided) Restricted DP method of the reference and the prototypes are evaluated with the same state beam range $B \in \{250, 300, 350, 400, 450, 500, 750, 1000, 1500, 2000, 2500\}$. For the a priori data prototype, equal to State and Out-link Restricted DP, we consider the number of out-links $L \in \{2, 3, 5, 10\}$. The feasible parts and meta-data prototypes both need a percentile level for the elite edge selection. Thereto, we evaluate the performance for the following percentile levels: $\tau \in \{0.1, 0.15, 0.20, \ldots, 0.95\}$. Recall that with $\tau = 0.95$ approximately 5% of the edges are selected as elites and similar with the other values.

The methods are tested using the following instances:

- 50 randomly generated Euclidean TSP instances with 15 customers,
- 50 randomly generated Euclidean TSP instances with 25 customers,
- 12 literature benchmark Euclidean TSP instances of varying sizes.

The random instances are used to illustrate the effect of different parameter values and of the increase in the size of the instance. From the analysis we determine a fixed setting for the parameters and solve selected literature benchmark instances. More details are given below.

The described methods do not depend on the type of TSP, asymmetric instances could also be used. We will restrict our study to Euclidean random instances and Euclidean literature benchmarks, because these are the most basic TSP classes. Note that for Euclidean TSP a polynomial-time approximation scheme (PTAS) exists ([2]).

3.2.1 Instances

For the numerical evaluation three sets of instances are available, see also Appendix C.2. The first set contains 50 randomly generated Euclidean instances with 15 customers. These instances are small enough to be solved exactly with DP, which allows us to evaluate performance by the relative gap from optimality. Furthermore, it allows us to use the (exact) Full DP edge counters to show the effectiveness of this information source. The main goal of the first set of instances is to show the effectiveness of the prototypes and the effect of the parameters.

The second set consists of larger instances: it contains 50 randomly generated Euclidean instances with 25 customers. The main goal of this set is to show the effect of the increase in the size of the instance. Again, all three solution prototypes are tested for different parameter settings. For each instance we have performed a State Restricted DP with state beam $B = 10^6$. We pretend that the resulting objective value is the optimal value to be able to benchmark the results.

Each of the random instances sets has two type of instances: with and without clusters (that is, grouped or randomly spread customer locations, respectively). Both types are equally represented, each with 25 instances. The generation of the instances in described in more detail below in Section 3.2.2.

The final set is a collection of benchmarks commonly used in the literature. We use 12 of the Euclidean benchmark instances from the TSPLIB library[1] that have less than 100 customers. The restriction to small instances is purely to keep the computational time within reasonable bounds. Note that instances without Euclidean data format are excluded. The optimal objective values are given for each instance, allowing us to evaluate the prototypes for literature benchmarks.

3.2.2 Generation of Random Instances

As mentioned, there are two types of generated instances: with one cluster or with multiple clusters. For the generation of random instances we have used a `Matlab Central File Exchange` program called 'Generate Data for Clustering' by Nuno Fachada, version August 2012. The 2D data points are created along straight lines with normally distributed slopes. Each line represents a cluster. The length of these lines is also normally distributed. The data points are randomly divided among the clusters based on the normal distribution. Note that empty clusters are prevented by reallocating points to clusters. The points are generated around these lines, where the horizontal and vertical deviation is again normally distributed. Finally, the cluster centres are uniformly spaced from each other.

As an extra step we have shifted the data points to the positive quadrant and rounded the coordinates to integers. Since these randomly generated instances are often relatively easy to solve, we have filtered out the easy instances. If a State Restricted DP with $B = 1000$ has the same objective as the best known solution (optimal or obtained with a beam of 10^6), it is discarded. This is a simple measure, as even the smallest positive gap from optimality is sufficient to accept the instance.

For instances with 15 customers and multiple clusters, we have set the number of clusters to be uniformly distributed in $\{2, 3, 4, 5\}$. For 25 customers it is uniformly in $\{5, 6, 7, 8\}$. The `Matlab` program is called with the following input:

```
generateData(0, 100, nofClusters, 15+5*round(rand), 15+5*round(rand),...
            5+10*round(rand), 5, 5, instanceSize)
```

For the exact meaning of the input, see the relevant documentation[2].

[1]See `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/` for more information.

[2]See `http://www.mathworks.com/matlabcentral/fileexchange/37435-generate-data-for-clustering` for further information.

3.3 Two-Phase DP Prototype under Optimal Conditions

To determine whether the edge counters derived from the DP state space indeed contain valuable information, we consider the case that these counter are from the Full DP method. Furthermore, we neglect the time needed to perform Full DP. Simply put, this is cheating. However, if the DP prototype performs badly under these optimal conditions, there is no good reason for it to work under realistic conditions.

3.3.1 Methodology

As mentioned before, the DP prototype uses DP edge counters as meta-data. In this case, these are derived from the (exact) Full DP as described in Section 2.4.1.4. Therefore, no bias correction is needed. From these counters we derive edge scores as in Section 2.4.1.5. By selecting a percentile level $\tau \in [0, 1]$ we determine the corresponding threshold percentile value $\rho \in [0, 1]$, such that a fraction of τ edge scores are below ρ. The edges with a score above ρ are classified as elite edges. For details, see Algorithm 3.3.2. Since the edge counters are from the Full DP method, it holds that $E^* = \hat{E}$ in the algorithm.

We guide the Restricted DP method by using the Limited Out-link Search guidance (with feasibility modification). To recall the method: if a state can be expanded with elite edges, only those edges are used for expansion. Otherwise, all available edges are expanded. See also Algorithm 3.3.1 (note that $B_1 = \infty$ and $L_1 = |\hat{V}|$ for exact edge counters).

Because the elite edges are determined from the DP state space, this prototype can be viewed as a Two-Phase DP method. The first phase corresponds to the DP approach for the information extraction, the second phase to Guided DP. We will often refer to these phases as Phase 1 and Phase 2.

One might wonder why we consider the edge counters obtained from Full DP to solve the problem (again) with a heuristic. Clearly, we already know the optimal solution after Full DP. As mentioned, this approach allows us to show the potential of the Two-Phase DP prototype. In practice, we cannot solve the problem with Full DP (due to the exponential running time) and must resort to heuristics. We can perform a Restricted DP as an exploration phase, obtaining approximate edge counters. With these approximate counters, we perform Guided Restricted DP to (hopefully) improve our solution. Unfortunately, issues arise when dealing with approximate counters, as seen in Section 2.4.1.6. We will discuss this method in more detail in Section 3.4.

Performance is measured in the required computational time and the resulting relative gap from optimality. The relative gap is defined by the difference between the resulting objective value and the optimal (or best known) value, normalised by the optimal (or best known) value. Note that in this case, we ignore the time required to complete Full DP. Hence, the computational times of the reference and prototype cannot be used to compare performance.

Finally, we need to set the parameters: the state beam and the percentile level. We evaluate the prototype for the following state beams

$$B_2 \in \{250, 300, 350, 400, 450, 500, 750, 1000, 1500, 2000, 2500\}$$

and for each of the following percentile levels:

$$\tau \in \{0.1, 0.15, 0.20, \ldots, 0.95\}.$$

For instance with $\tau = 0.9$, approximately the 10% highest scoring edges are selected as elite edges. The number of out-links is unrestricted, $L_2 = |\hat{V}|$.

3.3.2 Numerical Results for 15 Customers

In this section we illustrate the difference in performance of State Restricted DP and Guided Restricted DP, using the exact edge counters obtained from the Full DP method. Although having the exact counters is an artificial situation, it will show the best possible performance increase of the Guided Restricted DP method. The results in this section are derived by solving the 50 randomly generated TSP instances with 15 customers, as described in Section 3.2.1. We compare the performance of the reference and the prototype for a range of state beam widths and different percentile levels, see Figures C.3.1 to C.3.6.

The relative optimality gaps of both methods with respect to beam width are shown in Figures C.3.1 and C.3.2. The horizontal axis corresponds to the beam width, the vertical axis to the relative gap from the optimal solution. The (normal) State Restricted DP reference is shown in blue and is our reference. The Guided DP prototype is given in red.

Dashed lines are the minimum, respectively the maximum, relative gap obtained for the instances. Circles are the average relative gaps over all instances for fixed beam width. Stars are the medians of the relative gap for fixed beam width. The coloured range indicates the spread of the relative gap, showing the range between the 12.5% and 87.5% percentiles of the relative gap. Hence, the coloured range shows the relative gap of 75% of the instances (centred around the median).

Multiple settings for the percentile level are evaluated and shown in each figure. Figure C.3.1 gives an overview of a lower range of percentile levels ranging from 0.1 to 0.5, whereas Figure C.3.2 shows the higher range (0.55 to 0.95). Figures C.3.3 and C.3.4 depict the relative gap with respect to the computational time. All lines, symbols and colours have the same meaning as the previous two figures.

Likewise, Figures C.3.5 and C.3.6 show the relative gap with respect to the average computational time. The average time is taken over all instances for fixed beam width. It holds that a larger average time corresponds to a larger beam width.

The average, median and the coloured range are suitable for comparison between the two methods, as the maximum is not a very robust measure. We would prefer that the prototype outperforms the reference. Simply put, this would be the case if the red symbols are lower than the blue symbols, and the red coloured range is more narrow and lower than the blue one.

The most obvious result is that the performance of Guided Restricted DP is highly dependent on the percentile level parameter. In particular, high percentile levels (0.75 to 0.95) give the best performance. We can conclude that classifying about 5% of the edges as elites gives a significant improvement in performance (with respect to beam width) and in efficiency (with respect to computational time).

From Figure C.3.2 we see that the quality of Guided Restricted DP does not monotonically increase with respect to the percentile level (see percentile level 0.8). Nevertheless, high percentile levels seem to work better. In most cases, both the prototype and the reference seem to stagnate for larger beam widths (within the selected range). This is a common feature of Restricted DP, where the short-sightedness of the state selection takes its toll.

Figures C.3.3 and C.3.4 show that a great improvement in computational time is achieved, even with relative low percentile levels. Recall that the time for calculating the exact edge counters is not taken into account, but these figures do show the reduced computational time of Phase 2 (Guided DP). Also notice the increase in computational time for percentile level 0.95, with respect to percentile level 0.90. This is due to fewer elite edges (5% versus 10% approximately) and the feasibility adjustment: if no elite out-links are available, all out-links are expanded.

Overall, these results show that the (exact) edge counters contain valuable information which can be used to guide the Restricted DP method. Since instances with 15 customers are very small and can be solved exactly, we continue to see if this conclusion still holds for larger instances.

3.3.3 Numerical Results for 25 Customers

The only difference between this case study and the previous one, is the size of the instances (now 25 customers are visited). The visualisation of the results is still the same, but not all results are shown. Figure C.3.7 shows the relative optimality gap with respect to the beam width, Figure C.3.8 the required computational time and Figure C.3.9 the average efficiency.

It is clear from the relative optimality gaps that both methods perform worse for 25 customers. The main cause is the state beam widths. For 15 customers the maximum number of states during one DP stage is around $52 \cdot 10^3$, for 25 customers it is approximately $676 \cdot 10^5$. Therefore, the relative fraction of states in the beam is much smaller with 25 customers.

From Figure C.3.7 we see that improvements are made for percentile levels of 0.7 and higher, in particular for 0.9. Note that good percentile levels are not a guarantee for improvement, see the maximum relative gap for percentile level 0.8. Furthermore, the computational time required for Guided DP is smaller than the State Restricted DP method. Although the last figure (Figure C.3.9) shows a significant efficiency increase, keep in mind that the calculation of the exact counters is not taken into account.

Similar to the instances with 15 customers, the DP edge counters can be used to guide Restricted DP to improve the results. However, the increase in the size of the instance significantly affects the quality of the solutions for the fixed range of beam states (in a negative way). Guiding the DP search can therefore contribute more if instance size increases and beam width stays the same.

Algorithm 3.3.1 Two-Phase Guided Restricted Dynamic Programming for TSP

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$,
Phase 1 state beam width $B_1 \in \mathbb{N}_{>0}$, Phase 1 out-link number $L_1 \in \mathbb{N}_{>0}$,
Phase 2 state beam width $B_2 \in \mathbb{N}_{>0}$, Phase 2 out-link number $L_2 \in \mathbb{N}_{>0}$,
percentile level $\tau \in [0, 1]$
Output: travelled distance of Hamiltonian cycle if one exists

```
1: procedure TSP-Two-Phase Guided Restricted-DP(input)
2:     run: TSP-State and Out-link Restricted-DP($G, s, d, B_1, L_1$)
           ▷ giving edge expansion counter $C_{exp}$ and edge usage counter $C_{use}$
3:     set: $H$ = Elite Edge Selection based on Edge Counters($G, s, C_{exp}, C_{use}, \tau$)
4:     return TSP-LOS Guided Restricted-DP($G, s, d, B_2, L_2, H$)
```

Algorithm 3.3.2 Elite Edge Selection based on Edge Counters

Input: directed graph $G = (V, E)$, starting node $s \in V$,
edge expansion counter $C_{exp} : E \to \mathbb{N}$, edge usage counter $C_{use} : E \to \mathbb{N}$,
percentile level $\tau \in [0, 1]$
Output: set $H \subseteq E$ of elite edges

```
1: procedure Elite Edge Selection based on Edge Counters(input)
2:     remove edges connected to starting node: $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in (V \setminus \{s\})\}$
3:     remove unevaluated edges: $E^* = \{e \in \hat{E} : C_{exp}(e) > 0\}$
4:     determine edge scores: $A(e) = C_{use}(e)/C_{exp}(e) \in [0, 1]$    for all $e \in E^*$
5:     determine threshold: $\rho^* = \min\{\rho \in [0, 1] : |\{e \in E^* : A(e) \leq \rho\}|/|E^*| \geq \tau\}$
6:     classify elite edges: $H = \{e \in E^* : A(e) \geq \rho^*\}$
7:     return $H$
```

3.4 Two-Phase DP Prototype under Realistic Conditions

The performance of Guided Dynamic Programming shown in Section 3.3 is artificial as we normally do not have the edge counters of the Full DP method. In this section we will describe the Two-Phase Guided Restricted DP method, which can be applied in practice (it has a pseudo-polynomial running time). Instead of using the exact edge counters of Full DP, we use the counters derived from Restricted DP. Although these counters are approximations of the exact counters and are biased (see Section 2.4.1.6), they should be able to discern the attractiveness of edges to some degree.

3.4.1 Methodology

The Two-Phase Guided DP method starts with an exploration phase, solving the problem with the State Restricted DP and keeping track of the edge counters. Note that we apply the bias correction heuristic from Section 2.4.1.6: the counters are only updated if there is competition between in-links. Based on these counters, we select the elite edges the same way as under optimal conditions (Algorithm 3.3.2). Since the first phase restricts the number of states, it can happen that $E^* \neq \hat{E}$ in the algorithm. The second phase performs the Guided Restricted DP method using the elite edges, again using the same approach as under optimal conditions. To be able to discern the effect of guidance, we discard the DP solution of Phase 1 (which might actually be better than Phase 2). For an overview of the method, see Algorithm 3.3.1.

The parameters are set to the same values or ranges as under the optimal conditions case. However, we need to specify the state beam and out-link number of the first Restricted DP (the first phase). Since we want as much competition between in-links, we do not restrict the out-links ($L_1 = |\hat{V}|$). The state beam is evaluated for three different values: $B_1 \in \{250, 500, 1000\}$. The subscript indicates that these parameters hold for that specific phase. Thus, we also have

$$L_2 = |\hat{V}|, \qquad B_2 \in \{250, 300, 350, 400, 450, 500, 750, 1000, 1500, 2000, 2500\}$$

for out-link and state beam parameters of the second phase.

3.4.2 Numerical Results for 15 Customers

As in previous section we compare the performance of the Two-Phase Restricted DP method against the normal Restricted DP. The lay-out of the figures is the same as in Section 3.3. Not all results are shown. For instance, Figure C.3.10 shows the performance of both methods with respect to the beam width. On the horizontal axis the beam width is shown and on the vertical axis the relative gap from the optimal solution. Blue lines and symbols depict the normal Restricted DP (the reference) and red the Two-Phase DP method. The green line corresponds to Phase 1 of the Two-Phase DP method. Often, the green line is hardly visible, because Phase 1 only performs one DP. It is not very important, as it is equal to the reference with corresponding beam width.

Figures C.3.10 to C.3.12 show the results when using a state beam width of 250 for Phase 1. The results for a beam of 500 and 1000 are shown in Figures C.3.13 and C.3.14, respectively.

From the figures for a Phase 1 beam of 250 we see that using approximate edge counters derived from Restricted DP with a small beam already leads to significant improvements for high percentile levels, both with regard to the optimality gap as the computational time. In contrast to the results under optimal conditions, the computational times of the prototype shown here are fully comparable with the reference: the required time for the first phase is included.

Again, we see that the performance is not monotone in the percentile level, see for instance percentile level 0.75. Furthermore, the prototype suffers more from stagnation. For high percentile levels state beams of 2000 and 2500 give the same quality of solutions (Figure C.3.10). If we compare stagnation with the 500 and 1000 Phase 1 beams, we see that larger Phase 1 beams result in less stagnation. We conclude that stagnation is primarily caused by wrong guidance due to the approximate (biased) edge counters.

Overall, a larger state beam for Phase 1 results in (slightly) better performance with respect to the optimality gap. However, computational time also increases, making it difficult for beams of 1000 (or larger) to compete with the reference. These results confirm the viability of the Two-Phase DP guidance under realistic conditions for small instances, for instance by taking a Phase 1 beam of 250.

3.4.3 Numerical Results for 25 Customers

Due to the exponential increase in the maximum number of states per phase when increasing the size of the instance, we expect that the reliability (the quality) of the approximate edge counters decreases. Therefore, the guidance would be of low quality if we do not adjust our parameters for the increase in customers. The numerical results confirm this reasoning. Figure C.3.15 gives the results for a Phase 1 beam of 250 and Figure C.3.16 of 500. Figures C.3.17 to C.3.19 correspond to a beam of 1000.

Even the largest Phase 1 beam (1000) has difficulty to improve the optimality gap with respect to the reference. In fact, worse performance is not uncommon. Improvements are mainly due to the decrease in computational time, see for instance a Phase 1 beam of 250 and percentile level of 0.9 (Figure C.3.15). Most results do not seem that promising.

We know that the guidance can be successful for 25 customers if we have high quality edge counters (for instance, with a beam of 10^6 see Figure C.3.7). The somewhat disappointing results under realistic conditions show the main weakness of the Two-Phase DP guidance method: unreliable edge counters due to insufficient competition and exploration of in-links. Because of the relatively small Phase 1 beam, there is little competition between in-links which leads to biased counters (as described in Section 2.4.1.6). Using larger beams for the exploration phase would lead to uncompetitive computational times.

Perhaps reliability of the counters can be improved by using theoretical properties as derived in Appendix B. We have not looked into this approach. Instead, we also consider a different source of information to select elite edges: parts of random solutions.

3.5 Random Solutions Prototype

Using meta-data derived from DP edge counters for Guided DP suffers from the curse of dimensionality of DP. The exploration phase cannot use large state beams without losing the ability to compete with respect to computational time. A correction based on theoretical results could potentially improve the method. However, we are not limited to use the meta-data from the DP state space. In this section we investigate the potential to use random solutions to guide DP. We perform the same analysis as with the Two-Phase DP method.

3.5.1 Methodology

The Random Solutions prototype is very simplistic. First, generate a large amount of random solutions for the problem at hand. Second, determine the (objective) value of each of these solutions and select the best 5% solutions. Then detect if these solutions share common edges and select those as elite edges. Finally, we guide the Restricted DP method using Limited Out-link Search guidance (the same way as Two-Phase DP). For an overview, see Algorithm 3.5.1.

Random solutions are generated in the following way: for each state expand one of the allowed out-links, which is chosen by a uniform distribution. Since we start with only one state (the starting node), we have exactly one state per DP stage. In case of the (unconstrained) TSP this comes down to a random permutation of the customers. Note that duplicates are possible, so a choice has to be made if these are removed. Since the probability of duplicates is small, we ignore the possibility of duplicates. For constrained TSP this assumption can be invalid. The generation of random solutions can be executed using parallel processing. As some steps in the DP method also allow for a parallel implementation, we have decided to use no (special) parallel computing.

We only keep the best 5% of these solutions, those with the smallest travelled distance. The elite edges are determined in a similar way as the Two-Phase DP method and is shown in Algorithm 3.5.2. We have chosen to ignore edges that are not used by any of the solutions (see E^*). The normalisation of the edge scores is unnecessary, but it makes the algorithm similar to one based on edge counters. The percentile levels are again set to $\tau \in \{0.1, 0.15, 0.20, \dots, 0.95\}$.

The parameters for the Guided DP method are the same as before: no limitation to the out-links, $L = |\hat{V}|$, and a range of state beams, $B \in \{250, 300, 350, 400, 450, 500, 750, 1000, 1500, 2000, 2500\}$. We evaluate the performance for 10^3, 10^4 and 10^5 number of random solutions generated. Notice that the number of random solutions will affect the quality of the elite edges, just as the Phase 1 state beam for the Two-Phase DP method.

3.5.2 Numerical Results for 15 Customers

The result for the Random Solutions Guided DP are summarised by the same kind of figures as Two-Phase DP. Again, not all results are shown. The figures are grouped depending on the number of generated solutions. Figure C.3.20 corresponds to 10^3 generated solutions, whereas Figures C.3.21 to C.3.23 are the results for 10^4 solutions. We have evaluated the Random Solutions Guidance for multiple random solution sets, giving similar results as those shown here.

Generating only 10^3 random solutions often results in (drastically) worse solutions than the reference. However, great computational time reductions are achieved with higher percentile levels. The only reasonably good parameter setting is a percentile level of 0.95, see Figure C.3.20. Using 10^3 solutions is not robust enough to consistently give improvements.

With 10^4 solutions we see a good performance increase for percentile levels 0.80 to 0.95 (Figure C.3.21). This approach is also efficient as seen in Figure C.3.23. Generating even more solutions (10^5) gives further improvement in the optimality gaps. However, the generation is implemented with a linear time algorithm, which results in computational times far larger than the reference.

The Two-Phase DP prototype with Phase 1 beam of 500 and the Random Solutions prototype with 10^4 solutions give solutions of similar quality for percentile levels 0.75 and higher. However, generating the random solutions takes less time, making the Random Solutions Guided DP more efficient. It remains to be seen if this comparison still holds for larger instances.

3.5.3 Numerical Results for 25 Customers

The Two-Phase DP prototype has difficulty to achieve improvements for larger instances, because of the exponential increase in the number of DP states. The number of possible solutions increases even faster (factorial increments) so we would expect even worse results for the Random Solutions Guidance approach. Figures C.3.24 to C.3.26 give the results with 10^5 random solutions for the case with 25 customers.

We can directly say that 10^3 solutions is insufficient for guiding DP. Increasing the number of solutions to 10^4 results in performance similar to the reference. No real improvements are made. But with 10^5 solutions the Random Solutions Guidance prototype achieves a reasonably stable improvement with high percentile levels, both in optimality gap as in computational time. Two-Phase DP was unable to provide similar results.

For instances with 15 customers using 10^4 solutions provided good performance. For 25 customers this must be increased to 10^5. If this pattern holds for larger instances, we soon run into similar problems of dimensionality with Random Solutions Guidance. Nevertheless, it shows that a simplistic search for elite edges certainly has potential.

Algorithm 3.5.1 Random Solutions Guided Restricted Dynamic Programming for TSP

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and distance function $d : E \to \mathbb{R}$,
number of random solutions $M \in \mathbb{N}$,
state beam width $B \in \mathbb{N}_{>0}$, out-link number $L \in \mathbb{N}_{>0}$,
percentile level $\tau \in [0, 1]$

Output: distance of Hamiltonian cycle if one exists

```
1: procedure TSP-Random Solutions Guided Restricted-DP(input)
2:     generate a set of $M$ random solutions: $\Theta$
3:     select best 5% solutions according to distance: $\Theta^*$
4:     set: $H$ = Elite Edge Selection based on Random Solutions$(G, s, \Theta^*, \tau)$
5:     return TSP-LOS Guided Restricted-DP$(G, s, d, B, L, H)$
```

Algorithm 3.5.2 Elite Edge Selection based on Random Solutions

Input: directed graph $G = (V, E)$, starting node $s \in V$,
set of random solutions Θ where each solution $\theta \in \Theta$ is a subset of edges $\theta \subseteq E$,
percentile level $\tau \in [0, 1]$

Output: set $H \subseteq E$ of elite edges

```
1: procedure Elite Edge Selection based on Random Solutions(input)
2:     count all used edges: $C_{rnd}(e) = |\{\theta \in \Theta : e \in \theta\}| \in \mathbb{N}$   for all $e \in E$
3:     remove edges connected to starting node: $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in (V \setminus \{s\})\}$
4:     remove unevaluated edges: $E^* = \{e \in \hat{E} : C_{rnd}(e) > 0\}$
5:     determine edge scores: $A(e) = C_{rnd}(e)/|\Theta| \in [0, 1]$   for all $e \in E^*$
6:     determine threshold: $\rho^* = \min\{\rho \in [0, 1] : |\{e \in E^* : A(e) \leq \rho\}|/|E^*| \geq \tau\}$
7:     classify elite edges: $H = \{e \in E^* : A(e) \geq \rho^*\}$
8:     return $H$
```

3.6 Out-link Prototype

Both meta-data from the DP state space and random solutions have difficulty to outperform the reference when the size of instances increases. In contrast, the selection of the shortest out-links (using the distance matrix) is hardly affected by an increase in size. In this section we will look into the performance of this greedy approach.

3.6.1 Methodology

The Out-link Prototype is exactly the same as the State and Out-link Restricted DP method from Section 2.3.2 (Algorithm 2.3.2). When expanding a state only up to L shortest out-links are expanded. In the framework this corresponds to using the distance matrix as a priori data, together with state dependent binary selection range and Limited Out-link Search. Recall that we evaluate the prototype for $L \in \{2, 3, 5, 10\}$ for both sets of random instances.

3.6.2 Numerical Results for 15 Customers

The representation of the results slightly deviates from the previous two prototypes: no percentile level needs to be selected for the Out-link prototype. Instead, multiple numbers of allowed out-links are depicted. See Figures C.3.27 to C.3.29 for the results. The symbols and colours have the same meaning: blue corresponds to the reference (State Restricted DP) and red to the prototype (State and Out-link Restricted DP).

A Restricted DP search with two out-links is almost equal to a greedy search (one out-link). The quality of the solution is worse than the reference, but the computational time is almost zero. Given that for 75% of the instances an optimality gap of less than 10% is achieved, it can serve as a good first estimate.

The other end of the out-link range, $L = 10$, is hardly worth the effort. The results are the same as the reference, with only a minimal gain in computational time. More interesting are 3 and 5 out-links. Using three out-links leads to stagnation for higher state beams. Five out-links suffers less from stagnation (within this beam range). Some improvements in the optimality gaps are achieved, but the main strength of the prototype is the computational time. Figure C.3.29 clearly shows that the prototype is superior in efficiency to the reference for 3 and 5 out-links.

It is interesting that the Out-link prototype has its strengths in computational time and not in improved optimality gaps, whereas the Two-Phase DP and Random Solutions prototypes generally improved both performance and efficiency.

3.6.3 Numerical Results for 25 Customers

The Out-link prototype gives some remarkable results for the instances with 25 customers. Since the number of customers has increased, it would seems a good approach to also increase the number of out-links. That is, instead of using 3 or 5 out-links, use 5 or 10 to get the best results. However, our results contradict this intuition, see Figures C.3.30 to C.3.32.

The results for two out-links ($L = 2$) may seem incorrect as the 'optimality' gap is negative. However, recall that we have solved these instances with State Restricted DP with a state beam of 10^6, not to optimality. We pretended that the resulting value was the optimum (as described in Section 3.2.1). It turns out that for one instance, C018-25, the Out-link prototype with two out-links improves the DP result from 321.6518 to 290.6854. To keep our results transparent, we have not adjusted the best known value of this instance.

Interestingly, the smaller numbers of out-links (2 and 3) improve the optimality gaps, whereas 5 and 10 do not. Since the size of the instances has increased, this is not what we expected. Nevertheless, the reduction in computational time is again the strength of the prototype, making all four out-link settings competitive.

From these results we can confirm that the Out-link prototype (State and Out-link Restricted DP) suffers less from an increase in the size of the instance. The quality of the solutions does not greatly improve with respect to the reference, but the reduction in computational time makes the prototype a viable choice.

3.7 Results for Literature Benchmarks

In this section we present the results for the selected literature benchmark instances, as described in Section 3.2.1. The results are not aggregated, but presented per benchmark instance as the sizes of the instances vary. This allows us to see if there is a connection between the performance and instance size.

3.7.1 Methodology

Each prototype follows the same methodology as described in their sections, except that we fix all parameters to single values. We have primarily considered the results for the 25 customers instances (the largest instances of our tests). The reference will use all out-links $L = |V|$ and a state beam $B = 2500$. For the Two-Phase DP prototype we set the Phase 1 beam to the maximum of our range, $B_1 = 1000$. Out-links are again not constrained. The Random Solutions prototype also uses the maximum number of solutions in our range: 10^5 solutions are generated. Both methods use a percentile level $\tau = 0.95$ as this level performed reasonably for most parameter settings. Finally, the number of out-links in the Out-link prototype is set to 3.

3.7.2 Numerical Results

The benchmarks have a size varying from 50 customers to 99 customers, far more than our random instances. Based on the results for 25 customers, we have expressed our doubts if the Two-Phase DP prototype and the Random Solutions prototype would be able to handle large instances. The results for the benchmarks seem to confirm this, see Tables C.4.1 and C.4.2.

Table C.4.1 presents the list of benchmark instances, their optimal value and the results of the reference and prototypes. The results both give the objective value and the relative gap from optimality. Values in bold are the best values achieved for that instance. Values in italics are better than the reference. Hence, we see that the reference gives the best results for 7 of the 12 instances (of which 3 are tied with the Out-link prototype). The average gap is shown in the last row. Thus, the reference has an average gap of 0.1811.

The Two-Phase DP prototype has the best objective values for 3 instances, which are significant improvements of the results of the reference. For one additional instance it improves the reference. However, the average gap is 0.2225, worse than the reference. The Random Solutions prototype has in general bad performance for the larger instances, improving the reference only once and having an average gap of 0.3038. Similar to the results of 25 customers, the Out-link prototype (State and Out-link Restricted DP) performs admirably with results always near to the reference and with 4 improvements. The average gap is 0.1784, lower than the reference.

We also have to take into account the required computational time, see Table C.4.2. The reference needs on average 27.1540 seconds to solve the benchmarks. Two-Phase DP is usually faster except for two instances with 99 customers. For the largest instances Phase 1 takes a long time. The average time is 25.2574 seconds, slightly less than the reference. Overall, the Two-Phase DP prototype performs worse for instances with more than 75 customers (with respect to relative optimality gap). The reduction in computational time is not in proportion to the loss in performance.

The Random Solutions prototype is always significantly faster with an average of 6.6805 seconds. The resulting objective values are not great, but these must be seen within context of the required computational time. Nevertheless, the Out-link prototype clearly outperforms all other methods. With better or equal objective values and and average computational time of 1.7258, it is by far the best choice for large instances. In fact, the relative reduction in computational time of the Out-link prototype only increases if the size increases.

We conclude that both the Two-Phase DP prototype and the Random Solutions prototype suffer when the instance size increases. Often a great reduction in computational time is achieved for Guided DP (without Phase 1), but the quality of the resulting solution is highly dependent on the quality of the elite edges. Successes of Two-Phase DP are not excluded, as it significantly reduces the gap for KROB100 (an instance with 99 customers). However based on these results, it does not perform well on average.

Since the quality of the elite edges is low for the Two-Phase DP and Random Solutions prototypes, we also evaluated different percentile level settings (in the range $\{0.05, 0.10, \ldots, 0.95\}$). The best results where achieved with percentile level $\tau = 0.1$: for Two-Phase DP an average gap of 0.2051 (34.4299 seconds) and for the Random Solutions prototype an average gap of 0.1802 (25.2188 seconds). With these settings, the Random Solutions Prototype has similar performance as the reference, with a small reduction in computational time. A percentile level this low shows that the information on elite edges is of low quality as many edges are set to elite (around 90%, which explains the increase in computational time).

3.8 Conclusion

We have applied three types of Guided DP to the Travelling Salesman Problem based on the Guided Dynamic Programming Framework. The main difference between these prototypes is the information extraction:

- the Out-link prototype uses a priori data from the distance matrix,
- the Random Solutions prototype uses feasible parts data from random solutions,
- the Two-Phase DP prototype uses meta-data from DP edge counters.

The Out-link prototype is equal to the State and Out-link Restricted DP from the literature. The Random Solutions and Two-Phase DP prototypes use a state independent binary edge selection with Limited Out-link Search guidance.

To evaluate the performance of the three prototypes we have generated two sets of random Euclidean TSP instances and selected Euclidean TSP instances from literature benchmarks. That is, the following instances are used:

- 50 randomly generated Euclidean TSP instances with 15 customers,
- 50 randomly generated Euclidean TSP instances with 25 customers,
- 12 literature benchmark Euclidean TSP instances of varying sizes.

The effect of the prototype parameters is investigated by solving the randomly generated instances. Furthermore, the two sets allowed us to evaluate the effect of an increase in the size of the instance. Results obtained with the State Restricted DP method are our benchmark references, as this method corresponds to no guidance.

Since the edge counters have not been researched before, we needed to validate the usage of edge counters for guidance. Thereto, we have evaluated the Two-Phase DP under optimal conditions: using edge counters derived from the exact DP state space. The Two-Phase DP prototype showed significant improvements with respect to the reference. However, under realistic conditions (edge counters derived from the Restricted DP state space) the shortcomings of the Two-Phase DP prototype became apparent. For larger instances it seems impractical to obtain reliable edge counters, which greatly reduces the quality of Two-Phase DP. This is confirmed with the literature benchmark instances which have up to 100 nodes.

Since many properties of the edge counters derived from the exact DP state space can be determined, future research can investigate possible edge counter reliability corrections based on these properties. Two-Phase DP should scale better with such corrections, in addition to achieving higher quality solutions.

The Random Solutions prototype is more simplistic, using parts of randomly generated solutions to guide the DP search. From our numerical results we can conclude that the Random Solutions prototype is a valid alternative to Two-Phase DP. Again, the method does not scale well with the size of the instance, as many solutions must be generated for any improvement. It would be interesting to see if the scalability can be improved by generating random instances in a greedy way (non-uniform probabilities based on edge distance for out-link selection).

The Out-link prototype, equal to State and Out-link Restricted DP, gives similar objective values as the reference, but greatly reduces the required computational time. Therefore, its speed makes the method competitive with respect to the other prototypes. Additionally, the methods scales well with the size of the instance, as shown with the literature benchmarks: the Out-link prototype outperforms the reference and the other two prototypes.

Note that the scalability issues do not imply that the Two-Phase DP and Random Solutions prototypes are impractical. For smaller instances both methods show good performance and there are still improvements possible for larger instances (as remarked before). They can be used in combination with methods that reduce the dimension (for instance, by merging nodes). Nevertheless, we must be aware of the limitations of the methods.

For competitive methods for large instances we see potential in combined methods of guidance. For future research we would suggest to determine elite edges with Ant Colony Optimisation based random solutions. A binary, discrete or continuous elite selection range can be used. Furthermore, the feasibility correction of Limited Out-link Search (expanding all out-links if there are no elite out-links) can be altered to only expand the L shortest out-links. That is, we incorporate the Out-link prototype. Of course, Limited Discrepancy Search can be researched.

Chapter 4

Application to the Vehicle Routing Problem

4.1 Introduction

A common approach to solve the Vehicle Routing Problem is Column Generation. Several (exact) solution methods in combination with Column Generation are discussed. We focus on the generation of new columns and present a Guided Dynamic Programming approach. In contrast to the application to the Travelling Salesman Problem, this Guided Dynamic Programming method uses information that is only available when performing Column Generation.

Section 4.2 presents several formulations of the Vehicle Routing Problem, a Column Generation solution approach and several tour generating methods based on Dynamic Programming. In Section 4.3 we discuss new approaches for Guided Dynamic Programming compared to guidance for the Travelling Salesman Problem. Finally, we apply the described methods to literature benchmark instances. The methodology is described in Section 4.4 and the results in Section 4.5.

4.2 Vehicle Routing Optimisation Models

The Vehicle Routing Problem (VRP) can be seen a generalisation of the Travelling Salesman Problem. Instead of finding one tour that services all customers exactly once, multiple (disjoint) tours are allowed. Each tour corresponds to a vehicle and usually there is a fixed number of vehicles available. Many variants of the VRP exist. We will focus on the Capacitated VRP with Time Windows (CVRPTW): each vehicle has a maximum load (capacity) and each customer must be served between predefined time windows. Furthermore, a single depot exists where all vehicles start and end their tours. The vehicles are assumed to be homogeneous (identical). We give a more formal definition of this NP-hard problem ([24]).

Let $G = (V, E)$ be the graph (the network) of the CVRPTW with $K \in \mathbb{N}_{>0}$ homogeneous vehicles. Node $s \in V$ is the (only) depot, the other nodes the customers. Each customer has a load demand, which is given by the function $c_{load} : V \to \mathbb{R}_{\geq 0}$ (with $c_{load}(s) = 0$). Each vehicle has a maximum load demand of $C_{load} \in \mathbb{R}_{\geq 0}$. The function $c_{serv} : V \to \mathbb{R}_{\geq 0}$ states the service time for each customer (again with $c_{serv}(s) = 0$). Furthermore, service must start in a predefined time interval $[t_{begin}(v), t_{end}(v)]$ with $t_{begin}(v), t_{end}(v) \in \mathbb{R}_{\geq 0}$ for $v \in V$. Note that $t_{end}(s)$ denotes the time that a vehicle must be returned to the depot.

Each edge $e \in E$ has a travel distance $c_{dist}(e) \in \mathbb{R}_{\geq 0}$ and time $c_{time}(e) \in \mathbb{R}_{\geq 0}$. A route is a tour starting and ending in depot s, whilst visiting a subset of (unique) customers. It is feasible if the maximum vehicle load and customer time window constraints are satisfied. We are now tasked to find a set of up to K feasible routes such that each customer is served exactly once and the total distance travelled is minimised.

The CVRPTW can be described as a mathematical model using flows along edges. The decision variables are $x_k : E \to \{0, 1\}$ for $k \in \{1, \ldots, K\}$, which indicates whether edge e is part of the route of vehicle k. We need additional decision variables $y_k : V \to \mathbb{R}_{\geq 0}$ to implement the time interval constraints. Here $y_k(v)$ is the time at which service of node v by vehicle k starts. Similar to the TSP, we define $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$.

The mathematical model of the Flow formulation for CVRPTW is:

$$\min \sum_{k=1}^{K} \sum_{e \in E} c_{dist}(e) x_k(e),$$

subject to

$$\sum_{\{v_j \in V : (v_i, v_j) \in E\}} x_k(v_i, v_j) - \sum_{\{v_j \in V : (v_j, v_i) \in E\}} x_k(v_j, v_i) = 0 \qquad \forall\, v_i \in V, k \in \{1, \ldots, K\}, \tag{4.2.1}$$

$$\sum_{\{v \in V : (s, v) \in E\}} x_k(s, v) \leq 1 \qquad \forall\, k \in \{1, \ldots, K\}, \tag{4.2.2}$$

$$\sum_{k=1}^{K} \sum_{\{v_j \in V : (v_i, v_j) \in E\}} x_k(v_i, v_j) = 1 \qquad \forall\, v_i \in \hat{V}, \tag{4.2.3}$$

$$\sum_{(v_i, v_j) \in E} c_{load}(v_j) x_k(v_i, v_j) \leq C_{load} \qquad \forall\, k \in \{1, \ldots, K\}, \tag{4.2.4}$$

$$x_k(e) \in \{0, 1\} \qquad \forall\, e \in E, k \in \{1, \ldots, K\},$$

and the time constraints for all $k \in \{1, \dots, K\}$:

$$t_{begin}(v) \leq y_k(v) \leq t_{end}(v) \qquad \forall\, v \in V, \tag{4.2.5}$$

$$y_k(v_i) + c_{serv}(v_i) + c_{time}(v_i, v_j) - y_k(v_j) + \mathcal{M} x_k(v_i, v_j) \leq \mathcal{M} \qquad \forall\, (v_i, v_j) \in E, v_j \neq s, \tag{4.2.6}$$

$$y_k(v) + c_{serv}(v) + c_{time}(v, s) - t_{end}(s) + \mathcal{M} x_k(v, s) \leq \mathcal{M} \qquad \forall\, (v, s) \in E, \tag{4.2.7}$$

$$y_k(s) = 0$$

$$y_k(v) \geq 0 \qquad \forall\, v \in \hat{V}.$$

Equations (4.2.1) are the flow balance constraints. Multiple visits to the depot within one route are disallowed by Equation (4.2.2). Each customer is visited exactly once due to Equation (4.2.3) and the maximum load is satisfied by (4.2.4). Feasibility for the time intervals are guaranteed by (4.2.5). Equations (4.2.6) and (4.2.7) are needed for correct service start times. Note that $\mathcal{M}$ is a very large number. It is needed to ignore the two constraints if edge (v_i, v_j) or (v, s) is not used by vehicle k. Therefore, $y_k(v)$ has no interpretative value when v is not visited by vehicle k.

The Flow formulation can be solved directly, see [26] for an overview. Instead, we will focus on the Set Covering model of CVRPTW and a Column Generation solution approach with Dynamic Programming. New columns (vehicle tours) are generated by solving the corresponding pricing problem and a reformulation of CVRPTW with Dynamic Programming methods. We elaborate these methods in the following sections.

4.2.1 Set Partition and Set Covering Formulations

The Set Partition formulation can be derived from the above Flow model by applying a Dantzig-Wolfe decomposition (see for instance [14]) and is a very common formulation for the VRP. Let $\Omega = \{T_1, \dots, T_{|\Omega|}\}$ be the set of all feasible tours (routes) of the CVRPTW instance in question. Each vehicle tour $T_l \in \Omega$ is defined as $T_l = (s, \hat{V}_l, s)$ with $\hat{V}_l \subseteq \hat{V}$ the customers visited by the vehicle. The total travelled distance of this tour is given by $c_{dist}(T_l)$, total load by $c_{load}(T_l)$ and total required time $c_{time}(T_l)$. Note that $c_{time}(T_l)$ includes travel, service and waiting time. Let $a_l(v) = 1$ if customer $v \in \hat{V}$ is in T_l and 0 otherwise. The binary decision variables are z_l, indicating if vehicle tour T_l is used.

The Set Partition formulation is given by:

$$\min \sum_{T_l \in \Omega} c_{dist}(T_l) z_l$$

subject to

$$\sum_{T_l \in \Omega} a_l(v) z_l = 1 \qquad \forall\, v \in \hat{V}, \tag{4.2.8}$$

$$\sum_{T_l \in \Omega} z_l \leq K, \tag{4.2.9}$$

$$z_l \in \{0, 1\} \qquad \forall\, T_l \in \Omega.$$

All customers are visited exactly once by Equation (4.2.8) and the number of used vehicles is at most K (see (4.2.9)). When applying Column Generation the equality in Equation (4.2.8) is usually relaxed to an inequality: all customers must be visited at least once. The resulting formulation is called the Set Covering model. The Linear Programming (LP) relaxation of the Set Covering model generally results in more stable dual variables, a favourable property as will be seen later on (when using Column Generation). Furthermore, in practice the binary constraint for the decision variables in the Set Covering model is relaxed to non-negative integers.

The Linear Programming (LP) relaxation of the Set Covering model is given by:

$$\min \sum_{T_l \in \Omega} c_{dist}(T_l) z_l$$

subject to

$$\sum_{T_l \in \Omega} a_l(v) z_l \geq 1 \qquad \forall\, v \in \hat{V}, \tag{4.2.10}$$

$$\sum_{T_l \in \Omega} z_l \leq K, \tag{4.2.11}$$

$$z_l \geq 0 \qquad \forall\, T_l \in \Omega.$$

Since the number of variables in the Set Partition model is usually exponentially large, Column Generation ([14, 28]) is applied to solve the problem. Consider the LP relaxation of the Set Covering model. Instead of using all feasible tours Ω, only a subset $\Omega' \subseteq \Omega$ is used to solve the LP relaxation. The resulting (optimal) dual variables[1] are $\lambda^*(v) \geq 0$ for $v \in \hat{V}$ (Equation (4.2.10)) and $\mu^* \geq 0$ (Equation (4.2.11)). Based on these dual variables, new tours (in $\Omega \backslash \Omega'$) are generated by solving the pricing problem. If a new feasible tour has negative reduced cost, the LP solution can be improved. The reduced costs for a tour $T_l \in \Omega$ are defined as

$$c_{red}(T_l) = c_{dist}(T_l) - \sum_{v \in \hat{V}} \lambda^*(v) a_l(v) + \mu^*.$$

We can also define the reduced costs for each edge:

$$\begin{aligned} c_{red}(s, v_j) &= c_{dist}(s, v_j) - \lambda^*(v_j) && \forall\, v_j \in \hat{V}, \\ c_{red}(v_i, v_j) &= c_{dist}(v_i, v_j) - \lambda^*(v_j) && \forall\, (v_i, v_j) \in \hat{E}, \\ c_{red}(v_i, s) &= c_{dist}(v_i, s) + \mu^* && \forall\, v_i \in \hat{V}. \end{aligned}$$

Hence, the pricing problem for given dual variables $\lambda^*(v)$ for $v \in \hat{V}$ and μ^* is

$$\min\{c_{red}(T_l) : T_l \in \Omega\} = \min \Big\{ c_{dist}(T_l) - \sum_{v \in \hat{V}} \lambda^*(v) a_l(v) + \mu^* : T_l \in \Omega \Big\}.$$

If the minimum is non-negative, we have solved the LP relaxation to optimality. This optimal solution can then be used in a Branch and Bound (Brand and Price) method to solve the Set Covering model. Note that Branch and Bound usually branches on edges and the number of vehicles, not on variables z_l. See for instance [14] for more details.

[1] Dual variables can also be determined with Lagrangian relaxation, see [18, 23].

To solve the pricing problem we need to find a feasible tour such that its reduced cost is minimised. This corresponds to the Elementary Capacitated Shortest Path Problem with Time Windows (ECSPPTW) in the same graph G, but with reduced costs c_{red} as distance function. The ECSPPTW is strongly NP-hard, see [11].

However, if the elementary condition is relaxed (resulting in special CSPPTW instances), there exists a pseudopolynomial-time algorithm. This relaxation can be achieved by enlarging Ω to include non-elementary tours and setting $a_l(v)$ to the number of times v is visited by T_l. This non-elementary formulation provides weaker LP bounds in general, although improvements can be made by excluding k-cycles in non-elementary tours (see [8]). Therefore, there is a trade-off between the computational time of the pricing problem and the quality of the LP bounds.

Both the elementary and its non-elementary relaxation can be solved with Dynamic Programming[2]. We will discuss the DP method for ECSPPTW in the next section and remark the differences for the CSPPTW relaxation.

4.2.1.1 DP for ECSPPTW

The Dynamic Programming method for the Elementary Capacitated Shortest Path Problem with Time Windows is similar to the DP method for the Travelling Salesman Problem, again constructing a tour starting and ending in the depot. However, due to the resource constraints (time and load), multiple partial paths are needed for each DP state. Partial paths are incomparable if one has less distance travelled but higher load (and likewise with time consumption). Paths that appear to be better might be infeasible in the end, thus to maintain the feasibility and optimality guarantees all these incomparable paths must be evaluated. Comparable paths can be reduced to one path (the best one) by using dominance rules.

A state in the DP state space is defined by a subset of visited nodes $S \subseteq V$ and the last visited node $v \in S$. Every state ending in the depot s corresponds to a feasible vehicle tour. Each state can have multiple paths (found partial solutions) due to the described constrained resource incomparability. To be able to discern the paths of a state, we number these and use the corresponding indicator $p \in \mathbb{N}$. Each partial solution p of state (S, v) is characterised by a distance label $\mathcal{L}^p_{dist}(S, v)$, a path label $\mathcal{L}^p_{path}(S, v)$ and resource labels $\mathcal{L}^p_{time}(S, v)$ and $\mathcal{L}^p_{load}(S, v)$. These labels track the properties of the solution, which are elaborated below. The resource labels can easily be extended to model additional constraints.

As usual, the distance label $\mathcal{L}^p_{dist}(S, v)$ stores the distance of the path p starting in s and ending in v that visits all nodes in S exactly once (and no other nodes). Exceptions are states (S, s) where the starting node is visited twice (a tour). The path label $\mathcal{L}^p_{path}(S, v)$ stores the path p to be able to recover the actual solution in the end. Backtracking is also possible if we store the preceding node and the preceding solution number, which results in a two-dimensional label. Time label $\mathcal{L}^p_{time}(S, v)$ and load label $\mathcal{L}^p_{load}(S, v)$ track the used time and load, respectively. Note that the time label includes travel, service and waiting time. Furthermore, $\mathcal{L}^p_{load}(S, v)$ is the same for all solutions of (S, v), as the load follows directly from the set of visited nodes S.

[2]A recent alternative is suggested in [27], where a Depth-First Search approach is presented.

Consider a state (S, v) and two paths, p_1 and p_2, of that state. Path p_1 dominates path p_2 if and only if

$$\mathcal{L}^{p_1}_{dist}(S, v) \leq \mathcal{L}^{p_2}_{dist}(S, v),$$
$$\mathcal{L}^{p_1}_{time}(S, v) \leq \mathcal{L}^{p_2}_{time}(S, v),$$

and at least one inequality is strict. In that case, path p_2 can be discarded without loss of optimality. If these dominance rules are neglected, the DP method will correspond to a Brute Force approach. Dominance rules often greatly reduce the number of paths in the state space.

Dynamic Programming for ECSPPTW has $|V|$ stages: each stage corresponds to the cardinality of the set $S \subseteq V$ of visited nodes (excluding the first starting node). The stages are considered in increasing order and during each stage all corresponding states are expanded. Without loss of generality we can assume that $c_{load}(s) = 0$, $c_{serv}(s) = 0$ and $t_{begin}(s) = 0$. With this assumption, the first states are initialised for each node $v \in \hat{V}$ such that $(s, v) \in E$:

$$\begin{aligned}
\mathcal{L}_{dist}(\{v\}, v) &= c_{dist}(s, v),\\
\mathcal{L}_{load}(\{v\}, v) &= c_{load}(v),\\
\mathcal{L}_{time}(\{v\}, v) &= \max\{c_{time}(s, v), t_{begin}(v)\},\\
\mathcal{L}_{path}(\{v\}, v) &= (s).
\end{aligned}$$

Feasibility must be checked. Note that $c_{load}(v) \leq C_{load}$ must hold, otherwise the instance is infeasible. If $\mathcal{L}_{time}(\{v\}, v) > t_{end}(v)$, then that connection is infeasible. If c_{time} satisfies the triangle inequality and $\mathcal{L}_{time}(\{v\}, v) + c_{serv}(v) + c_{time}(v, s) > t_{end}(s)$, the vehicle cannot return to the depot in time. This would imply that the instance is infeasible. Infeasible paths are discarded. If an evaluated state has no feasible paths, it can be discarded as well and ignored in all expansions.

In stage $\sigma \in \{1, \ldots, |V| - 1\}$ each path p of the states (S, v_i) with $S \subseteq \hat{V}$, $|S| = \sigma$ and $v_i \in S$ is expanded to node $v_j \in V \setminus S$ if $(v_i, v_j) \in E$:

$$\begin{aligned}
\mathcal{L}^{q}_{dist}(S \cup \{v_j\}, v_j) &= \mathcal{L}^{p}_{dist}(S, v_i) + c_{dist}(v_i, v_j),\\
\mathcal{L}^{q}_{load}(S \cup \{v_j\}, v_j) &= \mathcal{L}^{p}_{load}(S, v_i) + c_{load}(v_j),\\
\mathcal{L}^{q}_{time}(S \cup \{v_j\}, v_j) &= \max\{\mathcal{L}^{p}_{time}(S, v_i) + c_{serv}(v_i) + c_{time}(v_i, v_j), t_{begin}(v_j)\},\\
\mathcal{L}^{q}_{path}(S \cup \{v_j\}, v_j) &= (\mathcal{L}^{p}_{path}(S, v_i), v_j),
\end{aligned}$$

where $q \in \mathbb{N}$ is the corresponding path (solution) number in the next state. The new path is feasible if:

$$\begin{aligned}
\mathcal{L}^{q}_{load}(S \cup \{v_j\}, v_j) &\leq C_{load},\\
\mathcal{L}^{q}_{time}(S \cup \{v_j\}, v_j) &\leq t_{end}(v_j),\\
\mathcal{L}^{q}_{time}(S \cup \{v_j\}, v_j) &\leq t_{end}(s) - (c_{serv}(v_j) + c_{time}(v_j, s)),
\end{aligned}$$

where the last condition (the total time required to return to the depot) assumes that the triangle inequality holds for c_{time}. Additional paths are discarded by using the dominance rules.

Any path ending in s is complete (a tour) and is not expanded further. During DP stage σ multiple tours with equal number of visited customers are generated (σ customers are visited). In context of Column Generation, if any of these tours has negative reduced cost, it can improve the LP relaxation. Recall that we use the reduced cost as distance function in the pricing problem. In addition, multiple new columns can be added after a single pricing problem, as DP (usually) generates multiple tours.

4.2.1.2 Improvements to DP

Many of the improvements in Section 2.2.3 for the TSP can also be applied to ECSPPTW, in particular bidirectional DP, see [34]. In [12] Feillet et al. suggest an improved use of the subset of visited nodes S. They define the term 'unreachable nodes': a node is unreachable for a path if it is already included in the path or if some of the resources (load or time) disallow the expansion to this node. This expansion is a direct expansion with an edge. Since the triangle inequality holds for both load and time consumption, any indirect expansion also cannot reach this node.

Instead of using S as the set of visited nodes, let S denote the set of unreachable nodes. Note that this requires some adjustments to the DP method as expansions can 'skip' DP stages. The benefit of this approach is that dominance rules can be applied more often (paths that first belonged to different states can now be aggregated into the same state).

The final improvement we mention is a general method when resource constraints are involved. Preprocessing the instance can often lead to the tightening of time windows, see [35]. For instance, minimal arrival time from predecessors, minimal arrival time to successors, maximal departure time from predecessors and maximal departure time to successors can be used.

4.2.1.3 DP for the CSPPTW Relaxation

If we project the DP state space with states (S, v) to a new state space with corresponding states $(|S|, v)$, the ECSPPTW transforms in a special case of the (non-elementary) CSPPTW. The key difference with the elementary DP method is that the non-elementary DP method does not need to keep track of the visited nodes, as these can be revisited. Only the number of visited nodes is used and each state can be depicted by (n, v) with $n \in \{1, \dots, |V|\}$ and $v \in V$. Since each state (n, v) is a projection of original states (S, v) with $|V| \geq |S| = n$, n can be at most $|V|$. Hence, the solution labels can only take on a finite number of values within the following bounds:

$$\begin{aligned}
\mathcal{L}_{dist} &\in \big[\min\{0, |V| \min\{c_{dist}(e) : e \in E\}\}, \max\{0, |V| \max\{c_{dist}(e) : e \in E\}\}\big], \\
\mathcal{L}_{load} &\in \big[0, C_{load}\big], \\
\mathcal{L}_{time} &\in \big[0, t_{end}(s)\big].
\end{aligned}$$

Tighter bounds exist. The distance label $\mathcal{L}_{dist}$ can be negative if we use the reduced cost c_{red} as distance function. Note that these bounds also hold for the ECSPPTW. However, the DP state space of the CSPPTW has $|V|^2$ states, instead of the exponential number of states for ECSPPTW. If we assume that c_{load}, c_{serv}, c_{time} and t_{begin} are discrete, then the number of dominating paths (partial solutions) in each state is polynomially bounded by the size of the instance parameters. Hence, the DP for the CSPPTW relaxation is pseudopolynomial.

The dominance relations are similar to those of the ECSPPTW, but we have to include the used load. For a state (n, v) with paths p_1 and p_2, path p_1 dominates p_2 if and only if

$$\begin{aligned}
\mathcal{L}^{p_1}_{dist}(n, v) &\leq \mathcal{L}^{p_2}_{dist}(n, v), \\
\mathcal{L}^{p_1}_{load}(n, v) &\leq \mathcal{L}^{p_2}_{load}(n, v), \\
\mathcal{L}^{p_1}_{time}(n, v) &\leq \mathcal{L}^{p_2}_{time}(n, v),
\end{aligned}$$

and at least one inequality is strict. We will not go into the details of the DP method for the (non-elementary) CSPPTW as it is similar to DP for the ECSPPTW (see also [9]).

If the distance function r_{dist} (or r_{red} in case of Column Generation) satisfies the triangle inequality, then the optimal solution of the CSPPTW is elementary. If G is acyclic the ECSPPTW and CSPPTW coincide as each path in G is elementary. The topological order of the nodes can be used to loop over all nodes. Similar ideas can be used if there is a resource consumption that is strictly positive, as this presents an order in the paths.

In [34] Righini and Salani present the Decremental State Space Relaxation: a procedure based on CSPPTW which iteratively adds elementary constraints for nodes that are revisited. For each of these critical nodes a new binary resource is introduced that is consumed when that node is visited. Thus, this approach can be seen as a compromise between elementary and non-elementary CSPPTW.

4.2.2 Giant Tour Representation

The Constrained Vehicle Routing Problem can be translated in a special case of the Constrained Travelling Salesman Problem. Consider the Constrained VRP instance $G = (V, E)$ with K vehicles and depot $s_1 \in V$. Recall that $\hat{V} = V \setminus \{s_1\}$. For each vehicle $k \in \{2, \dots, K\}$ add a copy of the depot to the graph, including edges to and from customers. These copies will indicate the start of the tour of a new vehicle. Denote all copies with $s_2, \dots, s_K$, respectively. All (copy) depots are interconnected. That is, we have the Constrained TSP instance $G' = (V', E')$ with starting node $s_1 \in V'$ and

$$\begin{aligned} V' &= V \cup \{s_k : k = 2, \dots, K\}, \\ E' &= E \cup \{(s_k, s_l) : k, l = 1, \dots, K, k \neq l\} \\ &\quad \cup \{(s_k, v) : (s_1, v) \in E, k = 2, \dots, K\} \\ &\quad \cup \{(v, s_k) : (v, s_1) \in E, k = 2, \dots, K\}. \end{aligned}$$

The distance and the resource consumption of copied edges is the same as the original, although visits to the copied depots reset all resource labels, to be explained next. Edges between depots have zero distance and no resource consumption. This special TSP instance is called the Giant Tour Representation (GTR) of the VRP (see also [16]).

To avoid confusion, we will call the vehicle (the salesman) of the constructed GTR the super-vehicle. Solving this TSP is similar to solving the ECSPPTW of Section 4.2.1.1. The super-vehicle starts in the original depot s_1, of course, followed by visits to some customers and a copy depot. The labels to keep track of the used resources are updated accordingly. When it visits any copy s_k ($k \in \{2, \dots, K\}$) all labels corresponding to the resource constraints are reset as if a new vehicle starts its tour. Note that the visited customers are not reset. Additional customers are visited and the process repeats itself when a copy depot is reached, until all customers are serviced.

In fact, this indicates exactly how we can translate the GTR solution back to a VRP solution. Given a GTR solution $T = (s_1, \hat{V}_1, s_2, \dots, s_k, \hat{V}_k, s_{k+1}, \dots, s_K, \hat{V}_K, s_1)$ with $\hat{V}_k \subseteq \hat{V}$ the customers visited between depots s_k and s_{k+1} (or s_K and s_1) for all $k \in \{1, \dots, K\}$. Note that $\cup_{k=1}^{K} \hat{V}_k = \hat{V}$ and $| \cup_{k=1}^{K} \hat{V}_k| = |\hat{V}|$. The corresponding (feasible) VRP solution is the set of K vehicle tours $T_k = (s_1, \hat{V}_k, s_1)$ with $k \in \{1, \dots, K\}$ (and vice versa). A tour $T_k = (s_1, s_1)$ indicates that a vehicle is not used. See Appendix A.3 for an example.

Thus, we can solve the Constrained VRP with the GTR, which boils down to solving a Constrained TSP with some special properties (the resource usage reset). The reset of resource labels when visiting a depot can easily be included in a DP solution method for the Constrained TSP. The next section describes Dynamic Programming for this special Constrained TSP.

Although the GTR solves the VRP directly, we will use it to generate vehicle tours in a heuristic way: the GTR can be used in combination with Column Generation as a source of new columns. The reduced cost c_{red} is used as distance function for the GTR and a solution is determined. Each (individual) vehicle tour of this solution can be added as a new column, but some choices have to be made. In general, the complete GTR solution can have positive reduced costs, whereas some individual columns can have negative reduced costs (improving the LP solution). However, it cannot provide an optimality guarantee for the LP relaxation (it does not solve the pricing problem). A (complete) GTR solution is a feasible Set Partition solution. Therefore, it makes sense to add all individual tours as columns.

4.2.2.1 DP for the Giant Tour Representation

Although most types of Constrained VRP have a Giant Tour Representation, we will focus on the CVRPTW as before. In this case the GTR is a special Capacitated TSPTW. The DP method for a general CTSPTW is the same as for ECSPPTW (Section 4.2.1.1) except that the depot node cannot be visited before all customers are serviced (see [31]). Note that TSPTW is an NP-hard problem ([36]), thus CTSPTW and GTR as well.

DP for the GTR is similar to a repeated ECSPPTW in the modified graph G'. States ending in a depot node are special. When a path can be extended to a depot whilst maintaining feasibility, both the time label $\mathcal{L}_{time}$ and load label $\mathcal{L}_{load}$ are reset to zero (the starting values). The distance label $\mathcal{L}_{dist}$ and path label $\mathcal{L}_{path}$ are retained. This path is then expanded in the usual way. When all customers have been serviced the remaining copy depot nodes are visited consecutively to complete the DP method.

4.2.2.2 Improvements to DP

Care must be taken to avoid duplicate partial solutions, as the GTR contains many symmetries: not only can the depots be interchanged without changing the real solution, this also holds for the sets V_k. It is therefore useful to only allow the depots to be visited in their natural order $1, \ldots, K$ and only allow chains of consecutive depots at the end of the GTR solution. The DP state space based on (unordered) sets of visited nodes removes some symmetric duplicates each time a depot is reached (but not all and not in between depots). Using the load of unvisited customers per remaining vehicle can reduce unwanted short vehicle tours, but a good upper bound on the optimal required number of vehicles is needed.

Again, some improvements for the TSP (Section 2.2.3) can also be applied to the GTR. Also, preprocessing the instance can tighten time windows and remove obsolete edges. For instance, if the load of two nodes exceed the maximum vehicle load, any edge between the two nodes can be removed. As a final remark, notice that a GTR can be solved exactly by first solving the ECSPPTW exactly and then evaluating combinations of these tours.

4.3 DP Guidance with Column Generation

The Guided Dynamic Programming Framework can be applied to the described DP methods in order to solve CVRPTW and GTR. The Guided DP will be based on the restricted version of the exact DP. That is, the number of paths (partial solutions) expanded is limited to B paths and the number of feasible out-links is restricted to L edges per path. Note that the B paths are selected from all states, not per state. Furthermore, the out-links are first checked for feasibility before restricted to L out-links. As expected, we call this method Restricted DP.

For Restricted DP the selection of the B paths to expand is (again) not trivial. In fact, the usage of the different resources (load and time) must be considered in addition to the travelled distance (or reduced costs). Similarly, should the shortest out-links be selected or the out-links to customers which can be served the earliest? In general, feasibility issues (for a complete CVRPTW solution) can occur if resources are not taken into account in the selection procedure.

The Restricted DP method will be guided as in Section 2.4. However, the resource constraints affect fundamental properties of the DP method. A straightforward imitation of the application to the TSP of Chapter 3 can give undesirable results. Note that the resource constraints already limit the allowed expansions. We will discuss the main differences and suggest alternative guidance methods. For instance, Column Generation allows new sources of information for the selection of elite edges. The three parts of the framework (information extraction, elite expansion selection and DP guidance) are treated separately.

4.3.1 Information Extraction

The resource data (travel times, time windows, loads) can function as a priori data. Although preprocessing the instance (e.g. removing infeasible edges due to time windows) is similar to guiding the DP search, it is an exact method whereas guidance is a heuristic. Put differently, preprocessing does not affect the optimality guarantee. However, consider the case that two customers are near each other with adjacent or overlapping time windows. Intuitively, visiting the two customers consecutively seems a good approach. This is not guaranteed optimal. By giving preference to this edge, we can implement this intuition in the guidance of the DP search.

Using feasible parts and meta-data is still valid. The edge counters do need adjustments, since each state has multiple paths due to incomparable partial solutions. After applying the dominance rules we can update the usage counters for each remaining path. In a way, this implies that each state has multiple origins. Of course, the theoretical properties of the usage counters in Appendix B.2 are no longer valid after this modification. Alternatively, one can select the best path of a state and only update the usage counter for this path. For instance, the path with the smallest travelled distance can be selected.

A new source of meta-data is the coordinating Column Generation and Branch and Bound procedure. Edges fixed by the Branch and Bound method can often be implemented by elite edges (depending on the guidance). More interesting is the meta-data from the master problem in Column Generation.

4.3.1.1 Meta-data from Column Generation

The tours in the previous LP solution can function as a good source of feasible parts data (we still classify this as meta-data). For ECSPPTW this implies that a tour similar to those LP tours is searched. This is particularly useful if the LP tours are reasonably good. As the GTR constructs a feasible partition solution, it can be seen to combine the (overlapping) LP tours to get a very similar partition solution. In this case, c_{dist} or c_{red} as distance function are both valid. The regular distance c_{dist} would give a solution based on the actual distances, which is more appropriate for a final solution. For generation of columns the reduced cost are more appropriate. In addition, the LP tours can be very similar between iterations, whereas reduced cost varies more.

Instead of only considering the last LP solution, we can use all edges ever used in an LP solution. This can lead to early stagnation in the elite edges. Only considering the last couple of LP solutions, say the last 5 iterations, is more robust.

In contrast to the previous examples which look at similar tours, we can focus on nodes with the least coverage among all generated columns. For example, there is only one generated column that covers a certain customer $v \in V$, but all other customers are covered by a multitude of generated columns. Clearly, more alternatives for covering this customer is generally beneficial for Column Generation. Giving preference to any edge to this node v guides the DP search towards tours that visit customer v.

The main benefit of using this meta-data for guidance, is that it is 'free': little to no extra computational effort is required. This is in contrast to the guidance applied to the TSP in Chapter 3, where we had to perform an extra DP or generate random solutions.

The use of LP solutions and generated columns can be seen as a primal approach to find new columns (no duality is needed), whereas using reduced costs is the classic dual approach. The combination of these primal and dual approaches show great potential to guide the generation of new columns.

4.3.2 Elite Expansion Selection

Time windows generally implicate a natural order in which nodes can be visited. If an expansion is very preferred at the end of the vehicle time window, it makes little sense to prefer this expansion at the beginning. The vehicle would probably have to wait a long time before this customer can be serviced. State dependent elite selection is most likely a better choice, for instance based on the current travel time and load of the vehicle.

State independent elite selection would work properly in combination with Limited Discrepancy Search (the search is less restricted with more elite edges). Some elite expansions made would make little sense, but these are most likely not selected among the next B paths to expand. A combination with Limited Out-link Search has a greater impact as non-elite expansions are ignored if at least one elite expansion is feasible. If the reliability of the elite edges is low, this can cause bad results (worse than with Limited Discrepancy Search). Binary, discrete or continuous elite selection ranges are all unchanged.

4.3.3 DP Guidance

Both the Limited Out-link Search (LOS) and the Limited Discrepancy Search (LDS) can guide the DP search for the ECSPPTW and the GTR. Besides the remarks made above and in Section 2.4, it is advisable to always allow expansions to the depot. In case of LOS (with feasibility adjustment), edges to the depot should not be elite, but simply allowed in addition to any other edge. If we would make the edges to the depot elite for each expansion, all non-elite edges are discarded. This would lead to unnecessary short vehicle tours. In case of LDS, edges to the depot should be elite.

There are two interpretations of the maximum allowed discrepancy for the GTR. The first one is the standard approach: the maximum holds for the whole tour of the super-vehicle. The second interpretation is that the maximum holds for each (individual) vehicle tour of the GTR. In this case, we must reset the discrepancy label (which stores the amount of discrepancy of the path) when a depot is visited, similarly as the time and load labels. We prefer the second approach, as it has a clear interpretation, it is more comparable to LDS for ECSPPTW and it solves the feasibility problem of LDS. Whenever a path of the super-vehicle cannot be expanded due to the maximum discrepancy it returns to the depot and can start afresh. Do note that a too strict maximum discrepancy leads to many short vehicle tours.

4.4 Methodology

We have a simplistic solution approach for the CVRPTW based on Column Generation as described in Section 4.2. We consider Euclidean CVRPTW instances and assume that the travel time is equal to the distance, $c_{time} = c_{dist}$. No Branch and Bound is implemented, instead we solve the Set Covering LP relaxation only once with Column Generation. The pricing problem is solved by the Restricted DP method for the ECSPPTW, a heuristic. Therefore, there is no guarantee that the LP relaxation is solved to optimality. If the LP solution has not changed for a fixed number of iterations, we assume that the method has converged and stop the Column Generation method. Afterwards, a feasible solution is constructed by using the assumption that travel times satisfy the triangle inequality.

First, we will give a general overview of the Column Generation solution approach, followed by the construction of a feasible CVRPTW solution. Finally, we discuss the methods for the pricing problem and the GTR. These column generating methods are simplistic heuristics based on the exact DP methods. These will be our references. Both methods can be guided as in Section 4.3, however, we only consider guiding DP for the GTR.

4.4.1 Column Generation Solution Approach

The solution approach for the CVRPTW is based on the Set Covering formulation in Section 4.2.1. The LP relaxation is solved using Column Generation with heuristic column generating methods, see Algorithm 4.4.1. First, only dummy tours $T = (v)$ for $v \in \hat{V}$ are available that have an artificial high cost such that it can never be optimal to choose a dummy tour over a feasible tour. These dummy tours do not use a vehicle. This allows us to always find a 'solution' for the LP. Of course, if any dummy tour is used by the solution, no actual feasible solution LP is found.

When the Set Covering LP relaxation is solved with this initial set of tours (columns), all dummy tours are selected and we get dual variables for each customer (all equal in this case). These dual variables allow us to generate new (feasible) tours with negative reduced cost, thus improving the earlier LP solution. We resolve the LP with the increased set of tours, resulting in new dual variables. This process repeats itself until the tour generating methods fail to construct tours with negative reduced cost. Note that since heuristics are used, this is not an optimality guarantee. In fact, if the objective value of the LP solution does not change for 10 consecutive iterations, or if 100 iterations are performed, the Column Generation method is terminated.

For large instances, many tours can be constructed which turn out to be of little value in the end. It is therefore useful to restrict the number of tours, after solving the LP. If the number of tours exceeds a predefined threshold we apply a strict reduction in the column pool. We have set the threshold to 1000 tours. All tours with non-positive reduced cost are retained, in addition to any tour that has ever been in the LP solution. That is, we use $C_{red} = 0$ in Algorithm 4.4.2.

Algorithm 4.4.3 gives an overview of the tour generation procedure. As mentioned before, we can generate tours by solving the pricing problem (ECSPPTW) or by solving the GTR. In both cases we can opt to guide the generation by applying the Guided Dynamic Programming Framework. The actual generation of tours will be described in the following sections. Before we go into the details, we discuss how a feasible CVRPTW solution is constructed.

Algorithm 4.4.1 Column Generation Solution Approach

Input: VRP instance parameters, general and guidance specific Column Generation parameters
Output: travelled distance of VRP solution

```
 1: procedure COLUMN GENERATION SOLUTION APPROACH(input)
 2:     initialise dummy tours with artificial high costs: $\Omega = \{(v) : v \in \hat{V}\}$

 3:     solve: Set Covering LP relaxation with $\Omega$
            ▷ giving optimal tours $\Omega^* \subseteq \Omega$ and dual variables $\lambda^* : \hat{V} \to \mathbb{R}_{\geq 0}$ and $\mu^* \geq 0$

 4:     while no LP convergence and maximum iterations not reached do
 5:         if number of tours exceeds threshold then
 6:             determine reduced costs: $c_{red}(T) = c_{dist}(T) - \sum_{v \in T \cap \hat{V}} \lambda^*(v) + \mu^*$     for all $T \in \Omega$
 7:             remove subset of tours: $\Omega = $ REDUCE TOURS$(\Omega, \Omega^*, c_{red}, C_{red})$
 8:         generate additional tours: $\Omega' = $ GENERATE TOURS$(\lambda^*, \mu^*)$
 9:         add tours: $\Omega = \Omega \cup \Omega'$
10:         solve: Set Covering LP relaxation with $\Omega$
                ▷ giving optimal tours $(\Omega^*)' \subseteq \Omega$ and dual variables $\lambda^* : \hat{V} \to \mathbb{R}_{\geq 0}$ and $\mu^* \geq 0$
11:         update list of optimal tours of any previous LP: $\Omega^* = \Omega^* \cup (\Omega^*)'$

12:     solve Set Partition IP: $\Omega^* = $ MAKE PARTITION SOLUTION$(\Omega^*)$

13:     return objective value of Set Partition IP solution $\Omega^*$
```

Algorithm 4.4.2 Reduce Tours Procedure

Input: set of tours Ω, subset of tours that have been part of any LP solution $\Omega^* \subseteq \Omega$,
reduced costs of tours $c_{red} : \Omega \to \mathbb{R}$, reduced cost threshold $C_{red} \in \mathbb{R}$
Output: subset of selected tours

```
 1: procedure REDUCE TOURS(input)
 2:     select all tours with reduced cost below threshold: $\Omega' = \{T \in \Omega : c_{red}(T) \leq C_{red}\}$

 3:     keep tours that have been part of any LP solution: $\Omega' = \Omega' \cup \Omega^*$

 4:     return $\Omega'$
```

Algorithm 4.4.3 Generate Tours Procedure

Input: VRP instance parameters, guidance specific Column Generation parameters, dual variables $\lambda^* : \hat{V} \to \mathbb{R}_{\geq 0}$ and $\mu^* \geq 0$

Output: set of tours

```
 1: procedure GENERATE TOURS(input)
 2:     initialise set of additional tours: $\Omega' = \emptyset$

 3:     determine reduced costs of edges:
            $c_{red}(s, v_j) = c_{dist}(s, v_j) - \lambda^*(v_j) \quad \forall\, v_j \in \hat{V},$
            $c_{red}(v_i, v_j) = c_{dist}(v_i, v_j) - \lambda^*(v_j) \quad \forall\, (v_i, v_j) \in \hat{E},$
            $c_{red}(v_i, s) = c_{dist}(v_i, s) + \mu^* \quad \forall\, v_i \in \hat{V}.$

 4:     if unguided pricing problem then
 5:         generate tours: $\Omega'' =$ ECSPPTW-RESTRICTED-DP($c_{red}$)
 6:         add tours: $\Omega' = \Omega' \cup \Omega''$
 7:     if guided pricing problem then
 8:         generate tours: $\Omega'' =$ ECSPPTW-GUIDED RESTRICTED-DP($c_{red}$)
 9:         add tours: $\Omega' = \Omega' \cup \Omega''$

10:     if unguided GTR then
11:         generate tours: $\Omega'' =$ CVRPTW-RESTRICTED-GTR($c_{red}$)
12:         add tours: $\Omega' = \Omega' \cup \Omega''$
13:     if guided GTR then
14:         generate tours: $\Omega'' =$ CVRPTW-GUIDED RESTRICTED-GTR($c_{red}$)
15:         add tours: $\Omega' = \Omega' \cup \Omega''$

16:     return $\Omega'$
```

4.4.2 Constructing a Feasible CVRPTW Solution

After the convergence of the Column Generation method we use the generated tours (no dummy tours) to construct a CVRPTW solution: switch to Integer Programming (IP) and the Set Partition formulation. Instead of trying to solve the Set Partition IP immediately, we first allow the generation of additional tours by removing some customers, see Algorithm 4.4.4.

First, we solve the Set Covering IP with the current set of tours. If this is infeasible, some customers are not visited by any tour or there are not enough vehicles for the current set of tours. If the instance is feasible and the number of vehicles is not too strict, this is unlikely to happen. If allowed, trivial tours (s, v, s) can be added for any uncovered customer $v \in \hat{V}$, but this does not solve the limited number of vehicles. This infeasibility never occurred for our instances and no feasibility improvements are implemented (an infeasibility error is returned).

Second, we iteratively remove customers from the (optimal) tours in the Set Covering IP solution. A customer that is covered by multiple optimal tours is removed in each optimal tour. The removal is done independently, so the resulting tours can still overlap. Therefore, we resolve the Set Covering IP with the increased set of tours and repeat the process up to 5 times. Note that these new tours are feasible because we assume that the travel times satisfy the triangle inequality.

Finally, we solve the Set Partition IP. If no feasible solution can be found, we solve the Set Covering IP instead and repeat the removal of customers (again up to 5 'inner' iterations as above). Overall, this process is repeated until a feasible CVRPTW solution is returned. Since a feasible Set Covering IP solution exists and the triangle inequality holds, such a solution must exist (at most 2 'outer' iterations were needed for our instances).

4.4.3 ECSPPTW Tours Generation

To solve the pricing problem of the CVRPTW we use a heuristic based on the exact DP method for ECSPPTW. Instead of having multiple (incomparable) paths per state we only keep the shortest path (if multiple shortest paths exist, one is arbitrarily chosen). Therefore, a state corresponds to one constructed path (a partial solution). The heuristic also restricts the number of states and out-links: at most B states are expanded, evaluating at most L feasible out-links for each expansion. The selection (ranking) of states is based on the reduced cost of the path, likewise for the out-links. This is similar to the State and Out-link Restricted DP method of the TSP, but do note that the out-links are first checked for feasibility. The expansion to the depot is always performed first. Algorithm 4.4.5 gives an overview of the heuristic.

In our implementation, any generated feasible vehicle tour is also stored separately during the execution of DP. The 10 shortest (feasible) tours generated by the DP heuristic are returned, not the set $\{\mathcal{L}_{path}(S, s) : S \subseteq V, |S| \geq 2\}$. When adding these new tours to the set of tours Ω' we remove any duplicates. That is, for each set $S \subseteq \hat{V}$ of serviced customers we only keep the shortest tour. Also, new tours are added even if their reduced cost is non-negative. These tours could potentially be useful for creating a feasible VRP solution.

The State and Out-link Restricted DP heuristic for ECSPPTW can be guided as described in Section 4.3. We will not guide the pricing problem, we focus on the GTR instead.

Algorithm 4.4.4 Make Partition Solution Procedure

Input: VRP instance parameters, set of tours Ω
Output: VRP solution

```
1:  procedure MAKE PARTITION SOLUTION(input)
2:      remove dummy tours: $\Omega' = \{T \in \Omega : T \neq (v), \forall v \in V\}$

3:      solve: Set Covering IP with $\Omega'$
                ▷ giving optimal tours $\Omega^* \subseteq \Omega'$ for Set Covering IP

4:      if no Set Covering IP solution then
5:          return $\Omega^* = \emptyset$

6:      while no Set Partition IP solution and maximum outer iterations not reached do
7:          while no Set Partition IP solution and maximum inner iterations not reached do
8:              generate new tours: $\Omega'' =$ REMOVE CUSTOMERS FROM TOURS($\Omega^*$)
9:              add tours: $\Omega' = \Omega' \cup \Omega''$
10:             solve: Set Covering IP with $\Omega'$
                        ▷ giving optimal tours $\Omega^* \subseteq \Omega'$ for Set Covering IP
11:         solve: Set Partition IP with $\Omega'$
                    ▷ giving optimal tours $\Omega^* \subseteq \Omega'$ for Set Partition IP

12:     return $\Omega^*$


13: procedure REMOVE CUSTOMERS FROM TOURS(set of tours $\Omega^*$)
14:     assumption: $c_{time}$ satisfies triangle inequality
15:     determine set of revisited customers: $V' = \{v \in \hat{V} : |\{T \in \Omega^* : v \in T\}| \geq 2\}$

16:     initialise set of new tours: $\Omega'' = \emptyset$

17:     for all $v \in V'$ do
18:         for all $T \in \Omega^*$ with $v \in T$ do
19:             remove $v$ from tour: $T' = T \setminus \{v\}$
20:             add new tour: $\Omega'' = \Omega'' \cup \{T'\}$

21:     return $\Omega''$
```

Algorithm 4.4.5 State and Out-link Restricted DP Heuristic for ECSPPTW

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and VRP instance parameters, state beam width $B \in \mathbb{N}_{>0}$, out-link number $L \in \mathbb{N}_{>0}$
Output: elementary tours

```
 1: procedure ECSPPTW-STATE AND OUT-LINK RESTRICTED-DP(input)
 2:     for all $S \subseteq V$ and $v \in S$, and for $(S, v) = (\emptyset, s)$ do
 3:         initialise state space labels:
              $\mathcal{L}_{dist}(S, v) = \infty, \quad \mathcal{L}_{load}(S, v) = 0, \quad \mathcal{L}_{time}(S, v) = 0, \quad \mathcal{L}_{path}(S, v) = \emptyset$

 4:     expand initial state: EXPAND STATE($(\emptyset, s), L$)

 5:     for all stages $\sigma = 1, \dots, |V| - 1$ do
 6:         for all states $(S_i, v_i)$ in STATES TO EXPAND($\sigma, B$) do
 7:             expand state $(S_i, v_i)$: EXPAND STATE($(S_i, v_i), L$)

 8:     return $\{\mathcal{L}_{path}(S, s) : S \subseteq V, |S| \geq 2\}$


 9: procedure EXPAND STATE(state $(S_i, v_i)$, out-link number $L$)
10:     for all nodes $v_j \in \{v \in V \setminus S_i : (v_i, v) \in E\}$ do
11:         rank edge $(v_i, v_j)$
12:         insert node $v_j$ in list according to rank of $(v_i, v_j)$

13:     initialise counter: $l = 0$
14:     for all nodes $v_j$ in list in order of rank do
15:         expand state: $b =$ ECSPPTW-EXPAND($(S_i, v_i), v_j$)
16:         if $b =$ true then
17:             increment counter: $++l$
18:             if $l \geq L$ then
19:                 return

20:     return


21: procedure STATES TO EXPAND(stage $\sigma$, state beam $B$)
22:     for all subsets $S_i \in \{S \subseteq V \setminus \{s\} : |S| = \sigma\}$ do
23:         for all nodes $v_i \in S_i$ do
24:             rank state $(S_i, v_i)$
25:             insert state in list according to rank of $(S_i, v_i)$

26:     return up to $B$ highest ranked elements from list
```

Algorithm 4.4.6 ECSPPTW Path Expansion Procedure

Input: directed graph $G = (V, E)$ with starting node $s \in V$ and VRP instance parameters, state (S, v_i) with $S \subseteq V \setminus \{s\}$ and $v_i \in S$, node $v_j \in V \setminus S$

Output: true if expansion to state $(S \cup \{v_j\}, v_j)$ is performed and feasible, false otherwise

```
1:  procedure ECSPPTW-Expand(input)
2:      initialise boolean: $b = \textbf{false}$

3:      new distance: $l_{dist} = \mathcal{L}_{dist}(S, v_i) + c_{dist}(v_i, v_j)$

4:      new load: $l_{load} = \mathcal{L}_{load}(S, v_i) + c_{load}(v_j)$

5:      new time: $l_{time} = \max\{\mathcal{L}_{time}(S, v_i) + c_{serv}(v_i) + c_{time}(v_i, v_j), t_{begin}(v_j)\}$

6:      new path: $l_{path} = (\mathcal{L}_{path}(S, v_i), v_j)$

7:      if $l_{load} \leq C_{load}$ then
8:          if $l_{time} \leq t_{end}(v_j)$ then
9:              if $l_{time} \leq t_{end}(s) - (c_{serv}(v_j) + c_{time}(v_j, s))$ then
10:                 set: $b = \textbf{true}$
11:                 if $l_{dist} < \mathcal{L}_{dist}(S \cup \{v_j\}, v_j)$ then
12:                     update distance label: $\mathcal{L}_{dist}(S \cup \{v_j\}, v_j) = l_{dist}$
13:                     update load label: $\mathcal{L}_{load}(S \cup \{v_j\}, v_j) = l_{load}$
14:                     update time label: $\mathcal{L}_{time}(S \cup \{v_j\}, v_j) = l_{time}$
15:                     update path label: $\mathcal{L}_{path}(S \cup \{v_j\}, v_j) = l_{path}$

16:     return $b$
```

4.4.4 GTR Tours Generation

The exact DP method for the GTR is restricted in the same way as for the ECSPPTW. Each state has only one path (the shortest partial solution), up to B states are expanded and up to L feasible out-links are allowed for expansion. Reduced costs are used for the selection of states and out-links. The heuristic is shown in Algorithm 4.4.7 (with $H' = \emptyset$ for unguided DP). Notice the usage of the GTR graph $G' = (V', E')$ and the reset of labels when visiting a copy depot. A return to the depot s_1 is only allowed if all customers (including copy depots) are visited.

Again, our implementation slightly deviates from the shown algorithm. To reduce duplicate solutions, we only allow expansion to one unvisited copy depot (the one with the lowest index, see the natural order in Section 4.2.2.2). This expansion is always performed first. Furthermore, all feasible GTR tours are stored separately. The 5 shortest GTR tours are translated to individual vehicle tours and returned. Duplicates are removed in the same way as for ECSPPTW. In contrast to ECSPPTW tour generation, the GTR is solved once in every 3 iterations.

When guiding the Restricted DP heuristic for GTR with Limited Out-link Search (LOS), we select elite edges $H' \subseteq E'$ based on the previous LP solution, see Algorithm 4.4.9. This selection guides DP to construct a feasible VRP solution similar to the LP solution.

We have the following ranking of the out-links. An out-link to a depot always has the highest rank (as mentioned we only allow one out-link to a depot). The elite out-links in H' are grouped and ranked second, followed by all remaining (normal) out-links. Among the elite out-links, the out-links are ranked according to their distance (reduced cost), in increasing order. The remaining out-links are sorted similarly. To implement LOS guidance, we limit the number of out-links to $\min\{L, 1 + |\{v \in \hat{V}' \setminus S_i : (v_i, v) \in H'\}|\}$ if there is a feasible elite expansion for state (S_i, v_i). Since the out-link to the depot is never elite, we need the $+1$ in the expression.

4.4.5 References and Prototype Methods

Two reference solution methods are our benchmarks, which do not use any guidance. The first method only uses the ECSPPTW pricing problem to generate new tours, whereas the second method uses both ECSPPTW and GTR. We call these methods reference 1 and reference 2, respectively. Reference 2 allows us to determine the effect of the GTR tour generation. For all DP methods of the references the state beam width is set to $B = 1000$ and the out-links are restricted to $L = 10$.

The prototype will use both ECSPPTW and GTR for generating tours. We apply LOS guidance to the GTR, where the elite edges are the edges used in the previous LP solution (as described in Section 4.4.4). All parameters are the same as the references: $B = 1000$, $L = 10$.

We only show the result for these parameter settings. The individual effects of each parameter is difficult to investigate and to present as the methods are used in a Column Generation approach. Column Generation performs multiple iterations, where each iteration depends on the preceding iterations. One iteration can be singled out, but this does not show the impact on the whole Column Generation approach. Therefore, we use the tour generation methods more like a black box. The main goal is to illustrate Guided DP in context of Column Generation.

Algorithm 4.4.7 LOS Guided Restricted DP Heuristic for Giant Tour Representation

Input: GTR graph $G' = (V', E')$ with depot $s_1 \in V'$ and copied depots $s_2, \ldots, s_K \in V'$, VRP parameters, state beam $B \in \mathbb{N}_{>0}$, out-link number $L \in \mathbb{N}_{>0}$, elite edges $H' \subseteq E'$
Output: GTR solution

```
1:  procedure GTR-LOS Guided Restricted-DP(input)
2:      set: $\hat{V}' = V' \setminus \{s_1\}$
3:      for all $S \subseteq V'$ and $v \in S$, and for $(S, v) = (\emptyset, s_1)$ do
4:          initialise state space labels:
                $\mathcal{L}_{dist}(S, v) = \infty$,   $\mathcal{L}_{load}(S, v) = 0$,   $\mathcal{L}_{time}(S, v) = 0$,   $\mathcal{L}_{path}(S, v) = \emptyset$

5:      expand initial state: Expand State$((\emptyset, s_1), L)$

6:      for all stages $\sigma = 1, \ldots, |\hat{V}'| - 1$ do
7:          for all states $(S_i, v_i)$ in States to Expand$(\sigma, B)$ do
8:              expand state $(S_i, v_i)$: Expand State$((S_i, v_i), L)$

9:      for all states $(\hat{V}', v_i)$ in States to Expand$(|\hat{V}'|, B)$ do
10:         expand state $(\hat{V}', v_i)$: GTR-Expand$((\hat{V}', v_i), s_1)$

11:     return $\mathcal{L}_{path}(V', s_1)$


12: procedure Expand State(state $(S_i, v_i)$, out-link number $L$, elite edges $H'$)
13:     for all nodes $v_j \in \{v \in \hat{V}' \setminus S_i : (v_i, v) \in E\}$ do
14:         rank edge $(v_i, v_j)$
15:         insert node $v_j$ in list according to rank of $(v_i, v_j)$

16:     initialise counter: $l = 0$
17:     for all nodes $v_j$ in list in order of rank do
18:         expand state: $b =$ GTR-Expand$((S_i, v_i), v_j)$
19:         if $b =$ true then
20:             if $(v_i, v_j) \in H'$ then
21:                 set: $L = \min\{L, 1 + |\{v \in \hat{V}' \setminus S_i : (v_i, v) \in H'\}|\}$
22:             increment counter: $++l$
23:             if $l \geq L$ then
24:                 return

25:     return


26: procedure States to Expand(stage $\sigma$, state beam $B$)
27:     for all subsets $S_i \in \{S \subseteq \hat{V}' : |S| = \sigma\}$ do
28:         for all nodes $v_i \in S_i$ do
29:             rank state $(S_i, v_i)$
30:             insert state in list according to rank of $(S_i, v_i)$

31:     return up to $B$ highest ranked elements from list
```

Algorithm 4.4.8 GTR Path Expansion Procedure

Input: GTR graph $G' = (V', E')$ with depot $s_1 \in V'$ and copied depots $s_2, \dots, s_K \in V'$, VRP parameters, state (S, v_i) with $S \subseteq \hat{V}'$ and $v_i \in S$, node $v_j \in V' \setminus S$

Output: true if expansion to state $(S \cup \{v_j\}, v_j)$ is performed and feasible, false otherwise

```
1:  procedure GTR-Expand(input)
2:      initialise boolean: $b = $ false

3:      new distance: $l_{dist} = \mathcal{L}_{dist}(S, v_i) + c_{dist}(v_i, v_j)$

4:      new load: $l_{load} = \mathcal{L}_{load}(S, v_i) + c_{load}(v_j)$

5:      new time: $l_{time} = \max\{\mathcal{L}_{time}(S, v_i) + c_{serv}(v_i) + c_{time}(v_i, v_j), t_{begin}(v_j)\}$

6:      new path: $l_{path} = (\mathcal{L}_{path}(S, v_i), v_j)$

7:      if $l_{load} \leq C_{load}$ then
8:          if $l_{time} \leq t_{end}(v_j)$ then
9:              if $l_{time} \leq t_{end}(s_1) - (c_{serv}(v_j) + c_{time}(v_j, s_1))$ then
10:                 set: $b = $ true
11:                 if $l_{dist} < \mathcal{L}_{dist}(S \cup \{v_j\}, v_j)$ then
12:                     if $v_j \in \{s_1, \dots, s_K\}$ then
13:                         reset load: $l_{load} = 0$
14:                         reset time: $l_{time} = 0$
15:                     update distance label: $\mathcal{L}_{dist}(S \cup \{v_j\}, v_j) = l_{dist}$
16:                     update load label: $\mathcal{L}_{load}(S \cup \{v_j\}, v_j) = l_{load}$
17:                     update time label: $\mathcal{L}_{time}(S \cup \{v_j\}, v_j) = l_{time}$
18:                     update path label: $\mathcal{L}_{path}(S \cup \{v_j\}, v_j) = l_{path}$

19:     return $b$
```

Algorithm 4.4.9 Elite Edge Selection based on LP Solution

Input: GTR graph $G' = (V', E')$ with depot $s_1 \in V'$ and copied depots $s_2, \dots, s_K \in V'$, original VRP graph $G = (V, E)$ with depot $s_1 \in V$, LP solution Ω^*

Output: set $H' \subseteq E'$ of elite edges

```
1:  procedure Elite Edge Selection based on LP Solution(input)
2:      initialise: $H = \emptyset$
3:      for all tours $T_l \in \Omega^*$ do
4:          translate tour into set of used edges: $\theta_l \subseteq E$
5:          add used edges to set: $H = H \cup \theta_l$

6:      remove edges to depot: $H = H \setminus \{(v, s_1) \in E : v \in V\}$

7:      translate to GTR graph: $H' = H \cup \{(s_k, v) \in E' : (s_1, v) \in H, k \in \{2, \dots, K\}\}$

8:      return $H'$
```

4.5 Results for Literature Benchmarks

The performance of the two reference methods and the prototype is evaluated using the Solomon CVRPTW instances, a common benchmark in the literature. In particular, we investigate the effect of the addition of GTR tour generation and the effect of guiding GTR, as described in Section 4.4. First, we discuss the Solomon instances in more detail.

4.5.1 Instances

The Solomon CVRPTW instances[3] with 100 customers are our benchmark instances. These instances are well-known and used often in the literature to compare methods. The objective is to minimise the travelled distance whilst serving each customer. The best known values will be used for the relative objective gaps. Note that the Solomon instances are also used with other objective functions, for instance, to first minimise the number of required vehicles and then the travelled distance.

The Solomon instances are 56 Euclidean instances, where travel time equals travel distance. They are divided into six types of instances:

- instances R1 and R2 are uniformly generated,
- instances C1 and C2 are clustered,
- instances RC1 and RC2 are a mix between uniform and clustered.

The coordinates of the customers (the distance matrices) are identical within instance classes R and RC and types C1 and C2, see also Figure D.2.1. Instances within one of the six types differ with respect to the time windows. Furthermore, instances R1, C1 and RC1 have a short scheduling horizon and allow few customers per vehicle tour (5 to 10 customers). Instances R2, C2 and RC2 have a long scheduling horizon and more than 30 customers can be serviced by one vehicle.

4.5.2 Numerical Results

The numerical results are presented in Appendix D. First the results of the Set Covering LP relaxation are shown. These results are after the Column Generation has terminated, but before the Set Partition is solved with Algorithm 4.4.4. The best known values are shown and used for the relative objective gaps. Note that negative gaps can occur, since the results are for the LP relaxation. Furthermore, some best known values have been found by heuristics, not by exact methods. For each reference and the prototype we state the objective value, the corresponding relative gap and the required computational time. The average for each of the six types of instances are given, in addition to the average for each class (R, C and RC).

For a general overview of the Set Covering LP results, see Table D.3.7. The performance of the references and the prototype vary significantly between the six types of instances. For C1, C2, R2 and RC2 the inclusion of the GTR has a significant positive effect on the objective value and the required computational time. The guided GTR (the prototype) often outperforms the unguided GTR (reference 1) or gives similar objective values in less computational time.

[3]For more information and the best known values, see `http://web.cba.neu.edu/~msolomon/problems.htm`.

For R1 and RC1 reference 1 (no GTR) gives the best objective values. Although reference 2 and the prototype often require less computational time, the resulting objective values are at least two times worse.

Overall, reference 1 obtains an average relative objective gap of 0.2197, requiring on average 96.1975 seconds. Reference 2 achieves a gap of 0.1544 in 39.7272 seconds and the prototype a gap of 0.1409 in 36.3945 seconds. Therefore, we can conclude that in general the addition of the GTR is beneficial, both in objective values as in computational time. An unguided GTR results in 6.5% lower relative objective gap and a 58.7% reduction in computational time. However, the guided GTR gives the best overall performance, with 7.9% lower relative objective gap and a 62.2% reduction in time.

As mentioned before, often Branch and Bound methods are combined with Column Generation to find a feasible (or optimal) CVRPTW solution. We have implemented a simple method to construct a feasible CVRPTW solution from the Set Covering LP solution, as described in Section 4.4.2. The Set Partition IP results are also shown in Appendix D. Note that a negative relative objective gap would imply a new best known value. However, this is never achieved.

The overview of the Set Partition IP results is shown in Table D.3.8. For the IP solution the inclusion of the GTR is beneficial for all types of instances (except for R1 with unguided GTR when only considering the relative objective gap). Again, the guided GTR outperforms the unguided GTR or gives similar results in less computational time. Overall, reference 1 achieves an average relative gap of 0.3085 in 124.0850 seconds. Reference 2 has an average gap of 0.1688 in 41.4538 seconds and the prototype 0.1506 in 36.8250 seconds. Notice the significant increase in computational time for reference 1 to construct a feasible CVRPTW solution. Hence, the unguided GTR leads to a 14.0% lower gap with a 66.6% reduction in computational time. The guided GTR again outperforms the other methods, with a 15.8% lower gap and a 70.3% reduction in time.

We conclude that incorporating GTR tour generation into the Column Generation approach is in general beneficial, both in objective values as in computational time. In particular this holds for instances with clustered customers (type C) or instances with long scheduling horizons (C2, R2, RC2). Guiding the GTR DP search with Limited Out-link Search and elite edges equal to those used by the previous LP solution (moderately) outperforms the unguided GTR method in general.

4.6 Conclusion

We have presented two exact solution methods for the Vehicle Routing Problem: Column Generation combined with Branch and Bound, and the Giant Tour Representation. The pricing problem of the Column Generation approach and the Giant Tour can be solved using Dynamic Programming. We have considered a special case of VRP: the Capacitated VRP with Time Windows. In this case, the pricing problem is an Elementary Capacitated Shortest Path Problem with Time Windows. The Giant Tour corresponds to a special case of Capacitated TSP with Time Windows. Both Dynamic Programming methods for these problems have similarities with the DP method for the TSP, however multiple partial solutions per state are required.

A common approach for the Column Generation approach is to solve a non-elementary CSPPTW as pricing problem. That is, the ECSPPTW is relaxed (projected) to non-elementary paths. Although it results in weaker lower bounds for the Branch and Bound procedure, the benefit is that the DP method becomes pseudopolynomial. We have focussed on elementary paths and limited ourselves to the ECSPPTW.

Although the Giant Tour Representation solves the CVRPTW directly, we have incorporated it in the Column Generation approach as a source of new tours (in addition to the pricing problem). Column Generation allows additional information sources for Guided DP, which can be applied to the GTR. Examples of new data are the edges used by LP solutions and customers that are least covered by all generated tours. This allows for a primal approach to guide the generation of new tours, in addition to the usual dual approach (reduced costs). The strength of GTR is that it can be guided to make a feasible CVRPTW solution from parts of tours (or at least very similar to those tours).

We have applied these ideas of the Guided DP Framework to the CVRPTW by implementing a simplistic Column Generation approach. Instead of exact DP methods we have used Restricted DP and limited the number of paths per state to one partial solution. Using the well-known Solomon CVRPTW instances with 100 customers, we have evaluated the performance of three methods:

- the first method only uses the (unguided) pricing problem to generate tours,
- the second method uses the (unguided) pricing problem and unguided GTR,
- the final method uses the (unguided) pricing problem and guided GTR.

The GTR is guided with Limited Out-link Search and elite edges equal to the edges used in the previous LP solution.

Overall, the addition of the GTR to generate tours is beneficial, both with regards to the resulting objective value as to the required computational time. Furthermore, the prototype (guided GTR) outperforms the other methods (on average). These results illustrate the inclusion of a primal approach to guide the generation of tours.

The presented Column Generation approach has several shortcomings. However, the main goal was to illustrate the use of Guided DP in context of Column Generation. Some suggested improvements to the approach are: the incorporation of Branch and Bound, more advanced 'column management' of the set of generated tours, and allowing multiple paths per state in the DP methods.

For future research alternative guidance based on Column Generation information can be investigated, in particular one that considers customers with low coverage among the generated tours. With Branch and Bound the initial guidance of a new branch can be based on the solution of its 'parent' branch node. Good management of the generated tours also prevents repeated generation of tours among the branches.

We have only considered guiding the GTR. A natural extension is to also guide the pricing problem. If the vehicles can only visit a few customers due to maximum load or time windows, scalability issues of guidance methods should have a reduced impact. This allows us to use the Guided DP methods similar to those applied to the Travelling Salesman[4]. Further research is needed to validate this intuition.

[4]Preliminary results of a Two-Phase DP pricing problem prototype combined with guided GTR are as follows. For the Set Covering LP a relative objective gap of 0.1333 is obtained in 39.3472 seconds on average. For the Set Partition IP a relative gap of 0.1438 is achieved in 39.7235 seconds on average. The best known value for RC106 is improved from 1424.73 to 1418.2943 (-0.4517%). Phase 1 uses $B_1 = 500$, $L_1 = 10$ and returns 5 tours. The percentile level is set to $\tau = 0.9$. Phase 2 uses $B_2 = 1000$, $L_2 = 10$ and returns 5 tours. The other parameters are the same as in Section 4.4.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Based on exact Dynamic Programming (DP) solution methods and existing DP heuristics, we have developed the Guided Dynamic Programming heuristic. It is an extension of Restricted Dynamic Programming and allows the DP search to be guided. The different possibilities for DP guidance have resulted in a general framework called the Guided Dynamic Programming Framework.

The framework consists of three parts: information extraction, elite expansion selection and DP guidance. We have divided the information extraction into three classes: data from instance parameters, data from solutions and data from solution methods. The elite expansion selection quantifies the level of eliteness of expansions using a binary, discrete or continuous classification. Finally, we can guide (Restricted) Dynamic Programming to only use elite expansions (Limited Out-link Search) or only allow a maximum deviation from elite expansions (Limited Discrepancy Search).

For the Travelling Salesman Problem (TSP) we have presented and evaluated three Guided Dynamic Programming prototypes, where the Out-link prototype corresponds to an existing method from the literature (State and Out-link Restricted DP). The Out-link prototype extracts information from the distance matrix. The other two prototypes, Two-Phase DP and Random Solutions, use the DP state space and random solutions as information source, respectively. All three prototypes use binary elite edges and Limited Out-link Search guidance. The (unguided) State Restricted DP is used as the benchmark method to compare performance.

The results show that a binary classification is sensitive to the selection criteria (the percentile level or the number of out-links in our research). Furthermore, reliability of the elite edges seems to be vital for good performance. Both the Two-Phase DP prototype and the Random Solutions prototype do not scale well with the size of the instance, as they are unable to extract reliable elite edges. The Out-link prototype had the best performance for the selected Euclidean TSP benchmark instances from the literature. However, for small randomly generated Euclidean instances, the Two-Phase DP and Random Solutions prototypes show significant improvement in performance with respect to the reference.

For the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) we have implemented a simplistic Column Generation solution approach. The pricing problem is solved with a Restricted DP heuristic that only allows one partial solution per DP state. This is our first reference method. The Giant Tour Representation (GTR) of the CVRPTW can also be used to generate new vehicle tours and can be solved with a similar Restricted DP heuristic. The inclusion of GTR as an additional tour generator is our second reference. Finally, we have developed a Guided DP method for the GTR. Edges in the previous Linear Programming (LP) relaxation solution of the Column Generation approach are classified as elite. Furthermore, this prototype uses binary elite edges and Limited Out-link Search guidance.

An interesting feature of this prototype is that is uses both primal and dual information to find new vehicle tours. The primal information is the set of edges used by the previous LP solution. We have also given more examples of primal information available with a Column Generation approach. The dual information is the usual reduced cost based on dual variables from the previous LP solution.

To evaluate the references and the prototype we have used the Solomon CVRPTW instances with 100 customers. The addition of the GTR (guided or unguided) significantly improves the resulting objective values and greatly reduces the required computational time in general. An added benefit is that a GTR constructs feasible CVRPTW solutions (upper bounds) during the Column Generation procedure. The Guided DP prototype outperforms both references, although the results are close to those of the unguided GTR reference.

To conclude, the Guided Dynamic Programming Framework allows the development of Guided Dynamic Programming heuristics that incorporate preferred expansions in the DP search. These preferred expansions can be determined by intuitive arguments or empirical results. Based on our numerical evaluation of several prototypes, we conclude that the framework certainly shows potential. However, care must be taken for a correct implementation, as guidance with unreliable elite edges can lead to unsatisfying results.

5.2 Future Work

We suggest the following adjustments for (potential) improvements of the discussed prototypes. The Two-Phase DP needs reliability corrections of edge counters from theoretical results if large instances need to be solved. These corrections can originate from the properties of edge counters derived from the exact DP state space or from a probabilistic analysis of the Restricted DP state space. For instance, edge counters from random walks can give expected values that can indicate the extremity of counters from DP.

The Random Solutions prototype can be improved by using a non-uniform distribution for the out-link selection. For instance, the edge distances can affect the probability that an edge is selected (giving preference to shorter edges). The Out-link prototype already performs well and originates from the literature. Combinations of the three prototypes can be investigated.

We have no direct improvements for the guided GTR prototype for the Vehicle Routing Problem, except to extend the heuristic to allow more partial solutions per state. Of course, any reference Dynamic Programming method should also use this improvement.

More general, other guidance methods can be investigated. In particular, discrete or continuous elite edge selection and Limited Discrepancy Search guidance can be used. Additional numerical evaluations can be performed, preferably with different types of instances (for example, clustered and uniformly distributed instances). Also, reference solution methods not based on Dynamic Programming can be used.

Two promising Guided Dynamic Programming areas are

- Guided DP based on primal information in Column Generation,
- Guided DP based on pheromone weights of Ant Colony Optimisation.

The usage of primal information in addition to dual information in a Column Generation approach has already been discussed before. A benefit of this information source is that it is readily available when performing a Column Generation procedure.

Guided DP can be incorporated in a parallel solution approach with Ant Colony Optimisation. During the execution of the Ant Colony procedure the current pheromone weights can be used to guide DP. Every couple of Ant Colony iterations a Guided DP can be performed in parallel, each time with updated weights. In fact, Guided DP can be included in many Meta-heuristics by using an iterative procedure where elite edges are updated.

Bibliography

[1] Arora, S. and B. Barak. *Computational Complexity: A Modern Approach*, Cambridge University Press, April 2009.

[2] Arora, S. 'Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems.' *Journal of the ACM*, Volume 45, Issue 5, September 1998, Pages 753-782.

[3] Bellman, R. 'The theory of dynamic programming.' *Bulletin of the American Mathematical Society*, Volume 60, Issue 6, August 1954, Pages 503–515.

[4] Bellman, R. 'Dynamic programming treatment of the Traveling Salesman Problem.' *Journal of the ACM*, Volume 9, Issue 1, January 1962, Pages 61–63.

[5] Bloemendal, C. 'Adapting the DP framework applied to the VRP to efficiently deal with similar nodes.' Master's Thesis, December 2010.

[6] Breimer, E., M. Goldberg, D. Hollinger and D. Lim. 'Discovering Optimization Algorithms Through Automated Learning.' *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Volume 69, December 2005, Pages 7–25.

[7] Cordeau, J.-F. and G. Laporte. 'Tabu Search Heuristics for the Vehicle Routing Problem.' In: C. Rego and B. Alidaee (editors). *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, Kluwer Academic Publishers, Boston, 2005, Pages 145–163.

[8] Desaulniers, G., F. Lessard and A. Hadjar. 'Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows.' *Transportation Science*, Volume 42, Issue 3, March 2008, Pages 387–404.

[9] Desrochers, M., J. Desrosiers and M. Solomon. 'A new optimization algorithm for the vehicle routing problem with time windows.' *Operations Research*, Volume 40, Issue 2, March 1992, Pages 342–354.

[10] Dorigo, M. and T. Stützle. 'Ant Colony Optimization: Overview and Recent Advances.' In: M. Gendreau and J.-Y. Potvin (editors). *Handbook of Metaheuristics*, Springer, 2010, Pages 227–263.

[11] Dror, M. 'Note on the complexity of the shortest path models for column generation in VRPTW.' *Operations Research*, Volume 42, Issue 5, 1994, Pages 977–978.

[12] Feillet, D., P. Dejax, M. Gendreau and C. Gueguen. 'An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems.' *Networks*, Volume 44, Issue 3, 2004, Pages 216–229.

[13] Feillet, D., M. Gendreau and L. Rousseau. 'New Refinements for the Solution of Vehicle Routing Problems with Branch and Price.' *INFOR*, Volume 45, Issue 4, November 2007, Pages 239–256.

[14] Feillet, D. 'A tutorial on column generation and branch-and-price for vehicle routing problems.' *4OR*, Volume 8, Issue 4, December 2010, Pages 407–424.

[15] Fischetti, M. and P. Toth. 'An additive bounding procedure for combinatorial optimization problems.' *Operations Research*, Volume 37, Issue 2, March 1989, Pages 319–328.

[16] Funke, B., T. Grünert and S. Irnich. 'Local Search for Vehicle Routing and Scheduling Problems: Review and Conceptual Integration.' *Journal of Heuristics*, Volume 11, 2005, Pages 267–306.

[17] Furcy, D. and S. Koenig. 'Limited discrepancy beam search.' *Proceedings of the 19th international joint conference on Artificial intelligence (IJCAI'05)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, August 2005, Pages 125–131.

[18] Huisman, D., R. Jans, M. Peeters and A. Wagelmans. 'Combining Column Generation and Lagrangian Relaxation.' In: G. Desaulniers, J. Desrosiers and M. Solomon (editors). *Column Generation*, Springer, New York, 2005, Pages 247–270.

[19] Gromicho, J., J. van Hoorn, A. Kok and J. Schutten. 'Restricted dynamic programming: A flexible framework for solving realistic VRPs.' *Computers and Operations Research*, Volume 39, Issue 5, May 2012, Pages 902–909.

[20] Held, M. and R.M. Karp. 'A dynamic programming approach to sequencing problems.' *Journal of the Society for Industrial and Applied Mathematics*, Volume 10, Issue 1, March 1962, Pages 196–210.

[21] Van den Hoeven, A. 'A truncated dynamic programming approach to the Traveling Salesman Problem.' Master's Thesis, July 2005.

[22] Van Hoorn, J. 'Iterative Dynamic Programming for the Constrained Vehicle Routing Problem.' Master's Thesis, October 2008.

[23] Kallehauge, B., J. Larsen and O. Madsen. 'Lagrangian duality applied to the vehicle routing problem with time windows.' *Computers and Operations Research*, Volume 33, 2006, Pages 1464–1487.

[24] Kallehauge, B. 'Formulations and exact algorithms for the vehicle routing problem with time windows.' *Computers and Operations Research*, Volume 35, 2008, Pages 2307–2330.

[25] Laporte, G. 'The Traveling Salesman Problem: An overview of exact and approximate algorithms.' *European Journal of Operational Research*, Volume 59, 1992, Pages 231–247.

[26] Laporte, G. 'The Vehicle Routing Problem: An overview of exact and approximate algorithms.' *European Journal of Operational Research*, Volume 59, 1992, Pages 345–358.

[27] Lozano, L. and A. Medaglia. ‘On an exact method for the constrained shortest path problem.’ *Computers and Operations Research*, Volume 40, Issue 1, January 2013, Pages 378–384.

[28] Lübbecke, M. ‘Column Generation.’ In: *Wiley Encyclopedia of Operations Research and Management Science*, John Wiley and Sons, Inc., 2011.

[29] Malandraki, C. and R.B. Dial. ‘A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem.’ *European Journal of Operational Research*, Volume 90, Issue 1, April 1996, Pages 45–55.

[30] Marsten, R. and T. Morin. ‘A hybrid approach to discrete mathematical programming.’ *Mathematical Programming*, Volume 14, 1978, Pages 21-40.

[31] Mingozzi, A., L. Bianco and S. Ricciardelli. ‘Dynamic programming strategies for the Traveling Salesman Problem with time window and precedence constraints.’ *Operations Research*, Volume 45, Issue 3, May 1997, Pages 365–377.

[32] Papadimitriou, C.H. and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, Inc. Mineola, New York, 1998.

[33] Pisinger, D. and S. Ropke. ‘A general heuristic for vehicle routing problems.’ *Computers and Operations Research*, Volume 34, 2007, Pages 2403–2435.

[34] Righini, G. and M. Salani. ‘New dynamic programming algorithms for the resource constrained elementary shortest path problem.’ *Networks*, Volume 51, Issue 3, 2008, Pages 155–170.

[35] Ropke, S. ‘Heuristic and exact algorithms for vehicle routing problems.’ Ph.D. Thesis, December 2005.

[36] Savelsbergh, M. ‘Local Search in Routing Problems With Time Windows.’ *Annals of Operations Research*, Volume 4, June 1985, Pages 285–305.

[37] Shen, Z., Q.-C. Zhao and Q.-S. Jia. ‘Quantifying Heuristics in the Ordinal Optimization Framework.’ *Discrete Event Dynamics Systems*, Volume 20, 2010, Pages 441–471.

[38] Tarantilis, C.D. ‘Solving the vehicle routing problem with adaptive memory programming methodology.’ *Computers and Operations Research*, Volume 32, Issue 9, September 2005, Pages 2309–2327.

[39] Toth, P. and D. Vigo. ‘The granular Tabu search and its application to the Vehicle-Routing Problem.’ *INFORMS Journal on Computing*, Volume 15, Issue 4, December 2003, Pages 333–346.

[40] Voß, S. ‘Meta-heuristics: The State of the Art.’ In: A. Nareyek (editor). *Local Search for Planning and Scheduling*, Springer, 2001, Pages 1–23.

Appendix A

Examples

A.1 Introduction

We give a few examples of the described Dynamic Programming (DP) methods, for the Travelling Salesman Problem and for the Vehicle Routing Problem. Not all DP methods discussed in Chapters 3 and 4 have an example. In particular, for the Vehicle Routing Problem we only show how to reformulate the instance as a Giant Tour.

Section A.2 gives an example of the Travelling Salesman Problem. This instance is solved with the Full DP method, the State Restricted DP method and Limited Out-link Search Guided DP. As mentioned above, Section A.3 illustrates the Giant Tour Representation of a Vehicle Routing Problem instance.

A.2 Travelling Salesman Problem

We consider a Travelling Salesman Problem instance with 5 customers: $V = \{1, \ldots, 6\}$ with depot $s = 1$ and $E = \{(v_i, v_j) : v_i, v_j \in V, v_i \neq v_j\}$. The distance matrix is given by

$$d = \begin{pmatrix} 0 & 1 & 6 & 26 & 19 & 24 \\ 3 & 0 & 7 & 21 & 23 & 25 \\ 31 & 9 & 0 & 22 & 27 & 20 \\ 8 & 28 & 33 & 0 & 10 & 15 \\ 30 & 5 & 34 & 12 & 0 & 16 \\ 4 & 36 & 29 & 13 & 18 & 0 \end{pmatrix},$$

where the first row and column correspond to the depot of the salesman. We will solve this TSP instance with the (exact) Full DP method and the State Restricted DP heuristic. Furthermore, we show an example of Guided DP with Limited Out-link Search (LOS) guidance.

A.2.1 Full DP Example

Table A.2.1 shows all DP states and their labels after performing the Full DP method, as described in Section 2.2.2. We denote the set $S \subseteq V$ as a binary vector with 1 on the position of $v \in S$ (all visited nodes) and 0 otherwise. For each set S we give the objective and path labels of each corresponding state (S, v) with $v \in S$. The sets are grouped together for each DP stage $\sigma \in \{0, \ldots, |V| - 1\}$. The optimal tour can be backtracked and is equal to $(1, 3, 6, 4, 5, 2, 1)$ with a travelled distance of 57.

The edge counters introduced in Section 2.4.1.4 can be determined whilst performing the Full DP method. In Table A.2.2 we see the edge expansion counter and the edge usage counter after each stage $\sigma \in \{0, \ldots, |V| - 1\}$. Note that no bias corrections are performed. We can represent the counters as matrices, where the rows and columns correspond to nodes in V. Therefore, we can refer to rows and columns of edge counters. For example, the sum of a row of an edge counter, corresponding to node $v_i \in V$, is equal to the sum of edge counters of all edges $(v_i, v_j) \in E$ with $v_j \in V$.

The counters can be derived from the path labels in Table A.2.1. For example, consider stage $\sigma = 2$ and the usage counter of edge $(4, 3)$. We count the number of occurrences of '4' in the column of node 3 in the path labels. Note that we count all sets up to and including stage 2. We count four occurrences, which corresponds to the shown counter. Do note that we did not count in the column of node 4 for the occurrences of '3', since the path label shows the originating node. For edge $(4, 3)$, we need node 4 to be the origin of a state ending in node 3.

A.2.2 State Restricted DP Example

We apply the State Restricted DP heuristic with $B = 2$ states that are expanded in each stage. Note that the number of out-links is unrestricted (as we do not perform State and Out-link Restricted DP). The B states with the smallest objective labels are expanded. The resulting DP state labels are shown in Table A.2.3. Only a few states are evaluated and the best found solution is the tour $(1, 2, 3, 4, 5, 6, 1)$ with travelled distance 60. This is not the optimal tour.

A.2.3 LOS Guided DP Example

The State and Out-link Restricted DP can be guided by using Limited Out-link Search as described in Section 2.4. Consider the following Guided DP. The number of states is restricted to $B = 2$ and the out-links are unrestricted $L = |\hat{V}|$. We use the edge usage counters from the Full DP method as information to select state independent binary elite edges. For each node we select one out-link as elite: the out-link with the highest edge usage counter (see Table A.2.2). Edges from or to the depot are ignored. Using edge scores gives the same result. Hence, the elite edges $H \subseteq E$ are:

$$H = \{(2,3),(3,6),(4,5),(5,2),(6,4)\},$$

or depicted as a matrix:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Note that the elite edges are almost exactly equal to the shortest out-links. Only edge $(3,6)$ is not the shortest out-link of node 3, which is edge $(3,2)$ (again excluding edges from or to the depot).

The results are shown in Table A.2.4. Coincidentally, only elite edges are used (excluding edges from or to the depot), resulting in only very few state evaluations. No feasibility adjustment is needed. The Guided DP finds the optimal solution with these elite edges.

As the elite edges are almost equal to the shortest out-link of each node, we can also perform an unguided State and Out-link Restricted DP with $B = 2$ and $L = 1$. This results in the tour $(1,2,3,6,4,5,1)$ with travelled distance 81, worse than the other methods.

Stage	Visited Nodes	Objective Label						Path Label					
	Set	*Node*						*Node*					
σ	$(1,2,3,4,5,6)$	1	2	3	4	5	6	1	2	3	4	5	6
0	$(0,0,0,0,0,1)$	-	-	-	-	-	24	-	-	-	-	-	1
	$(0,0,0,0,1,0)$	-	-	-	-	19	-	-	-	-	-	1	-
	$(0,0,0,1,0,0)$	-	-	-	26	-	-	-	-	-	1	-	-
	$(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{0})$	-	-	**6**	-	-	-	-	-	**1**	-	-	-
	$(0,1,0,0,0,0)$	-	1	-	-	-	-	-	1	-	-	-	-
1	$(0,0,0,0,1,1)$	-	-	-	-	42	35	-	-	-	-	6	5
	$(0,0,0,1,0,1)$	-	-	-	37	-	41	-	-	-	6	-	4
	$(0,0,0,1,1,0)$	-	-	-	31	36	-	-	-	-	5	4	-
	$(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{1})$	-	-	53	-	-	**26**	-	-	6	-	-	**3**
	$(0,0,1,0,1,0)$	-	-	53	-	33	-	-	-	5	-	3	-
	$(0,0,1,1,0,0)$	-	-	59	28	-	-	-	-	4	3	-	-
	$(0,1,0,0,0,1)$	-	60	-	-	-	26	-	6	-	-	-	2
	$(0,1,0,0,1,0)$	-	24	-	-	24	-	-	5	-	-	2	-
	$(0,1,0,1,0,0)$	-	54	-	22	-	-	-	4	-	2	-	-
	$(0,1,1,0,0,0)$	-	15	8	-	-	-	-	3	2	-	-	-
2	$(0,0,0,1,1,1)$	-	-	-	48	47	46	-	-	-	6	4	4
	$(0,0,1,0,1,1)$	-	-	64	-	44	49	-	-	6	-	6	5
	$(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{1})$	-	-	70	**39**	-	43	-	-	4	**6**	-	4
	$(0,0,1,1,1,0)$	-	-	64	45	38	-	-	-	4	5	4	-
	$(0,1,0,0,1,1)$	-	47	-	-	44	40	-	5	-	-	6	5
	$(0,1,0,1,0,1)$	-	65	-	39	-	37	-	4	-	6	-	4
	$(0,1,0,1,1,0)$	-	41	-	36	32	-	-	5	-	5	4	-
	$(0,1,1,0,0,1)$	-	62	55	-	-	28	-	3	6	-	-	3
	$(0,1,1,0,1,0)$	-	38	31	-	35	-	-	5	2	-	3	-
	$(0,1,1,1,0,0)$	-	56	55	30	-	-	-	4	4	3	-	-
3	$(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$	-	-	75	56	**49**	54	-	-	6	5	**4**	5
	$(0,1,0,1,1,1)$	-	52	-	53	49	48	-	5	-	6	4	5
	$(0,1,1,0,1,1)$	-	49	54	-	46	51	-	5	2	-	6	3
	$(0,1,1,1,0,1)$	-	67	66	41	-	45	-	4	6	6	-	4
	$(0,1,1,1,1,0)$	-	43	48	47	40	-	-	5	2	5	4	-
4	$(\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$	-	**54**	59	58	51	56	-	**5**	2	5	4	5
5	$(\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$	**57**	-	-	-	-	-	**2**	-	-	-	-	-

Table A.2.1: Full DP example.

Stage		**Edge Expansion**						**Edge Usage**					
		Node						*Node*					
σ	*Node*	1	2	3	4	5	6	1	2	3	4	5	6
0	1	0	1	1	1	1	1	0	1	1	1	1	1
	2	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	1	1	1	0	1	1	1	1	1
	2	0	0	1	1	1	1	0	0	1	1	1	1
	3	0	1	0	1	1	1	0	1	0	1	1	1
	4	0	1	1	0	1	1	0	1	1	0	1	1
	5	0	1	1	1	0	1	0	1	1	1	0	1
	6	0	1	1	1	1	0	0	1	1	1	1	0
2	1	0	1	1	1	1	1	0	1	1	1	1	1
	2	0	0	4	4	4	4	0	0	2	1	1	1
	3	0	4	0	4	4	4	0	2	0	2	2	2
	4	0	4	4	0	4	4	0	3	4	0	4	4
	5	0	4	4	4	0	4	0	4	1	3	0	3
	6	0	4	4	4	4	0	0	1	3	4	3	0
3	1	0	1	1	1	1	1	0	1	1	1	1	1
	2	0	0	7	7	7	7	0	0	4	1	1	1
	3	0	7	0	7	7	7	0	2	0	2	2	3
	4	0	7	7	0	7	7	0	4	4	0	7	5
	5	0	7	7	7	0	7	0	7	1	5	0	5
	6	0	7	7	7	7	0	0	1	5	6	4	0
4	1	0	1	1	1	1	1	0	1	1	1	1	1
	2	0	0	8	8	8	8	0	0	5	1	1	1
	3	0	8	0	8	8	8	0	2	0	2	2	3
	4	0	8	8	0	8	8	0	4	4	0	8	5
	5	0	8	8	8	0	8	0	8	1	6	0	6
	6	0	8	8	8	8	0	0	1	5	6	4	0
5	1	0	1	1	1	1	1	0	1	1	1	1	1
	2	1	0	8	8	8	8	1	0	**5**	1	1	1
	3	1	8	0	8	8	8	0	2	0	2	2	**3**
	4	1	8	8	0	8	8	0	4	4	0	**8**	5
	5	1	8	8	8	0	8	0	**8**	1	6	0	6
	6	1	8	8	8	8	0	0	1	5	**6**	4	0

Table A.2.2: Full DP edge counters example.

Stage	**Visited Nodes**	**Objective Label**						**Path Label**					
	Set	*Node*						*Node*					
σ	$(1, 2, 3, 4, 5, 6)$	1	2	3	4	5	6	1	2	3	4	5	6
0	$(0, 0, 0, 0, 0, 1)$	-	-	-	-	-	24	-	-	-	-	-	1
	$(0, 0, 0, 0, 1, 0)$	-	-	-	-	19	-	-	-	-	-	1	-
	$(0, 0, 0, 1, 0, 0)$	-	-	-	26	-	-	-	-	-	1	-	-
	$(0, 0, 1, 0, 0, 0)$	-	-	6	-	-	-	-	-	1	-	-	-
	$\mathbf{(0, 1, 0, 0, 0, 0)}$	-	**1**	-	-	-	-	-	**1**	-	-	-	-
1	$(0, 0, 0, 0, 1, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 0, 0, 1, 0, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 0, 0, 1, 1, 0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 0, 1, 0, 0, 1)$	-	-	-	-	-	26	-	-	-	-	-	3
	$(0, 0, 1, 0, 1, 0)$	-	-	-	-	33	-	-	-	-	-	3	-
	$(0, 0, 1, 1, 0, 0)$	-	-	-	28	-	-	-	-	-	3	-	-
	$(0, 1, 0, 0, 0, 1)$	-	-	-	-	-	26	-	-	-	-	-	2
	$(0, 1, 0, 0, 1, 0)$	-	-	-	-	24	-	-	-	-	-	2	-
	$(0, 1, 0, 1, 0, 0)$	-	-	-	22	-	-	-	-	-	2	-	-
	$\mathbf{(0, 1, 1, 0, 0, 0)}$	-	15	**8**	-	-	-	-	3	**2**	-	-	-
2	$(0, 0, 0, 1, 1, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 0, 1, 0, 1, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 0, 1, 1, 0, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 0, 1, 1, 1, 0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 1, 0, 0, 1, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 1, 0, 1, 0, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 1, 0, 1, 1, 0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 1, 1, 0, 0, 1)$	-	-	-	-	-	28	-	-	-	-	-	3
	$(0, 1, 1, 0, 1, 0)$	-	-	-	-	35	-	-	-	-	-	3	-
	$\mathbf{(0, 1, 1, 1, 0, 0)}$	-	-	-	**30**	-	-	-	-	-	**3**	-	-
3	$(0, 0, 1, 1, 1, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 1, 0, 1, 1, 1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0, 1, 1, 0, 1, 1)$	-	-	-	-	46	-	-	-	-	-	6	-
	$(0, 1, 1, 1, 0, 1)$	-	-	-	41	-	45	-	-	-	6	-	4
	$\mathbf{(0, 1, 1, 1, 1, 0)}$	-	-	-	-	**40**	-	-	-	-	-	**4**	-
4	$\mathbf{(0, 1, 1, 1, 1, 1)}$	-	-	-	-	51	**56**	-	-	-	-	4	**5**
5	$\mathbf{(1, 1, 1, 1, 1, 1)}$	**60**	-	-	-	-	-	**6**	-	-	-	-	-

Table A.2.3: State Restricted DP example.

Stage	**Visited Nodes**	**Objective Label**						**Path Label**					
	Set	*Node*						*Node*					
σ	$(1,2,3,4,5,6)$	1	2	3	4	5	6	1	2	3	4	5	6
0	$(0,0,0,0,0,1)$	-	-	-	-	-	24	-	-	-	-	-	1
	$(0,0,0,0,1,0)$	-	-	-	-	19	-	-	-	-	-	1	-
	$(0,0,0,1,0,0)$	-	-	-	26	-	-	-	-	-	1	-	-
	$\mathbf{(0,0,1,0,0,0)}$	-	-	**6**	-	-	-	-	-	**1**	-	-	-
	$(0,1,0,0,0,0)$	-	1	-	-	-	-	-	1	-	-	-	-
1	$(0,0,0,0,1,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,0,0,1,0,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,0,0,1,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$\mathbf{(0,0,1,0,0,1)}$	-	-	-	-	-	**26**	-	-	-	-	-	**3**
	$(0,0,1,0,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,0,1,1,0,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,0,0,0,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,0,0,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,0,1,0,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,1,0,0,0)$	-	-	8	-	-	-	-	-	2	-	-	-
2	$(0,0,0,1,1,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,0,1,0,1,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$\mathbf{(0,0,1,1,0,1)}$	-	-	-	**39**	-	-	-	-	-	**6**	-	-
	$(0,0,1,1,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,0,0,1,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,0,1,0,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,0,1,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,1,0,0,1)$	-	-	-	-	-	28	-	-	-	-	-	3
	$(0,1,1,0,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,1,1,0,0)$	-	-	-	-	-	-	-	-	-	-	-	-
3	$\mathbf{(0,0,1,1,1,1)}$	-	-	-	-	**49**	-	-	-	-	-	**4**	-
	$(0,1,0,1,1,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,1,0,1,1)$	-	-	-	-	-	-	-	-	-	-	-	-
	$(0,1,1,1,0,1)$	-	-	-	41	-	-	-	-	-	6	-	-
	$(0,1,1,1,1,0)$	-	-	-	-	-	-	-	-	-	-	-	-
4	$\mathbf{(0,1,1,1,1,1)}$	-	**54**	-	-	51	-	-	**5**	-	-	4	-
5	$\mathbf{(1,1,1,1,1,1)}$	**57**	-	-	-	-	-	**2**	-	-	-	-	-

Table A.2.4: LOS Guided DP example.

A.3 Vehicle Routing Problem

Consider the Vehicle Routing Problem instance with $V = \{1, \ldots, 6\}$, depot $s_1 = 1$, $E = \{(v_i, v_j) : v_i, v_j \in V, v_i \neq v_j\}$ and $K = 4$ vehicles. The distance matrix is the same as in the Travelling Salesman Problem example:

$$c_{dist} = \begin{pmatrix} 0 & 1 & 6 & 26 & 19 & 24 \\ 3 & 0 & 7 & 21 & 23 & 25 \\ 31 & 9 & 0 & 22 & 27 & 20 \\ 8 & 28 & 33 & 0 & 10 & 15 \\ 30 & 5 & 34 & 12 & 0 & 16 \\ 4 & 36 & 29 & 13 & 18 & 0 \end{pmatrix}.$$

No resource constraints are considered.

We will solve this instance with the Giant Tour Representation of Section 4.2.2. The Giant Tour Representation adds $(K - 1)$ copy depot nodes, in our case $s_2 = -1$, $s_3 = -2$ and $s_4 = -3$. The new distance matrix is:

$$c'_{dist} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 6 & 26 & 19 & 24 \\ 0 & 0 & 0 & 0 & 1 & 6 & 26 & 19 & 24 \\ 0 & 0 & 0 & 0 & 1 & 6 & 26 & 19 & 24 \\ 0 & 0 & 0 & \mathbf{0} & \mathbf{1} & \mathbf{6} & \mathbf{26} & \mathbf{19} & \mathbf{24} \\ 3 & 3 & 3 & \mathbf{3} & \mathbf{0} & \mathbf{7} & \mathbf{21} & \mathbf{23} & \mathbf{25} \\ 31 & 31 & 31 & \mathbf{31} & \mathbf{9} & \mathbf{0} & \mathbf{22} & \mathbf{27} & \mathbf{20} \\ 8 & 8 & 8 & \mathbf{8} & \mathbf{28} & \mathbf{33} & \mathbf{0} & \mathbf{10} & \mathbf{15} \\ 30 & 30 & 30 & \mathbf{30} & \mathbf{5} & \mathbf{34} & \mathbf{12} & \mathbf{0} & \mathbf{16} \\ 4 & 4 & 4 & \mathbf{4} & \mathbf{36} & \mathbf{29} & \mathbf{13} & \mathbf{18} & \mathbf{0} \end{pmatrix}.$$

Any resource constraint would be copied in a similar way. The optimal solution turns out to be the same as the Travelling Salesman Problem, that is, only one vehicle is needed for the optimal solution. An alternative is Giant Tour $(1, 2, -1, 3, 4, 5, 6, -2, -3, 1)$ with a total travelled distance of 62. Two vehicles are needed and the vehicle routes are $(1, 2, 1)$ and $(1, 3, 4, 5, 6, 1)$. Note that the part $(-2, -3, 1)$ of the Giant Tour indicates that two vehicles are not used.

Appendix B

Theoretical Properties of Edge Counters

B.1 Introduction

In this appendix we will discuss some theoretical properties of edge counters which are introduced in Section 2.4.1.4. For convenience we restate the definitions of the edge counters. The edge counters track the following occurrences for each edge (see also Algorithm 2.4.1):

C_{exp} **Edge Expansion Counter**
The number of times a state is expanded by this edge, possibly improving an earlier found path, i.e. the edge is evaluated.

C_{use} **Edge Usage Counter**
The number times a state is expanded by this edge as part of an optimal path, i.e. the edge is used.

See Appendix A.2 for an example where we determine the edge counters. There we introduce the terminology of rows and columns of edge counters (matrices). Recall that we can represent edge counters by matrices. The sum of a row of an edge counter, corresponding to node $v_i \in V$, is equal to the sum of edge counters of all edges $(v_i, v_j) \in E$ with $v_j \in V$. Similarly, the sum of a column corresponding to node $v_j \in V$ is equal to the sum of edge counters of all $(v_i, v_j) \in E$ with $v_i \in V$. These sums over rows and columns are usually restricted to $\hat{E}$.

The statements in this appendix consider a complete graph $G = (V, E)$ with $s \in V$ the start node for the tour. Furthermore, G should not be a trivial instance (so $|V| > 3$). We define $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$. Furthermore, we assume that the minimisers in recursive relations (2.2.1) are unique (or that one minimiser is randomly chosen). We first prove theoretical properties of counters from the Full DP state space, followed by some remarks on the counters from the Restricted DP state space. Note that any general property must hold for all instances and must therefore be invariant under permutations of nodes (invariant under automorphisms of the graph with the starting node fixed).

B.2 Edge Counters from the Full DP State Space

When edge counters are derived from the Full DP state space we can determine bounds or even exact values of the edge counters that must hold for all instances. We will start with the edge expansion counter, which turns out to be the same for all instances of equal size. However, the edge usage counter does depend on the instance (the distance matrix and size). Nevertheless, we are able to determine (non-trivial) bounds on the values of the usage counter.

B.2.1 Edge Expansion Counter

The values of the edge expansion counter are relatively straightforward. Since we assume G is complete, we can simply count the number of states and all their out-links. This leads to the following proposition.

Proposition B.2.1.1. *Consider any complete graph $G = (V, E)$ with $s \in V$ as the start node for the tour. Let $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$ be the graph excluding the start node. We apply the Full DP method.*

After completion of stage zero (depot to first node) it holds that:

$$C_{exp}(v_i, v_j) = \begin{cases} 1 & \text{if } v_i = s, (s, v_j) \in E \\ 0 & \text{otherwise} \end{cases}.$$

After completing stage $\sigma \in \{1, \ldots, |\hat{V}| - 1\}$ we have:

$$\begin{aligned} C_{exp}(v_i, v_i) &= 0 && \forall\, v_i \in V, \\ C_{exp}(s, v_j) &= 1 && \forall\, v_j \in \hat{V}, \\ C_{exp}(v_i, s) &= 0 && \forall\, v_i \in \hat{V}, \\ C_{exp}(v_i, v_j) &= \sum_{k=1}^{\sigma} \binom{|V| - 3}{k - 1} && \forall\, (v_i, v_j) \in \hat{E}. \end{aligned}$$

Finally, after the final stage (last node to depot):

$$\begin{aligned} C_{exp}(v_i, v_i) &= 0 && \forall\, v_i \in V, \\ C_{exp}(s, v_j) &= 1 && \forall\, v_j \in \hat{V}, \\ C_{exp}(v_i, s) &= 1 && \forall\, v_i \in \hat{V}, \\ C_{exp}(v_i, v_j) &= 2^{|V|-3} && \forall\, (v_i, v_j) \in \hat{E}. \end{aligned}$$

Proof. As mentioned before, these results follow from the completeness of the graph and applying Full DP. For any edge $(v_i, v_j) \in \hat{E}$ we can count the number of states in stage σ that have (v_i, v_j) as an out-link:

$$\left| \left\{ S_{ij} \subset \hat{V} : v_i \in S_{ij}, v_j \notin S_{ij}, |S_{ij}| = \sigma \right\} \right| = \binom{|V| - 3}{\sigma - 1}.$$

When we require that a subset $S_{ij} \subset \hat{V}$ with $|S_{ij}| = \sigma$ contains v_i but not v_j, we have to select $(\sigma - 1)$ elements from the remaining $(|\hat{V}| - 2)$ elements. For each of these sets S_{ij} we have state (S_{ij}, v_i) which is expanded to v_j in stage σ. In particular, after completing Full DP we have

$$C_{exp}(v_i, v_j) = \sum_{k=1}^{|V|-2} \binom{|V|-3}{k-1} = 2^{|V|-3}.$$

The other identities are trivial. □

From Proposition B.2.1.1 we can conclude that we know the exact values of the edge expansion counter C_{exp} during the Full DP method. Note that the values only depend on the number of nodes and are equal for all graphs of equal size. Hence, these counters do not give any information over good or optimal solutions. As seen in Section 2.4.1.5, we use the edge expansion counters as normalisation weights for the edge usage counters.

Of course, we can determine the total expansion counter over rows and columns. These values are shown in the next corollary.

Corollary B.2.1.2. *Consider any complete graph $G = (V, E)$ with $s \in V$ as the start node for the tour. Let $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$ be the graph excluding the start node. After applying the Full DP method, we have:*

$$\begin{aligned}
\sum_{v_j \in \hat{V}} C_{exp}(s, v_j) &= |V| - 1, \\
\sum_{v_i \in \hat{V}} C_{exp}(v_i, s) &= |V| - 1, \\
\sum_{v_i \in \hat{V}} C_{exp}(v_i, v_j) &= (|V| - 2)2^{|V|-3} && \forall\, v_j \in \hat{V}, \\
\sum_{v_j \in \hat{V}} C_{exp}(v_i, v_j) &= (|V| - 2)2^{|V|-3} && \forall\, v_i \in \hat{V}, \\
\sum_{v_i \in \hat{V}} \sum_{v_j \in \hat{V}} C_{exp}(v_i, v_j) &= (|V| - 1)(|V| - 2)2^{|V|-3}.
\end{aligned}$$

Proof. These are direct consequences of Proposition B.2.1.1. □

B.2.2 Edge Usage Counter

Recall that the usage counter of an edge indicates the number of times a state is expanded by this edge, but only if that connection is part of an optimal path in the DP state space. Put differently, if this edge is the minimiser in recursive relations (2.2.1). The edge usage counter depends on the distance matrix of an instance. Nevertheless, we can derive general properties, see Proposition B.2.2.1.

Proposition B.2.2.1. *Consider any complete graph $G = (V, E)$ with $s \in V$ as the start node for the tour. Let $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$ be the graph excluding the start node. We apply the Full DP method.*

In any stage we can bound edge usage by edge expansion:

$$0 \leq C_{use}(v_i, v_j) \leq C_{exp}(v_i, v_j) \qquad \forall\, v_i, v_j \in V.$$

Furthermore, slightly tighter bounds exist. After completion of stage zero (depot to first node):

$$C_{use}(v_i, v_j) = \begin{cases} 1 & \text{if } v_i = s, (s, v_j) \in E \\ 0 & \text{otherwise} \end{cases}.$$

After completing stage $\sigma \in \{1, \ldots, |\hat{V}| - 1\}$, the individual edge usage counters are bounded by:

$$\begin{aligned} C_{use}(v_i, v_i) &= 0 && \forall\, v_i \in V, \\ C_{use}(s, v_j) &= 1 && \forall\, v_j \in \hat{V}, \\ C_{use}(v_i, s) &= 0 && \forall\, v_i \in \hat{V}, \\ 1 \leq C_{use}(v_i, v_j) &\leq \sum_{k=1}^{\sigma} \binom{|V| - 3}{k - 1} && \forall\, (v_i, v_j) \in \hat{E}. \end{aligned}$$

After the final stage (last node to depot) we have:

$$\begin{aligned} C_{use}(v_i, v_i) &= 0 && \forall\, v_i \in V, \\ C_{use}(s, v_j) &= 1 && \forall\, v_j \in \hat{V}, \\ \sum_{v_i \in \hat{V}} C_{use}(v_i, s) &= 1, && \\ 1 \leq C_{use}(v_i, v_j) &\leq 2^{|V|-3} && \forall\, (v_i, v_j) \in \hat{E}, \end{aligned}$$

Proof. Most results are trivial. The lower bound $1 \leq C_{use}(v_i, v_j)$ follows from state $(\{v_i, v_j\}, v_j)$, where v_i must be the origin. The upper bound on $C_{use}(v_i, v_j)$ is just the edge expansion counter. Notice that exactly one edge (v_i, s) for some $v_i \in \hat{V}$ is used in the final stage. This edge is therefore also in the optimal Hamiltonian tour by construction. □

Instances can be constructed where the both the lower and upper bound on the edge usage counter are attained by some edges. Therefore, no stronger bounds can be determined for individual edges. However, when we consider multiple edges simultaneously we can derive stronger bounds. See Proposition B.2.2.2.

Proposition B.2.2.2. *Consider any complete graph $G = (V, E)$ with $s \in V$ as the start node for the tour. Let $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$ be the graph excluding the start node. We apply the Full DP method.*

After completing stage $\sigma \in \{1, \ldots, |\hat{V}| - 1\}$, we can consider the edge usage counters for multiple edges simultaneously. By doing so, we get the following results for sums over the columns of the edge usage counter matrix.

For fixed $v_j \in \hat{V}$ we can bound the total usage of a subset of edges ending in v_j as follows:

$$\sum_{v_i \in S_j} C_{use}(v_i, v_j) \geq 2^{|S_j|} - 1 \qquad \forall S_j \in \{S \subset \hat{V} : 1 \leq |S| \leq \sigma, v_j \notin S\}, \forall v_j \in \hat{V},$$

$$\sum_{v_i \in S_j} C_{use}(v_i, v_j) \leq \sum_{k=1}^{\sigma} \binom{|V|-2}{k} - (|V| - 2 - |S_j|) \quad \forall S_j \in \{S \subset \hat{V} : 1 \leq |S| \leq \sigma, v_j \notin S\}, \forall v_j \in \hat{V}.$$

Similarly, the total usage of all edges in $\hat{E}$ ending in $v_j \in \hat{V}$ is equal to:

$$\sum_{v_i \in \hat{V}} C_{use}(v_i, v_j) = \sum_{k=1}^{\sigma} \binom{|V|-2}{k} \qquad \forall v_j \in \hat{V}.$$

For the usage of all edges in $\hat{E}$ it holds that:

$$\sum_{v_i \in \hat{V}} \sum_{v_j \in \hat{V}} C_{use}(v_i, v_j) = (|V| - 1) \sum_{k=1}^{\sigma} \binom{|V|-2}{k}.$$

Proof. Fix $\sigma \in \{1, \ldots, |\hat{V}| - 1\}$, $v_j \in \hat{V}$ and $S_j \in \{S \subset \hat{V} : 1 \leq |S| \leq \sigma, v_j \notin S\}$. For each $\emptyset \neq U \subseteq S_j$ we can consider the state $(U \cup \{v_j\}, v_j)$ which has exactly one origin in U. Hence, the contributions $\Delta C_{use}(v_i, v_j)$ of state $(U \cup \{v_j\}, v_j)$ to $C_{use}(v_i, v_j)$ with $v_i \in U$ can be bounded by

$$\sum_{v_i \in U} \Delta C_{use}(v_i, v_j) = 1.$$

This holds for each non-empty subset of S_j, thus the total contribution of these subsets is $2^{|S_j|} - 1$. The lower bound follows.

Note that equality does not hold because C_{use} also contains contributions of other subsets of $\hat{V}$ (partially) overlapping with S_j. If $\sigma = (|\hat{V}| - 1)$ and $S_j = \hat{V} \setminus \{v_j\}$ hold, no such other subsets exist and equality holds. This follows both from this proof and from the fact that the lower and upper bound are equal in this case.

For the upper bound we first prove that

$$\sum_{v_i \in \hat{V}} C_{use}(v_i, v_j) = \sum_{k=1}^{\sigma} \binom{|V|-2}{k}$$

holds. This is straightforward as it is just the number of states ending in v_j whilst considering only states up to stage σ. Each of these states ends in v_j and has some origin $v_i \in \hat{V}$.

The upper bound follows by also using the lower bound in Proposition B.2.2.1:

$$\sum_{v_i \in S_j} C_{use}(v_i, v_j) = \sum_{v_i \in \hat{V}} C_{use}(v_i, v_j) - \sum_{v_i \in \hat{V} \setminus S_j} C_{use}(v_i, v_j) \leq \sum_{k=1}^{\sigma} \binom{|V|-2}{k} - (|\hat{V}| - 1 - |S_j|).$$

Note that the minus 1 in $(|\hat{V}| - 1 - |S_j|)$ follows from the fact that $C_{use}(v_j, v_j) = 0$. □

From Proposition B.2.2.2 we can already draw a non-trivial conclusion. Using $\sigma = 2$, $|S_j| = 2$ and the lower bound

$$\sum_{v_i \in S_j} C_{use}(v_i, v_j) \geq 2^{|S_j|} - 1,$$

we see that each column (restricted to $\hat{V}$) can contain at most a single 1 after completing stage two. There exist instances where all elements in a column (again restricted to $\hat{V}$) are larger than one.

The upper bound in Proposition B.2.2.2 can be improved in the following way. By using

$$\begin{aligned} &\sum_{v_i \in S_j} C_{use}(v_i, v_j) \geq 2^{|S_j|} - 1 && \forall\, S_j \in \{S \subset \hat{V} : 1 \leq |S| \leq \sigma, v_j \notin S\}, \forall\, v_j \in \hat{V}, \\ &\sum_{v_i \in \hat{V}} C_{use}(v_i, v_j) = \sum_{k=1}^{\sigma} \binom{|V| - 2}{k} && \forall\, v_j \in \hat{V}, \end{aligned}$$

and the complement of S_j, we can get stronger upper bounds on $\sum_{v_i \in S_j} C_{use}(v_i, v_j)$. Split the complement $\hat{V} \setminus S_j$ into as few subsets with cardinality at most σ. Then apply the above lower bounds for the usage of these complement subsets together with the total usage. We have used the same technique to prove the upper bound in Proposition B.2.2.2, but only using complement subsets of cardinality one.

For completeness we also state the results when Full DP is completed, see Proposition B.2.2.3.

Proposition B.2.2.3. *Consider any complete graph $G = (V, E)$ with $s \in V$ as the start node for the tour. Let $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$ be the graph excluding the start node. We apply the Full DP method.*

After completing the final stage (last node to depot), we can consider the edge usage counters for multiple edges simultaneously. The following holds for sums over columns of the edge usage matrix:

$$\begin{aligned} &\sum_{v_i \in S_j} C_{use}(v_i, v_j) \geq 2^{|S_j|} - 1 && \forall\, S_j \in \{S \subset \hat{V} : 1 \leq |S| \leq |\hat{V}| - 1, v_j \notin S\}, \forall\, v_j \in \hat{V}, \\ &\sum_{v_i \in S_j} C_{use}(v_i, v_j) \leq 2^{|V|-2} - (|V| - 1 - |S_j|) && \forall\, S_j \in \{S \subset \hat{V} : 1 \leq |S| \leq |\hat{V}| - 1, v_j \notin S\}, \forall\, v_j \in \hat{V}, \\ &\sum_{v_i \in \hat{V}} C_{use}(v_i, v_j) = 2^{|V|-2} - 1 && \forall\, v_j \in \hat{V}, \end{aligned}$$

and in total:

$$\sum_{v_i \in \hat{V}} \sum_{v_j \in \hat{V}} C_{use}(v_i, v_j) = (|V| - 1)\left(2^{|V|-2} - 1\right).$$

Proof. These statements follow the same proof as in Proposition B.2.2.2. The upper bound is slightly rewritten:

$$\sum_{v_i \in S_j} C_{use}(v_i, v_j) \leq \left(2^{|V|-2} - 1\right) - (|V| - 2 - |S_j|) = 2^{|V|-2} - (|V| - 1 - |S_j|).$$

□

Notice that we only state bounds for columns of the edge usage matrix and none for rows. Instances can be constructed where the following trivial bounds are attained for different $v_i \in \hat{V}$. After completion of stage $\sigma \in \{1, \ldots, |\hat{V}| - 1\}$, it holds that:

$$|S_i| \leq \sum_{v_j \in S_i} C_{use}(v_i, v_j) \leq |S_i| \sum_{k=1}^{\sigma} \binom{|V|-3}{k-1} \qquad \forall\, S_i \in \{S \subseteq \hat{V} : 1 \leq |S| \leq \sigma, v_i \notin S\}, \forall\, v_i \in \hat{V},$$

$$|V| - 2 \leq \sum_{v_j \in \hat{V}} C_{use}(v_i, v_j) \leq (|V| - 2) \sum_{k=1}^{\sigma} \binom{|V|-3}{k-1} \quad \forall\, v_i \in \hat{V}.$$

Similarly, after the final stage:

$$|S_i| \leq \sum_{v_j \in S_i} C_{use}(v_i, v_j) \leq |S_i| 2^{|V|-3} \qquad \forall\, S_i \in \{S \subseteq \hat{V} : 1 \leq |S| \leq |\hat{V}| - 1, v_i \notin S\}, \forall\, v_i \in \hat{V},$$

$$|V| - 2 \leq \sum_{v_j \in \hat{V}} C_{use}(v_i, v_j) \leq (|V| - 2) 2^{|V|-3} \quad \forall\, v_i \in \hat{V}.$$

These trivial bounds are tight and correspond to the cases that an edge is never used if possible (lower bound) or always used if possible (upper bound). Take for instance very short or long distances for edges originating from a certain $v_i \in \hat{V}$.

To conclude our findings from these propositions, we see that we know the sum of each column in the edge usage matrix. The way the counters are divided between the rows is unknown in advance, but some bounds exist. We know the total edge usage of all edges, but also for all those in $\hat{E}$. Therefore, we can calculate the average of C_{use} over all $(v_i, v_j) \in \hat{E}$. After completing Full DP we get:

$$\mathbb{E}_{\hat{E}} \left[C_{use} \right] = \frac{(|V| - 1) 2^{|V|-2} - (|V| - 1)}{(|V| - 1)(|V| - 2)} = \frac{2^{|V|-2} - 1}{|V| - 2}. \tag{B.2.1}$$

This value could perhaps be used as an indication of the extremity of edge usage for individual edges. An interesting question is if this average value can be attained exactly for each edge for non-trivial instances.

B.3 Edge Counters from the Restricted DP State Space

Although the information gained from the Full DP state space is interesting from a theoretical point of view, in practice we gain that information after having solved the problem to optimality. Hence, it has little value as we have already found the optimal solution. However, there might be some applications involving uncertainty or small perturbations in the distance between nodes. In case of small perturbations, it seems justifiable to assume that the new optimal solution would be similar to the old optimal solution. That is, the usage counters (and thus the elite edges) are similar between the instances. The old elite parts can be used to guide DP for the new (perturbed) instance. We will not consider perturbations.

We will focus on Restricted DP to find solutions due to the exponential running time of Full DP. The derived results for the edge counters under Full DP are no longer valid when restricting DP. In fact, the analysis becomes quite complicated due to two factors. Firstly, the behaviour of Restricted DP and the effect on the state space must be determined. This behaviour depends on the instance (the distance between nodes) and is difficult if not impossible to generalise. Secondly, properties of the counters must be derived from the restricted state space. Carefully designed simulations could give insight to these properties.

One way to approximate properties of the state space and used edges is to consider DP where all choices are made randomly. So, all ranked lists of states and edges are just randomly sorted (see for instance Algorithm 2.4.2 for the ranked lists). We will only derive a very basic approximation for properties of the edge counters under Restricted DP. Further research must be done for counters under Restricted DP for it to be of some value. One interesting question is if we can fill in missing (or incomplete) counters based on the theoretical properties under Full DP (e.g. similar to Bayesian prior probability distribution).

B.3.1 Edge Expansion Counter

Since the edge expansion counter is less dependent on the distance matrix, we can provide an approximation on its values, assuming that each state has one in-link. See Proposition B.3.1.1. Although this is only sensible for large instances with relative small beam, it can serve as a starting point for further research.

Proposition B.3.1.1. *Consider any complete graph $G = (V, E)$ with $s \in V$ as the start node. Let $\hat{V} = V \setminus \{s\}$ and $\hat{E} = \{(v_i, v_j) \in E : v_i, v_j \in \hat{V}\}$ be the graph excluding the depot.*

We apply Restricted DP with state beam width $B = 1$, out-link number $L = |\hat{V}|$ and random ranking of states and edges. In this case we use all allowed out-links for expansion. The expected expansion counter of edges in $\hat{E}$ is

$$\mathbb{E}_{\hat{E}}\left[C_{exp}\right] = \frac{1}{2}.$$

We apply Restricted DP with state beam width $B = 1$, out-link number $0 < L < |\hat{V}|$ and random ranking of states and edges. Here the allowed out-links are limited. The expected expansion counter of edges in $\hat{E}$ is

$$\mathbb{E}_{\hat{E}}\left[C_{exp}\right] = \frac{\frac{1}{2}L(L+1) + (|V| - L - 2)L}{(|V| - 1)(|V| - 2)}.$$

Proof. In the first case all out-links are used for expansion and only one state is expanded, thus any expansion counter matrix can be written as

$$C_{exp} = P \begin{pmatrix} 0 & 1 & 1 & \dots & 1 & 1 & 1 \\ 0 & 0 & 1 & \dots & 1 & 1 & 1 \\ 0 & 0 & 0 & \dots & 1 & 1 & 1 \\ \vdots & & & \ddots & & & \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 & 0 \end{pmatrix} P^\top,$$

where P is a permutation matrix, dependent on the order of visited nodes. So, the expansion counter matrix restricted to $\hat{E}$ is a permutation of the all ones upper triangular matrix. Note that the all ones upper triangular matrix corresponds to the expansion matrix of tour $(s, v_1, \dots, v_{\hat{N}}, s)$. Therefore,

$$\sum_{e \in \hat{E}} C_{exp}(e) = \frac{1}{2}(|V|-1)(|V|-2),$$

implying

$$\mathbb{E}_{\hat{E}}\left[C_{exp}\right] = \frac{1}{|\hat{E}|} \sum_{e \in \hat{E}} C_{exp}(e) = \frac{1}{2} \frac{(|V|-1)(|V|-2)}{(|V|-1)(|V|-2)} = \frac{1}{2}.$$

In the second case, the out-links are limited and the expansion counter matrix cannot be written as a permuted matrix (the selected out-links are not consistent through DP). However, the natural expansion matrix related to tour $(s, v_1, \dots, v_{\hat{N}}, s)$ is

$$C_{exp} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

shown for $|V| = N = 8$ and $L = 3$. Thus, we have

$$\sum_{e \in \hat{E}} C_{exp}(e) = \frac{1}{2}L(L+1) + (|V| - L - 2)L,$$

which completes the proof. Setting $L = |\hat{V}| = |V| - 1$ gives the same result as the first case. □

As an approximation for larger state beams B, we can assume that the expanded states are independent in the sense that their expansions do not coincide. This gives a linear relation:

$$\mathbb{E}_{\hat{E}}\left[C_{exp}\right] \approx \frac{\frac{1}{2}L(L+1) + (|V| - L - 2)L}{(|V|-1)(|V|-2)} B.$$

The assumption of independent paths fails under Restricted DP with more realistic beam widths, especially since we do not use random ranking. A simple upper bound for the total of the expansion counter under Restricted DP with beam width B and out-link number L is

$$\sum_{v_i \in \hat{V}} \sum_{v_j \in \hat{V}} C_{exp}(v_i, v_j) \leq BL(|V| - 1).$$

This holds since there are $(|V| - 1)$ stages involving the edges in $\hat{E}$ and in each stage up to B states are expanded by up to L out-links. More useful bounds should be determined.

B.3.2 Edge Usage Counter

For edge usage counters under Restricted DP we have no estimates or bounds. For completeness we only state the following. Under the same conditions as Proposition B.3.1.1, it holds that $\mathbb{E}_{\hat{E}}[C_{use}] = \mathbb{E}_{\hat{E}}[C_{exp}]$. As only one state is expanded per stage, all states have exactly one in-link. Therefore, the usage and expansion counter are equal and we cannot derive a similar estimate as Equation (B.2.1) with these results.

B.4 Conclusion

For a complete graph we can derive several bounds for the edge counters if these are obtained from the (exact) Full DP method. Because we assume that the graph is complete, the edge expansion counter is the same for all instances of the same size. The edge usage counter does depend on the instance, in particular on the distance matrix.

For further research on the exact edge counters we suggest to look into edge counters obtained with Full DP on the reversed graph. That is, if we reverse all edges in the graph, how does this affect the edge counters and is there a relation with the normal edge counters? In the case of symmetric instances, the new edge counters are the same as the normal counters. Can we extract additional information from this result?

The edge scores can also be seen as an alternative distance matrix. By changing the sign of the edge scores, we can solve a Travelling Salesman Problem that maximises the score of a tour. How does this optimal score tour relate to the optimal distance tour?

Finally, little is known on the edge counters under Restricted DP. A possible start is to investigate random selection of states to expand and random selection of out-links, as this is independent of the distance matrix. Properties of random paths that can intersect are required. If successful, can we use the results for Full DP and Restricted DP counters to improve the quality (reliability) of the edge counters?

Appendix C

Numerical Results for TSP

C.1 Introduction

This appendix lists the generated Travelling Salesman Problem instances used to perform the parameter analysis for the prototypes. The results of this analysis are given below, followed by the results for the literature benchmarks. The used methodology is described in Chapter 3.

The specifications of the system are: 2.60GHz Intel Core i7-3720QM processor, 8GB RAM and a Windows 7 64-bit operating system. Used software are: Matlab 2007b, version 7.5.0.342 and Visual Studio 2010 Professional, version 10.0.40219.1 SP1Rel. The Dynamic Programming algorithms are implemented in C++.

C.2 Generated Instances

C.2.1

Generated Instances

15 Customers

C001-15		**C002-15**		**C003-15**		**C004-15**		**C005-15**		**C006-15**		**C007-15**		**C008-15**		**C009-15**		**C010-15**	
(209.2328)		(217.2880)		(119.5418)		(273.7785)		(179.4402)		(185.2547)		(129.4625)		(155.7445)		(125.8463)		(184.4049)	
52	59	40	64	33	18	22	20	28	6	2	46	23	46	38	68	6	24	33	61
55	65	28	63	35	25	24	18	31	11	4	55	22	44	40	78	26	10	37	55
62	62	26	58	35	19	24	19	27	6	16	7	23	49	42	54	28	1	44	58
14	63	31	58	32	20	30	24	30	9	30	21	22	47	40	60	33	12	34	58
8	63	30	68	44	23	34	20	15	57	30	2	27	47	7	45	28	14	37	49
10	68	19	34	32	25	41	70	1	54	21	12	9	55	4	42	38	2	48	54
12	81	21	33	33	17	42	72	48	63	23	5	25	48	1	39	30	1	17	63
5	80	18	33	42	17	47	68	19	28	38	8	26	59	36	50	34	11	17	62
15	63	19	77	49	18	42	71	10	30	52	3	14	38	40	56	27	6	15	52
1	64	19	71	35	17	38	80	23	26	39	6	28	56	47	62	26	11	19	53
6	77	64	8	40	1	80	86	19	23	38	12	19	51	42	53	29	8	15	49
12	79	71	6	40	19	89	72	22	26	1	13	11	56	35	52	24	9	27	45
15	74	67	12	22	15	90	85	20	24	43	14	52	51	32	50	43	9	30	51
17	25	69	1	15	13	83	79	22	16	33	18	1	39	36	61	46	12	28	32
37	67	65	7	8	11	1	76	17	31	45	14	7	43	42	55	45	10	1	1
39	77	59	4	17	12	12	48	24	35	36	20	8	41	31	30	48	25	1	2

C011-15		**C012-15**		**C013-15**		**C014-15**		**C015-15**		**C016-15**		**C017-15**		**C018-15**		**C019-15**		**C020-15**	
(139.9877)		(167.8814)		(99.2978)		(79.6704)		(107.8993)		(135.4801)		(124.8658)		(100.6606)		(159.1617)		(146.7207)	
9	17	52	25	25	1	10	28	30	33	45	30	12	31	15	20	53	27	30	53
14	15	49	23	19	3	10	17	20	36	38	27	9	27	28	19	59	18	38	60
13	15	40	37	27	28	16	24	17	26	31	23	13	25	25	19	49	22	35	48
10	17	46	28	29	12	10	15	9	32	23	20	5	33	22	34	55	31	34	41
5	11	43	25	24	11	14	8	9	29	32	36	11	21	23	16	63	29	40	35
41	3	47	22	15	4	7	23	20	15	28	31	11	30	23	20	58	24	31	55
43	7	48	19	22	13	17	12	27	33	36	29	1	22	24	18	53	20	57	30
53	13	49	31	17	9	23	8	17	28	39	30	4	36	20	12	19	45	61	13
6	27	8	59	25	2	26	7	11	32	51	3	12	37	22	12	11	38	56	1
2	29	6	58	18	17	22	5	5	30	59	18	15	33	24	27	16	33	59	17
1	24	1	57	30	7	19	10	15	48	54	4	9	29	36	27	14	40	63	13
6	29	8	48	25	3	24	9	19	39	56	1	6	28	27	30	15	34	51	10
8	14	12	57	12	27	22	2	1	25	38	23	9	32	24	1	19	46	59	33
6	30	32	9	19	7	21	12	8	27	16	16	33	23	18	8	1	29	47	41
4	30	31	4	13	21	21	14	14	39	20	18	48	25	14	6	14	25	43	31
9	34	35	9	7	20	28	10	16	34	20	22	38	21	22	6	26	21	38	42

C021-15		**C022-15**		**C023-15**		**C024-15**		**C025-15**		**R001-15**		**R002-15**		**R003-15**		**R004-15**		**R005-15**	
(134.8202)		(155.6807)		(83.6895)		(170.1163)		(173.8742)		(71.8771)		(69.4384)		(85.5708)		(79.7886)		(79.7724)	
1	36	4	20	25	1	2	2	67	42	23	21	19	20	35	13	11	21	8	22
12	25	11	20	19	10	1	22	62	47	16	26	20	33	23	22	14	16	8	18
12	44	8	25	19	18	10	10	64	50	14	23	17	21	23	15	20	24	7	8
12	32	8	33	24	19	2	13	63	44	9	24	18	19	32	21	10	25	14	10
11	18	1	25	31	17	8	21	61	44	18	33	15	24	29	12	16	15	2	18
8	27	12	17	30	12	3	10	43	18	15	32	14	20	24	11	23	19	22	12
6	29	5	21	27	22	4	20	36	23	13	17	10	24	23	8	13	15	11	12
12	26	43	29	21	31	30	23	52	24	11	21	19	21	33	13	12	12	5	21
13	27	52	34	33	23	34	28	44	18	18	17	8	25	19	6	19	16	10	16
8	24	44	37	23	22	36	18	50	1	14	18	21	19	28	16	1	15	1	19
17	31	48	39	26	16	35	18	20	27	1	25	1	28	28	21	18	20	5	18
7	28	45	30	22	27	43	23	34	19	10	29	16	15	22	16	14	13	11	24
5	28	58	15	28	21	41	20	15	23	13	30	12	28	24	13	17	13	11	25
41	17	43	23	19	16	35	55	26	18	20	34	12	23	14	18	18	6	15	28
29	8	51	17	27	21	31	12	24	13	11	18	15	23	21	10	18	25	2	16
34	2	47	20	23	23	41	16	16	23	20	22	13	18	12	1	19	19	10	31

Table C.2.1: Instances with 15 customers (1 of 2). Each row corresponds to the 2D coordinates of a node. The first node is the depot, the other nodes are the customers. The optimal travelled distance is shown within brackets.

R006-15		**R007-15**		**R008-15**		**R009-15**		**R010-15**		**R011-15**		**R012-15**		**R013-15**		**R014-15**		**R015-15**	
(79.6270)		(55.3060)		(60.5045)		(80.8353)		(75.8955)		(54.7658)		(91.1274)		(60.5398)		(75.1052)		(74.0696)	
17	24	13	6	5	14	1	8	5	9	4	9	12	18	20	16	11	19	10	27
15	24	13	10	9	5	11	27	15	21	2	8	10	22	21	13	19	24	11	20
23	13	14	1	5	5	6	23	14	8	9	17	11	16	22	1	15	20	2	22
14	26	10	13	12	8	9	20	12	14	9	5	10	27	30	8	20	23	16	29
15	26	2	6	6	8	6	19	15	16	10	1	13	29	31	12	15	16	18	21
23	12	13	3	15	8	3	20	17	13	7	14	6	17	23	7	15	22	10	31
17	19	7	7	7	22	13	13	14	9	11	7	22	34	28	10	18	1	1	31
1	30	15	14	1	11	17	16	12	5	9	6	16	26	20	13	22	13	15	19
20	26	10	1	6	14	26	18	5	1	6	9	15	21	25	7	11	15	17	21
3	18	13	9	16	12	15	22	16	6	8	10	1	25	13	14	11	16	20	33
19	25	5	10	16	4	6	20	10	15	18	9	11	28	21	1	17	27	23	24
13	21	18	7	15	6	13	14	19	6	8	16	12	8	25	6	10	14	10	21
14	18	12	8	6	17	19	15	19	2	14	13	6	20	22	16	9	25	9	21
9	33	9	5	4	7	6	11	25	8	10	6	8	4	24	7	9	17	3	26
17	20	17	5	10	7	11	16	18	11	13	11	11	10	20	14	11	21	17	23
2	27	7	12	11	16	13	16	11	14	11	4	15	13	26	13	16	24	10	24

R016-15		**R017-15**		**R018-15**		**R019-15**		**R020-15**		**R021-15**		**R022-15**		**R023-15**		**R024-15**		**R025-15**	
(74.4260)		(62.3025)		(79.3086)		(76.7599)		(64.4498)		(75.8491)		(75.0825)		(70.1006)		(74.6108)		(59.8296)	
17	27	5	20	12	5	13	12	16	10	15	14	12	15	6	18	2	21	14	11
24	24	13	29	17	7	10	21	22	1	17	6	12	17	12	14	12	13	13	12
24	11	14	22	25	6	15	16	21	14	14	13	13	8	17	23	18	18	5	19
23	28	4	17	22	14	15	17	20	15	17	15	16	26	7	7	19	9	12	8
21	10	11	20	20	13	1	28	18	19	19	7	14	16	1	13	8	19	11	13
12	7	10	13	30	1	11	26	28	8	19	12	4	17	5	12	14	12	8	9
14	12	1	13	18	10	18	22	22	15	20	28	1	23	15	3	13	4	13	19
19	16	13	15	14	12	12	23	24	17	15	18	3	31	11	12	17	15	8	20
13	9	12	19	25	24	16	15	18	10	15	10	8	25	3	17	17	18	12	19
25	20	13	21	28	18	14	23	13	14	16	7	10	15	13	11	10	12	8	15
15	20	4	19	16	15	26	13	11	21	14	6	11	27	10	14	13	19	1	25
19	18	4	31	27	5	21	9	18	16	12	25	9	17	9	13	19	17	9	17
23	1	13	19	25	2	8	18	30	13	24	1	13	29	9	5	13	16	16	16
24	18	11	13	26	2	7	22	19	9	14	17	9	20	5	20	11	23	4	12
18	19	9	30	13	10	9	26	19	15	17	26	7	21	9	2	7	8	9	16
23	25	9	19	21	16	15	20	11	20	13	4	16	24	6	9	11	1	11	14

Table C.2.2: Instances with 15 customers (2 of 2). Each row corresponds to the 2D coordinates of a node. The first node is the depot, the other nodes are the customers. The optimal travelled distance is shown within brackets.

C.2.2

Generated Instances

25 Customers

C001-25		C002-25		C003-25		C004-25		C005-25		C006-25		C007-25		C008-25		C009-25		C010-25	
(345.2361)		(364.6851)		(240.1857)		(433.3800)		(272.9364)		(470.5267)		(247.4525)		(252.8773)		(301.1792)		(356.0261)	
27	33	81	108	20	3	8	17	69	87	25	143	41	26	42	23	82	33	18	36
37	41	76	123	30	6	3	13	70	82	29	151	54	17	46	23	87	26	5	37
27	36	103	80	26	4	2	21	64	82	11	125	48	19	51	22	78	36	31	74
9	93	104	94	18	9	10	26	72	90	31	150	60	8	42	35	88	30	43	65
7	90	108	81	48	3	1	12	78	80	90	65	52	6	48	33	30	31	38	57
1	100	118	88	48	9	11	17	52	66	114	63	46	26	42	27	17	26	36	71
5	99	101	102	8	42	69	63	50	60	115	72	50	10	50	33	91	76	60	1
44	46	106	97	8	52	69	49	44	71	102	58	47	12	68	32	78	20	57	7
43	14	109	69	9	42	47	55	37	63	112	71	56	10	64	43	69	27	98	59
39	22	112	76	13	40	104	53	56	91	105	57	65	1	70	34	70	12	106	74
40	17	109	71	6	44	108	63	67	88	109	68	62	8	57	53	50	15	98	78
35	22	104	66	7	42	110	73	64	87	105	61	77	50	59	51	68	1	99	71
36	16	116	67	8	51	114	61	63	42	119	125	62	51	53	46	70	13	98	67
65	60	1	39	1	42	107	63	73	52	104	117	63	44	64	50	73	25	94	75
58	56	1	40	4	45	111	67	57	19	97	114	60	55	57	50	73	11	109	74
95	92	15	49	42	12	114	62	54	28	99	110	62	55	61	55	61	8	51	46
105	94	35	49	46	13	52	106	60	10	108	120	66	66	62	56	64	19	58	38
101	102	38	41	47	15	49	111	49	24	114	1	56	54	60	49	65	18	97	87
94	91	87	97	45	7	56	70	57	17	114	6	66	72	19	1	68	17	22	26
96	87	91	114	49	19	50	65	53	11	121	4	70	59	9	16	71	13	13	30
42	46	80	112	44	13	58	63	65	1	101	133	32	22	16	7	76	2	15	36
26	47	90	110	27	47	80	44	63	11	102	142	32	16	12	8	35	62	21	42
31	52	88	112	28	45	84	43	55	4	15	124	29	29	18	14	34	75	23	43
40	40	65	64	21	73	140	93	71	4	13	118	19	16	19	8	42	55	23	29
26	42	56	69	32	78	123	70	65	17	63	73	30	29	16	13	35	59	21	28
37	49	68	64	35	78	108	73	70	10	71	71	26	11	41	78	34	67	15	43

C011-25		C012-25		C013-25		C014-25		C015-25		C016-25		C017-25		C018-25		C019-25		C020-25	
(457.5132)		(357.1914)		(360.0444)		(295.7747)		(331.7468)		(487.5829)		(425.9180)		(321.6518)		(447.4489)		(323.9592)	
74	77	123	13	54	64	99	74	52	55	90	66	85	115	57	1	7	68	8	84
73	78	121	15	56	61	93	66	60	52	85	70	85	107	67	12	13	60	13	80
77	79	115	1	49	89	67	31	55	62	90	55	10	107	66	40	11	69	14	93
16	94	123	18	50	79	67	17	53	44	93	69	12	116	63	49	14	66	12	82
2	103	75	82	47	64	65	21	50	61	121	7	5	121	61	53	10	52	8	58
20	90	36	92	60	65	70	19	48	50	116	18	15	109	69	47	12	69	10	67
5	84	36	91	49	69	79	64	45	59	115	4	1	117	65	38	16	60	8	66
1	89	30	95	49	56	70	65	23	2	120	20	6	119	55	42	3	20	19	56
139	90	39	82	52	61	74	63	25	6	70	108	12	119	63	39	1	24	9	59
93	103	34	93	58	61	62	52	26	1	72	102	37	67	67	39	75	57	12	51
112	114	41	93	1	69	68	63	17	15	143	91	71	94	61	44	72	51	13	66
101	93	23	95	15	59	94	84	31	7	139	94	130	19	17	76	78	68	11	65
39	62	35	96	7	44	94	79	27	15	138	88	136	24	30	89	68	48	7	63
35	77	33	88	107	78	95	73	17	7	5	123	131	25	59	21	108	18	1	58
90	30	109	16	99	75	106	82	28	25	148	80	124	14	56	52	138	112	98	36
92	32	100	14	104	75	101	76	21	4	156	90	128	19	58	55	141	105	90	44
9	79	101	18	96	81	38	31	27	20	5	128	132	72	10	14	140	106	93	38
17	74	109	12	46	19	34	37	45	51	7	137	133	47	7	9	17	52	60	56
15	74	98	11	49	20	36	33	44	87	15	123	132	48	17	19	13	58	60	47
18	80	26	63	38	36	71	24	38	95	1	137	129	59	9	22	15	51	56	56
9	70	17	66	40	22	71	16	44	34	85	87	130	47	14	12	13	48	57	39
13	78	13	76	35	28	79	29	86	109	80	90	7	101	20	23	78	74	53	38
15	77	20	69	37	45	80	24	85	111	91	92	21	111	20	25	78	81	56	44
14	87	18	85	40	27	76	19	82	112	86	95	19	106	9	12	77	62	57	48
18	70	14	69	12	7	24	6	94	107	89	81	20	116	13	20	74	80	56	35
148	26	4	74	23	6	23	1	93	113	79	79	15	112	8	19	79	74	10	16

Table C.2.3: Instances with 25 customers (1 of 3). Each row corresponds to the 2D coordinates of a node. The first node is the depot, the other nodes are the customers. The best-known travelled distance with Restricted DP is shown within brackets. The best-known travelled distance for C018-25 is 290.6854.

C021-25		**C022-25**		**C023-25**		**C024-25**		**C025-25**		**R001-25**		**R002-25**		**R003-25**		**R004-25**		**R005-25**	
(288.9252)		(266.0786)		(295.0417)		(322.7863)		(369.1406)		(76.9925)		(85.9541)		(106.3055)		(99.2573)		(82.9911)	
73	23	33	29	51	16	108	31	18	102	19	8	15	23	16	15	9	25	4	16
67	30	24	32	1	58	99	38	21	97	23	19	19	5	12	14	7	10	17	18
84	73	27	38	6	69	102	41	21	101	17	13	14	11	12	11	10	21	13	16
78	78	20	40	12	62	94	9	7	97	18	8	9	5	14	19	4	18	12	28
82	86	30	51	7	60	94	5	12	88	21	6	16	11	19	15	2	18	7	14
80	88	79	9	12	71	32	54	5	117	28	13	15	10	20	29	9	21	10	10
89	67	78	16	10	64	34	52	9	118	18	14	12	16	18	17	5	22	10	19
73	73	86	1	11	63	37	46	9	114	17	8	18	22	21	9	7	24	17	20
72	29	77	11	13	68	33	49	7	116	13	13	7	8	10	14	14	31	8	29
69	21	82	8	8	60	30	45	14	119	10	13	20	10	16	11	14	23	15	18
71	24	37	25	85	59	33	42	1	115	25	10	7	13	22	27	2	24	14	18
57	45	32	14	77	55	39	40	58	116	21	18	22	15	13	9	8	25	15	23
65	73	35	17	79	48	30	42	61	116	9	2	24	15	12	15	7	13	14	12
42	88	15	40	83	55	39	46	48	126	20	8	24	13	4	20	13	14	18	18
32	84	15	31	73	84	25	54	58	120	21	13	9	13	17	14	4	38	1	20
23	92	22	38	83	85	41	51	66	25	21	17	20	5	22	13	3	28	17	21
79	12	15	38	88	82	35	42	50	25	20	20	17	11	15	12	5	31	17	26
77	13	11	38	84	80	30	55	44	31	11	9	17	5	7	13	8	19	6	8
84	12	15	35	74	86	66	11	51	28	16	1	15	2	17	15	2	33	20	17
80	13	86	53	73	78	70	3	48	37	18	5	12	12	9	22	17	24	11	15
82	6	69	53	77	76	65	7	48	33	24	6	13	21	12	16	1	17	3	24
89	14	85	54	83	82	1	84	110	88	20	11	11	1	16	13	6	23	4	11
89	10	85	57	66	54	64	54	98	82	12	1	8	20	17	17	4	12	10	28
77	15	74	30	70	47	52	59	83	26	17	6	14	12	8	3	5	35	12	15
83	11	87	21	67	53	50	53	80	27	20	16	19	4	17	1	19	17	18	20
82	1	83	13	72	62	58	49	90	28	26	13	11	7	14	8	6	20	20	15

R006-25		**R007-25**		**R008-25**		**R009-25**		**R010-25**		**R011-25**		**R012-25**		**R013-25**		**R014-25**		**R015-25**	
(76.3670)		(93.8313)		(93.6213)		(110.5255)		(98.9248)		(142.7791)		(80.4524)		(95.6564)		(101.7864)		(126.7721)	
14	6	8	19	6	8	17	9	9	14	25	22	9	19	11	15	26	1	39	20
5	10	10	23	3	8	20	21	6	14	19	31	7	13	17	23	21	7	41	19
10	17	3	26	10	20	24	9	5	5	18	38	7	15	13	17	19	15	33	20
17	8	14	27	11	16	18	26	11	4	20	20	16	18	17	27	28	9	30	30
6	20	20	12	10	15	15	15	9	21	14	25	10	9	15	18	27	10	36	25
9	6	8	18	20	14	19	15	10	27	18	26	3	16	19	30	23	14	19	26
20	11	15	12	3	14	14	16	2	19	6	22	13	16	12	1	32	9	28	39
5	16	14	17	7	16	22	1	13	6	24	26	1	17	22	10	29	8	38	1
18	13	11	24	8	12	12	8	6	11	30	14	11	24	8	17	25	14	33	30
14	11	17	14	2	10	7	24	9	8	14	33	11	15	23	16	20	13	37	21
9	23	8	17	2	13	19	23	1	23	24	21	7	10	16	9	18	2	36	22
9	20	14	14	15	27	12	19	6	17	14	30	9	25	17	7	25	11	35	34
7	2	20	27	13	14	26	14	10	22	17	33	19	11	11	18	19	19	35	19
8	13	14	22	15	19	21	13	11	10	23	28	12	13	21	13	24	9	33	31
11	21	14	21	17	12	17	14	2	21	22	33	11	22	29	13	43	10	33	14
10	1	1	31	11	10	28	15	17	7	10	23	5	13	17	25	33	3	28	21
7	8	10	26	6	11	20	18	1	11	19	14	12	16	17	21	31	12	38	35
12	11	17	22	14	17	28	9	9	12	13	10	9	12	26	26	28	16	37	25
10	10	4	17	1	23	25	18	6	10	13	28	15	12	23	23	25	6	37	22
11	15	15	16	13	8	18	17	3	16	18	23	9	18	18	11	32	14	32	23
12	17	4	12	8	18	13	16	9	18	1	34	12	25	17	11	31	8	40	24
9	7	10	9	6	16	26	20	17	5	14	35	10	12	27	18	24	14	38	14
8	17	7	19	12	18	30	5	16	23	10	27	8	12	13	11	22	18	34	13
8	3	3	29	3	12	31	9	8	17	19	22	15	8	22	15	31	19	26	15
11	17	16	22	7	17	19	14	16	17	29	39	9	5	22	21	33	18	42	16
7	13	12	19	17	27	8	16	11	14	20	29	14	16	17	24	37	4	31	26

Table C.2.4: Instances with 25 customers (2 of 3). Each row corresponds to the 2D coordinates of a node. The first node is the depot, the other nodes are the customers. The best-known travelled distance with Restricted DP is shown within brackets.

R016-25		**R017-25**		**R018-25**		**R019-25**		**R020-25**		**R021-25**		**R022-25**		**R023-25**		**R024-25**		**R025-25**	
(120.5333)		(107.7712)		(104.6819)		(96.2795)		(79.9395)		(89.4408)		(94.9247)		(87.2121)		(104.0333)		(96.8526)	
23	12	16	23	6	22	7	10	7	12	12	20	32	16	19	8	13	16	20	36
13	15	23	6	14	18	2	18	13	1	12	22	27	10	14	15	1	8	19	22
18	16	22	17	12	13	8	26	17	14	21	12	23	13	6	18	12	5	11	24
1	11	25	22	16	23	14	13	14	21	6	20	18	8	10	11	20	9	11	31
8	8	16	21	15	19	5	11	15	14	14	19	21	15	15	15	10	12	9	26
9	9	26	16	13	13	3	21	17	18	11	1	27	12	23	9	18	17	4	22
10	15	4	10	13	15	7	18	15	11	16	13	15	13	7	15	12	13	22	21
23	13	10	10	14	20	11	19	20	17	9	15	29	21	3	13	13	23	13	23
13	9	11	5	12	16	8	16	13	5	9	16	18	15	13	8	14	11	7	29
19	13	12	6	11	21	8	21	19	10	17	16	19	17	14	22	18	13	13	21
16	7	13	16	18	9	1	6	9	1	8	20	22	14	2	15	10	16	22	15
14	17	14	1	1	16	15	14	15	15	8	15	22	20	11	11	26	30	19	32
20	8	16	6	7	15	10	21	17	11	18	12	21	13	4	19	13	21	13	20
11	8	16	10	10	32	14	5	19	13	15	13	18	14	14	17	13	15	22	20
12	21	4	21	19	22	14	3	20	4	10	23	22	7	15	18	11	9	15	19
14	6	17	10	8	20	4	10	13	11	16	18	32	4	1	8	4	9	23	24
5	4	18	20	17	21	8	7	19	5	10	10	18	17	5	7	16	25	17	22
18	20	18	3	26	14	12	9	13	6	11	16	19	20	3	9	16	5	22	26
21	1	12	2	9	17	17	21	10	4	6	16	22	8	5	17	11	7	14	25
12	26	11	14	18	19	14	16	23	9	16	12	16	20	9	14	20	15	11	25
3	19	19	16	13	37	2	11	18	11	21	17	27	18	19	7	7	18	12	25
16	16	20	23	14	23	17	3	12	13	20	25	24	7	11	10	13	22	30	21
19	12	10	21	18	18	11	22	13	14	13	9	29	17	13	3	16	11	10	28
22	29	17	12	10	29	15	1	19	11	20	12	18	12	11	7	10	10	11	22
25	16	16	1	10	21	10	11	17	19	11	8	22	12	7	12	17	24	1	26
12	13	7	16	8	14	13	23	20	11	19	15	15	1	16	11	8	17	7	25

Table C.2.5: Instances with 25 customers (3 of 3). Each row corresponds to the 2D coordinates of a node. The first node is the depot, the other nodes are the customers. The best-known travelled distance with Restricted DP is shown within brackets.

C.3

Parameter Analysis

for

Prototypes

C.3.1

Two-Phase DP Prototype

under

Optimal Conditions

15 Customers

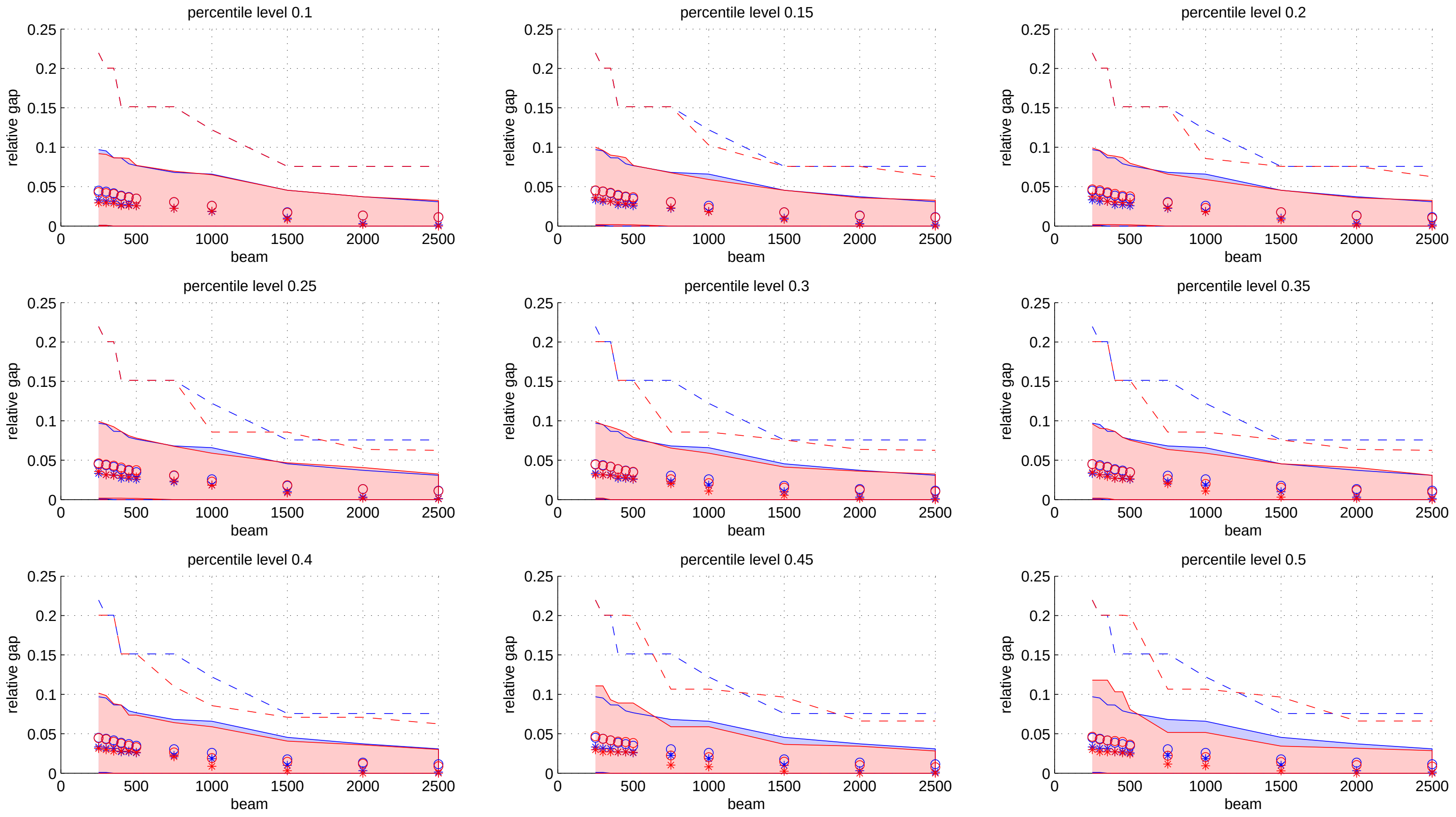


Figure C.3.1: Relative gap from optimality for different beam widths for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Instances with 15 customers.

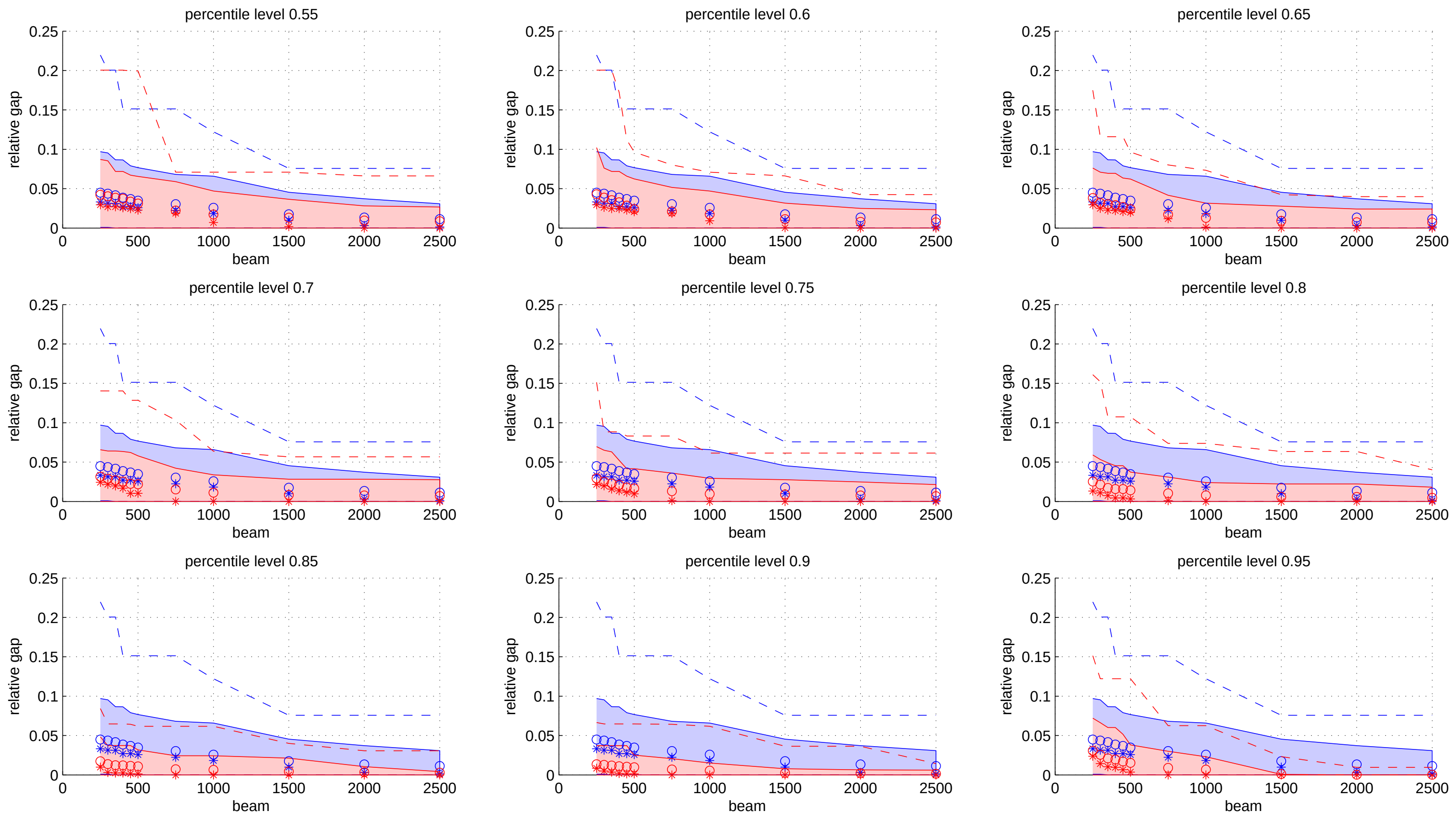


Figure C.3.2: Relative gap from optimality for different beam widths for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Instances with 15 customers.

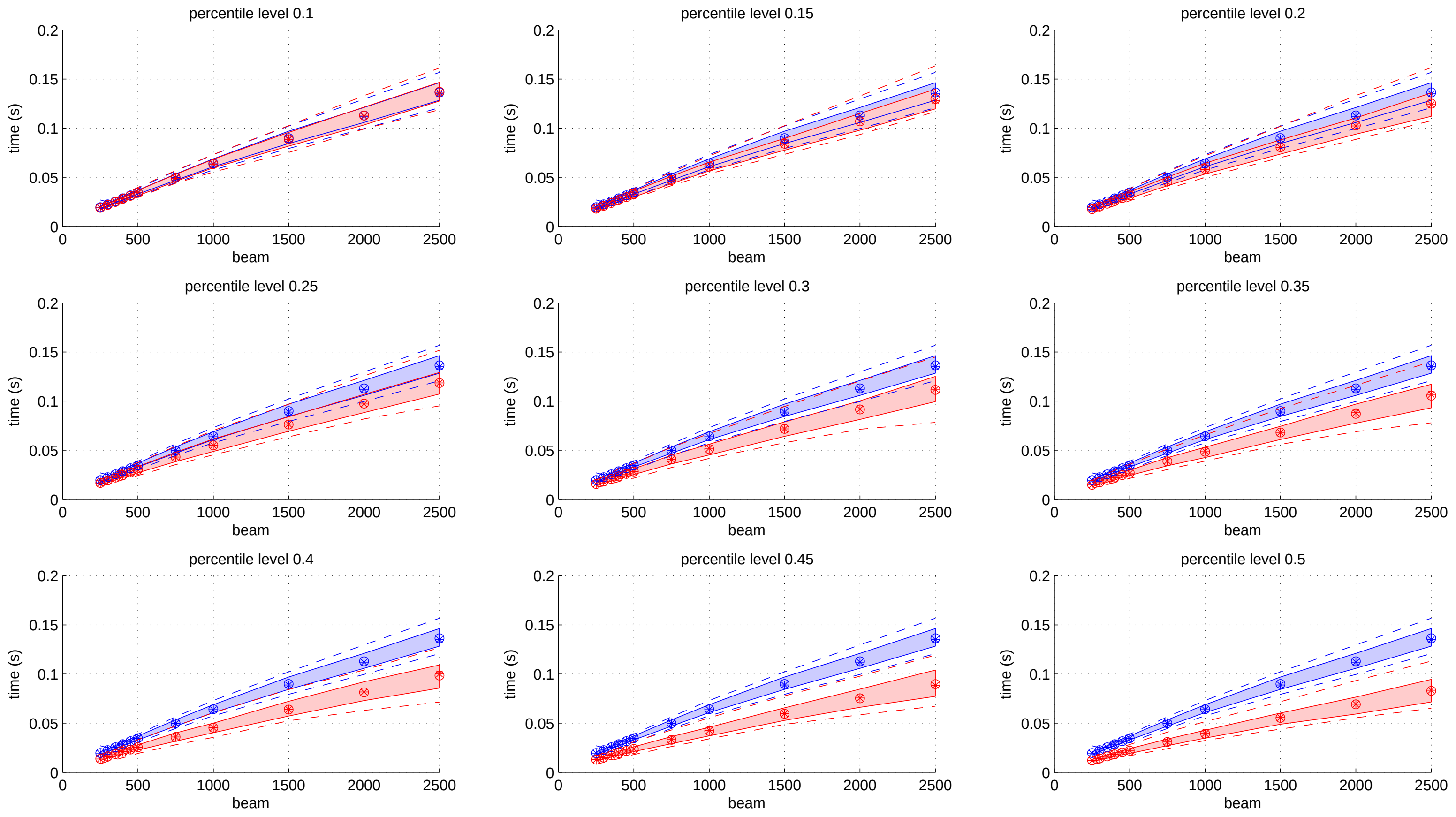


Figure C.3.3: Computational time for different beam widths for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Computational time for determining the exact counters is ignored. Instances with 15 customers.

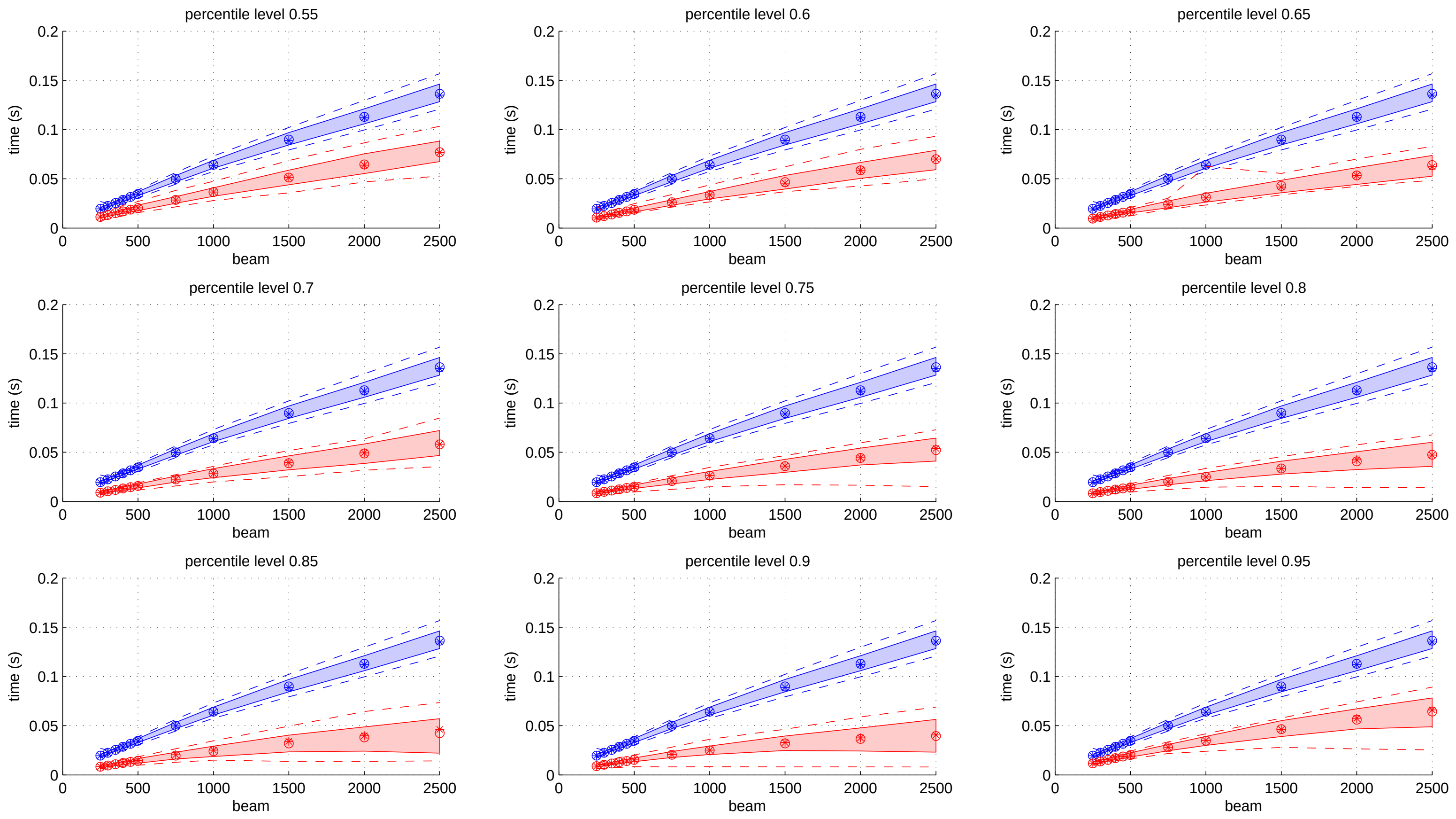


Figure C.3.4: Computational time for different beam widths for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Computational time for determining the exact counters is ignored. Instances with 15 customers.

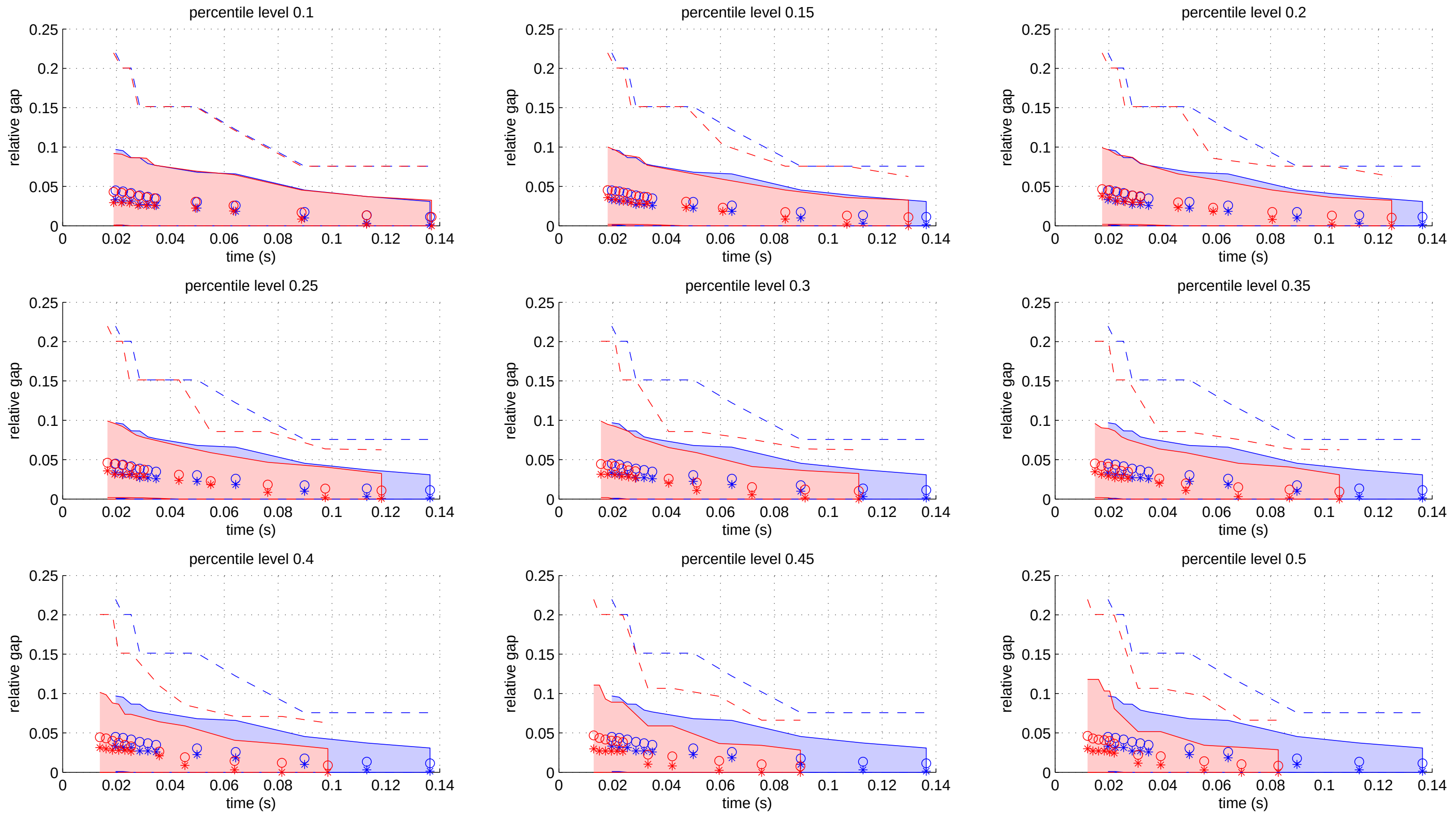


Figure C.3.5: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Computational time for determining the exact counters is ignored. Instances with 15 customers.

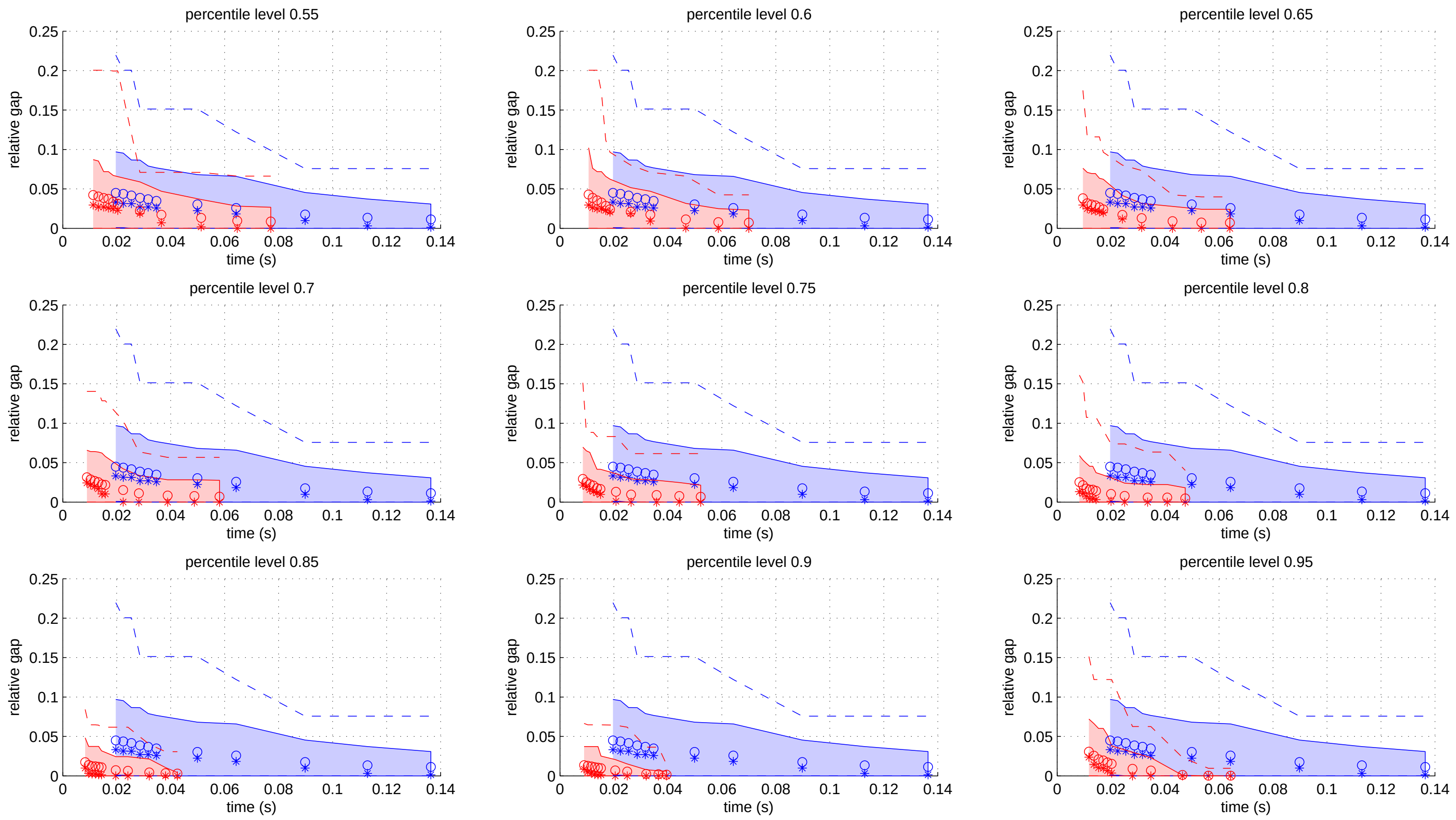


Figure C.3.6: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Computational time for determining the exact counters is ignored. Instances with 15 customers.

C.3.2

Two-Phase DP Prototype under Optimal Conditions

25 Customers

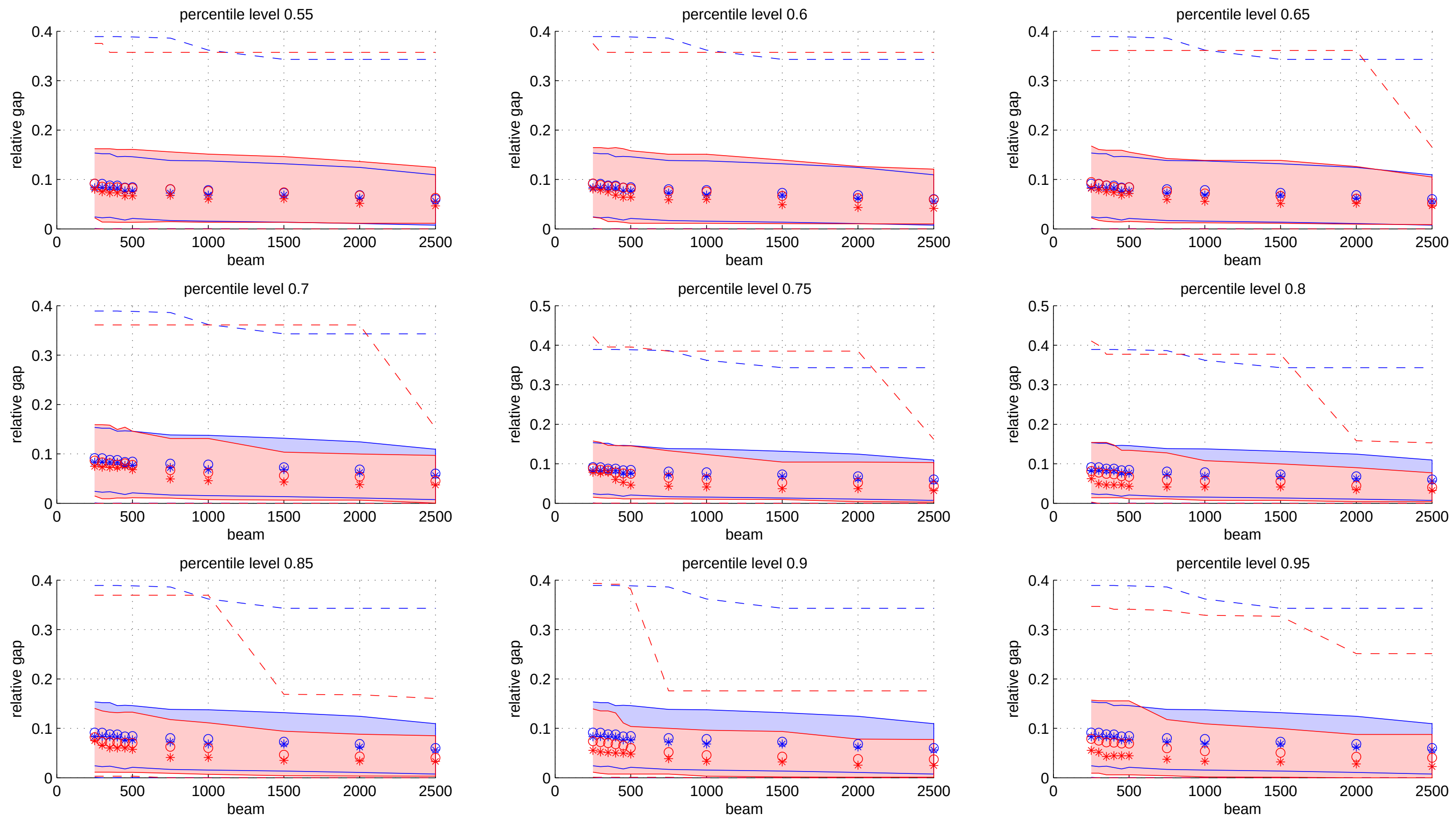


Figure C.3.7: Relative gap from optimality for different beam widths for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Instances with 25 customers.

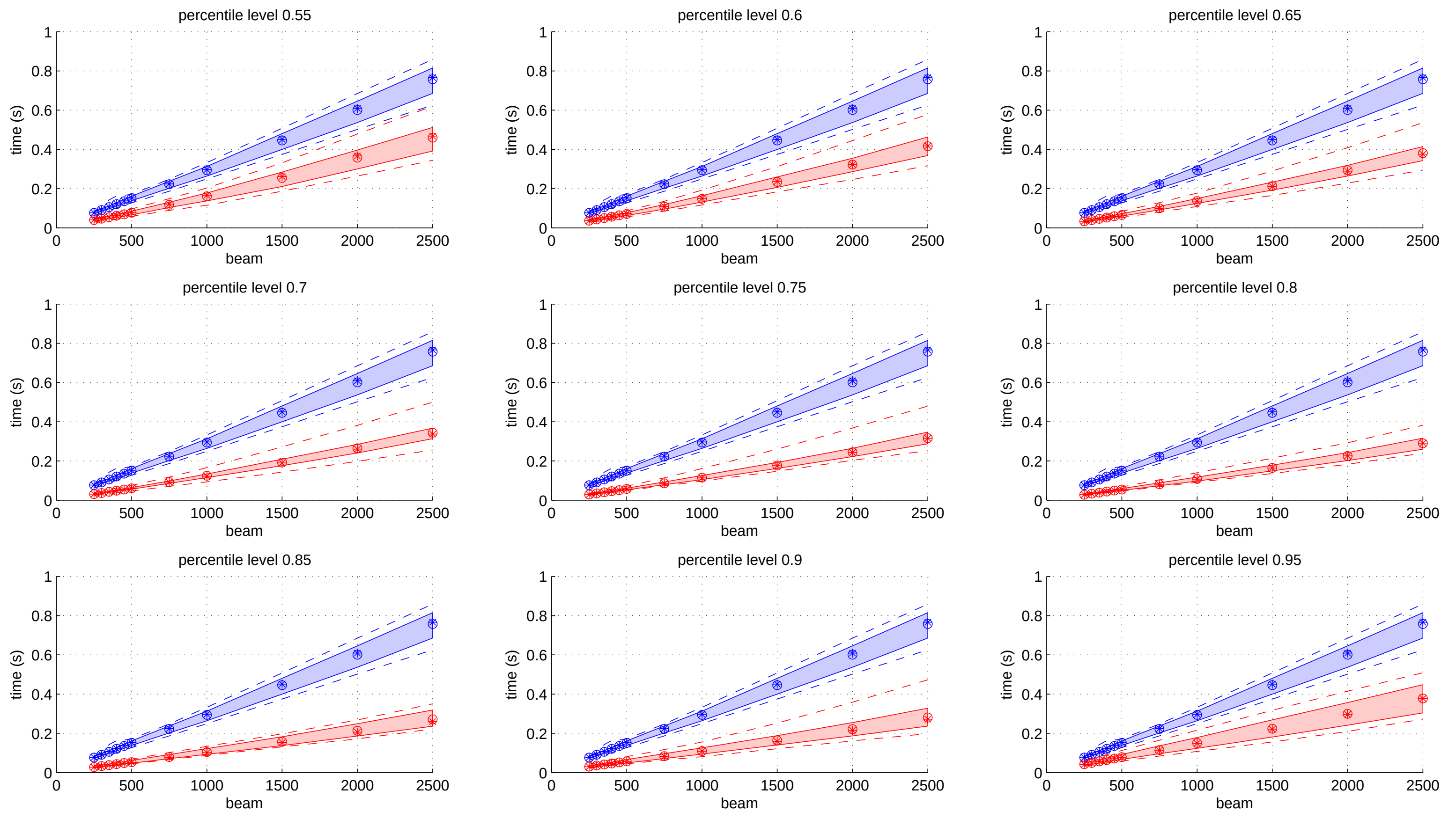


Figure C.3.8: Computational time for different beam widths for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Computational time for determining the exact counters is ignored. Instances with 25 customers.

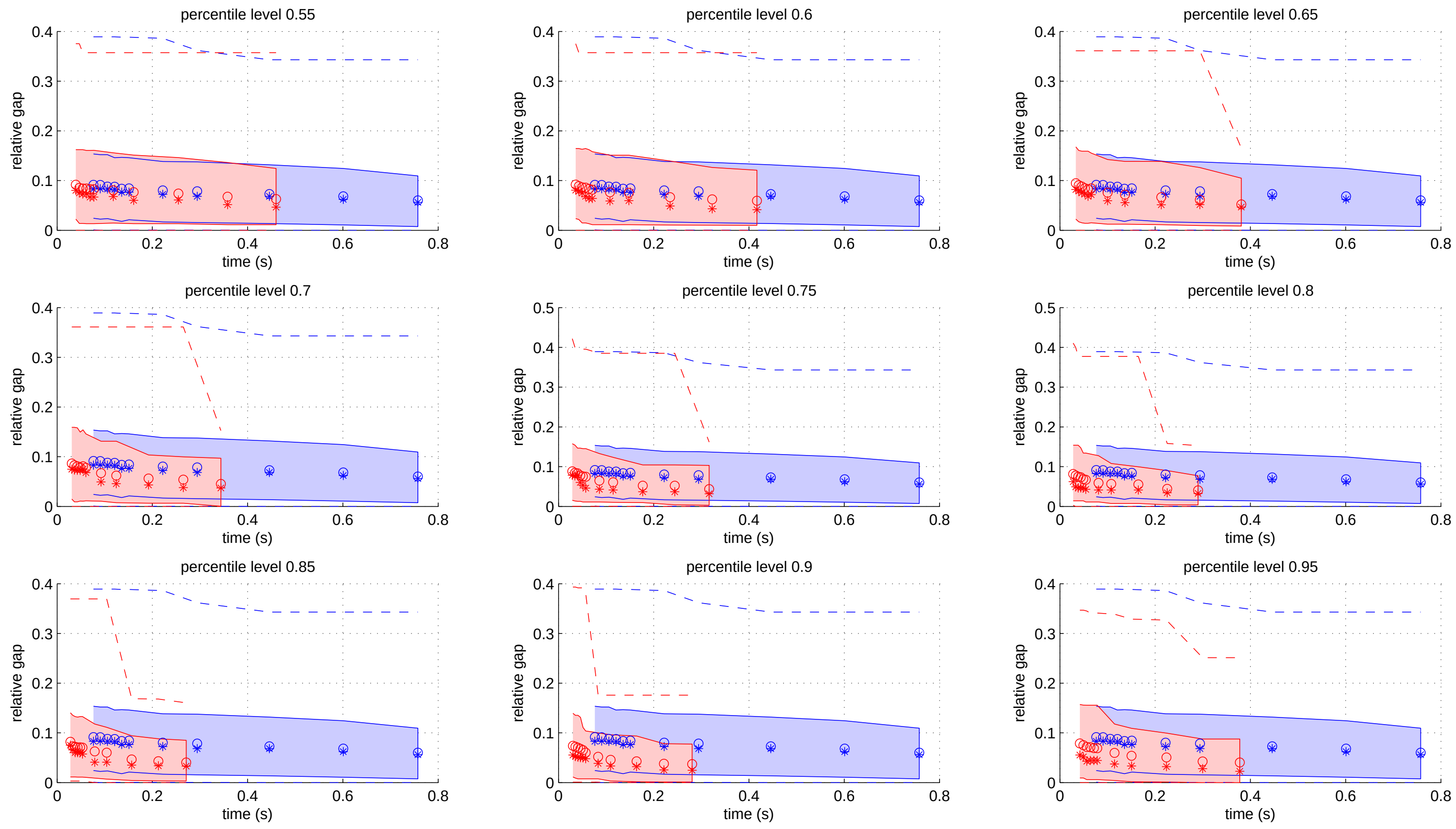


Figure C.3.9: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided DP using exact edge counters (red). Computational time for determining the exact counters is ignored. Instances with 25 customers.

C.3.3

Two-Phase DP Prototype under Realistic Conditions

15 Customers

Phase 1 Beam of 250

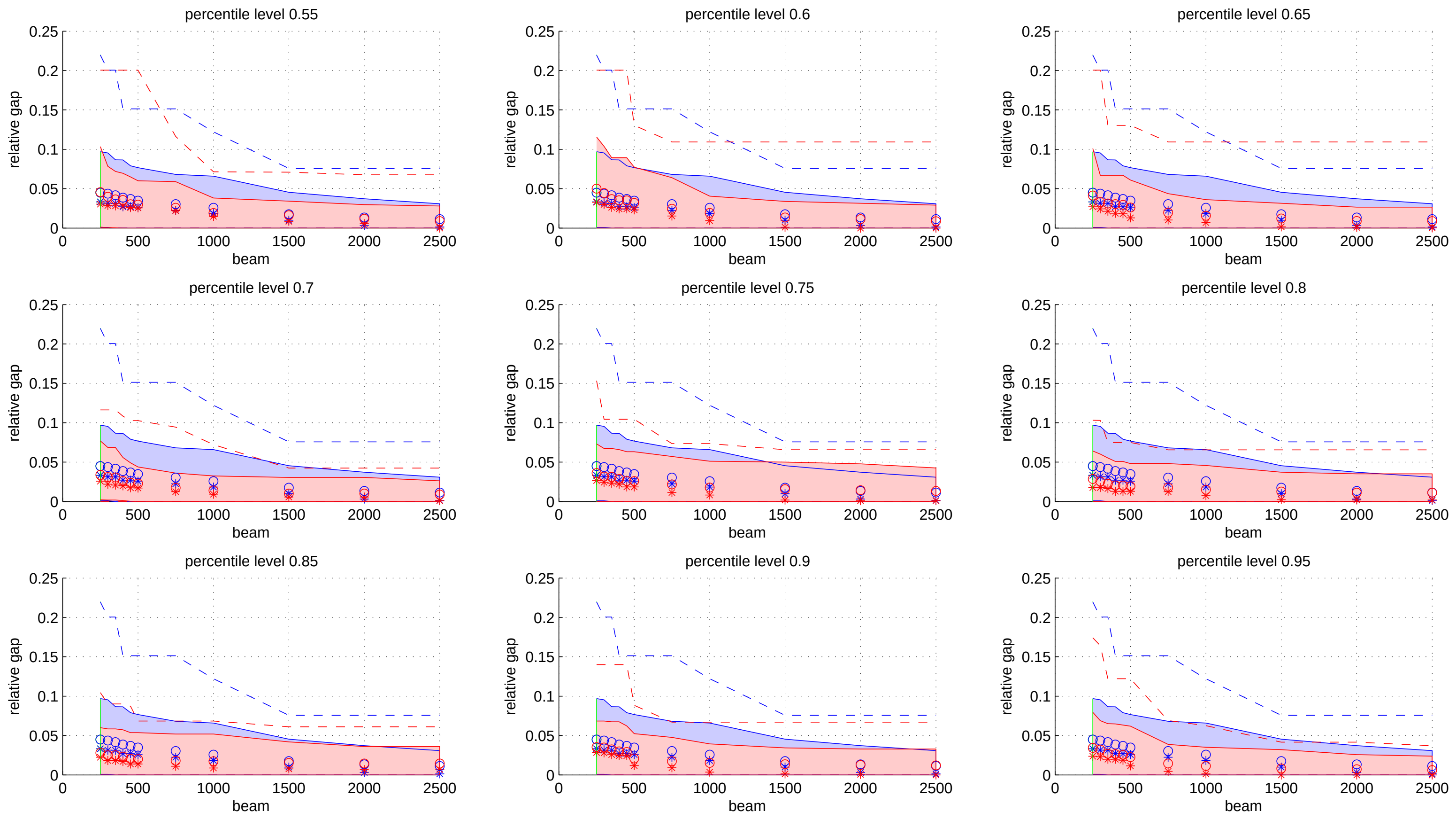


Figure C.3.10: Relative gap from optimality for different beam widths for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 250. Instances with 15 customers.

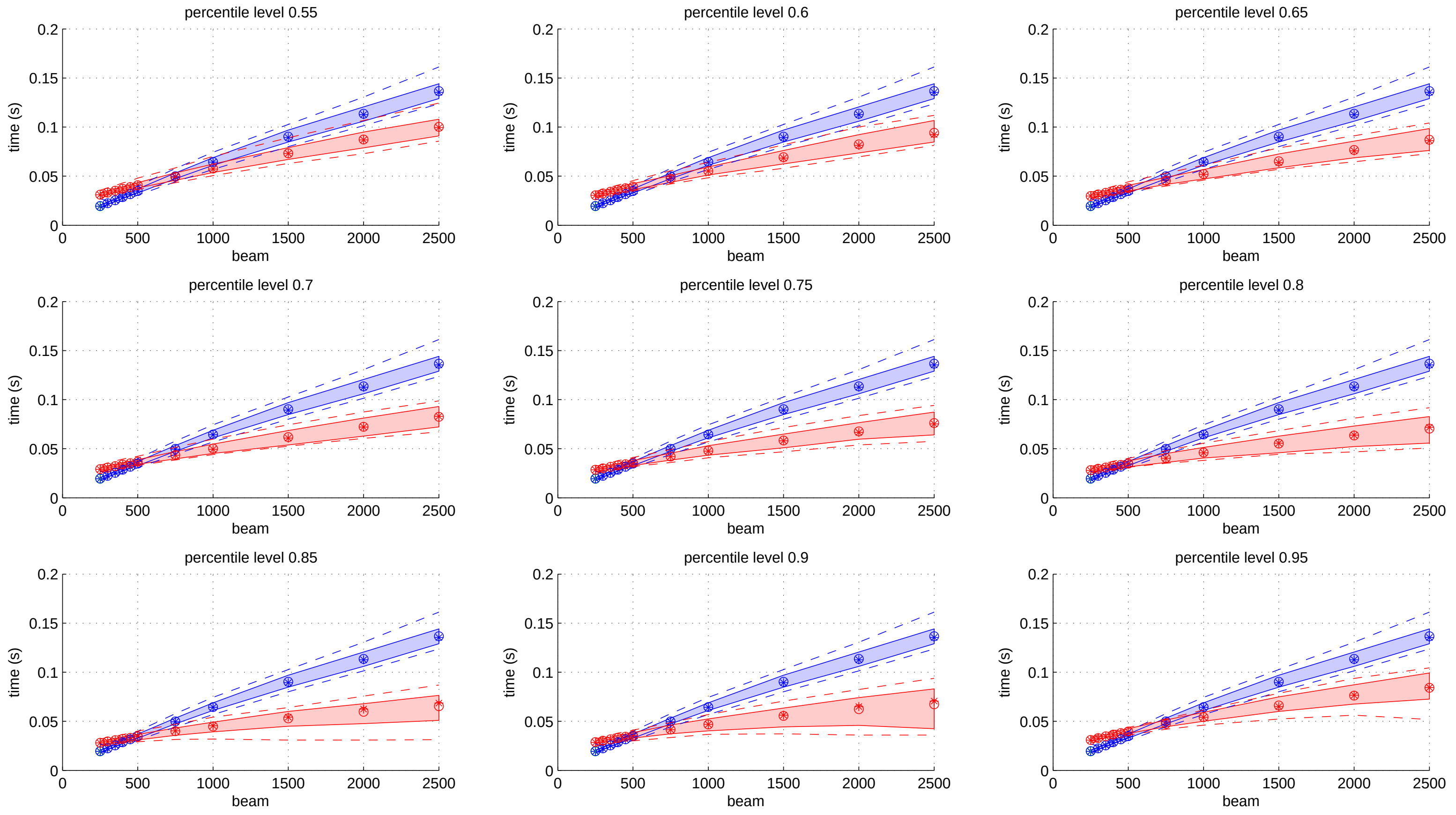


Figure C.3.11: Computational times for different beam widths for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 250. Instances with 15 customers.

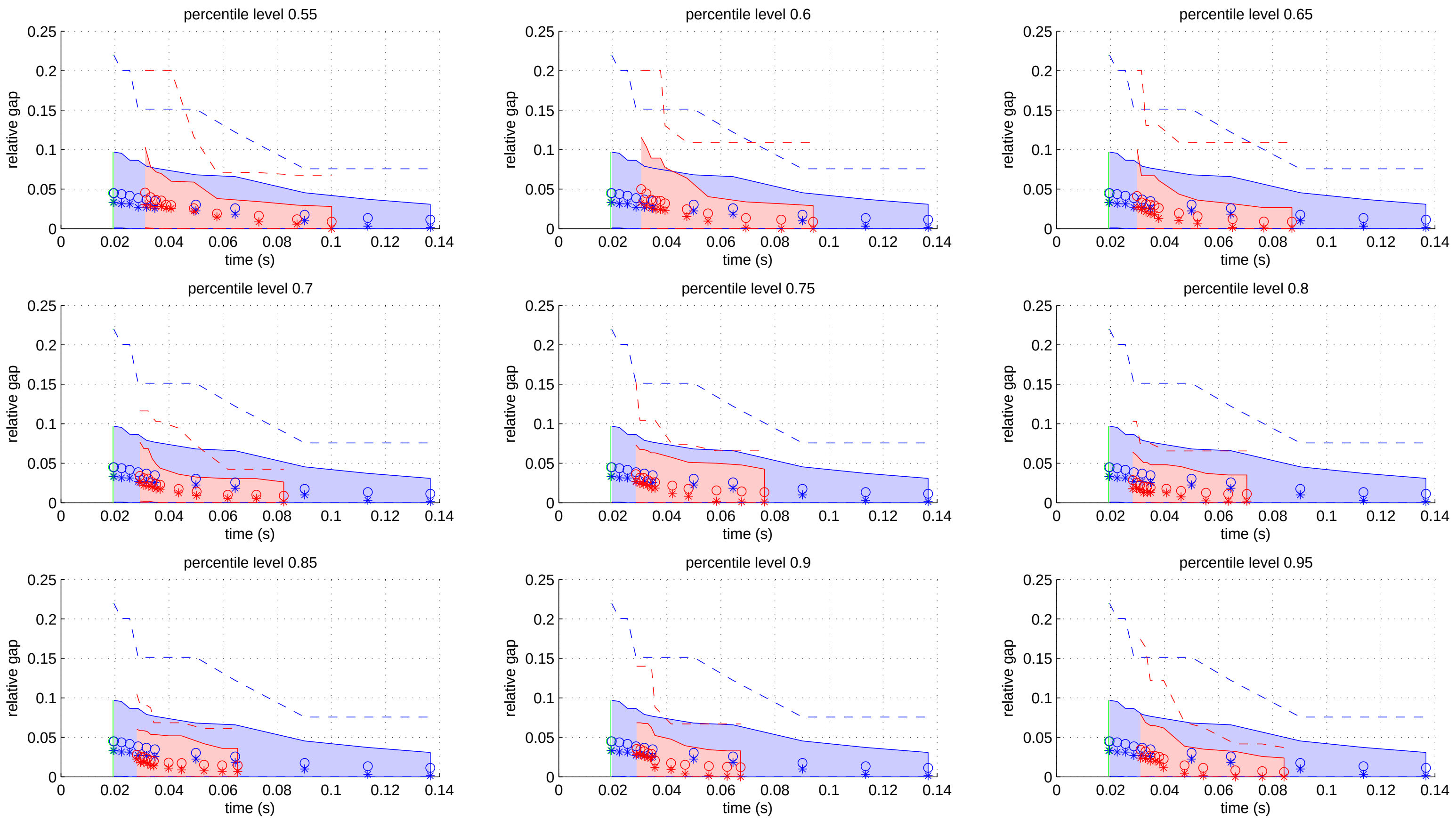


Figure C.3.12: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 250. Instances with 15 customers.

Two-Phase DP Prototype under Realistic Conditions

15 Customers

Phase 1 Beam of 500

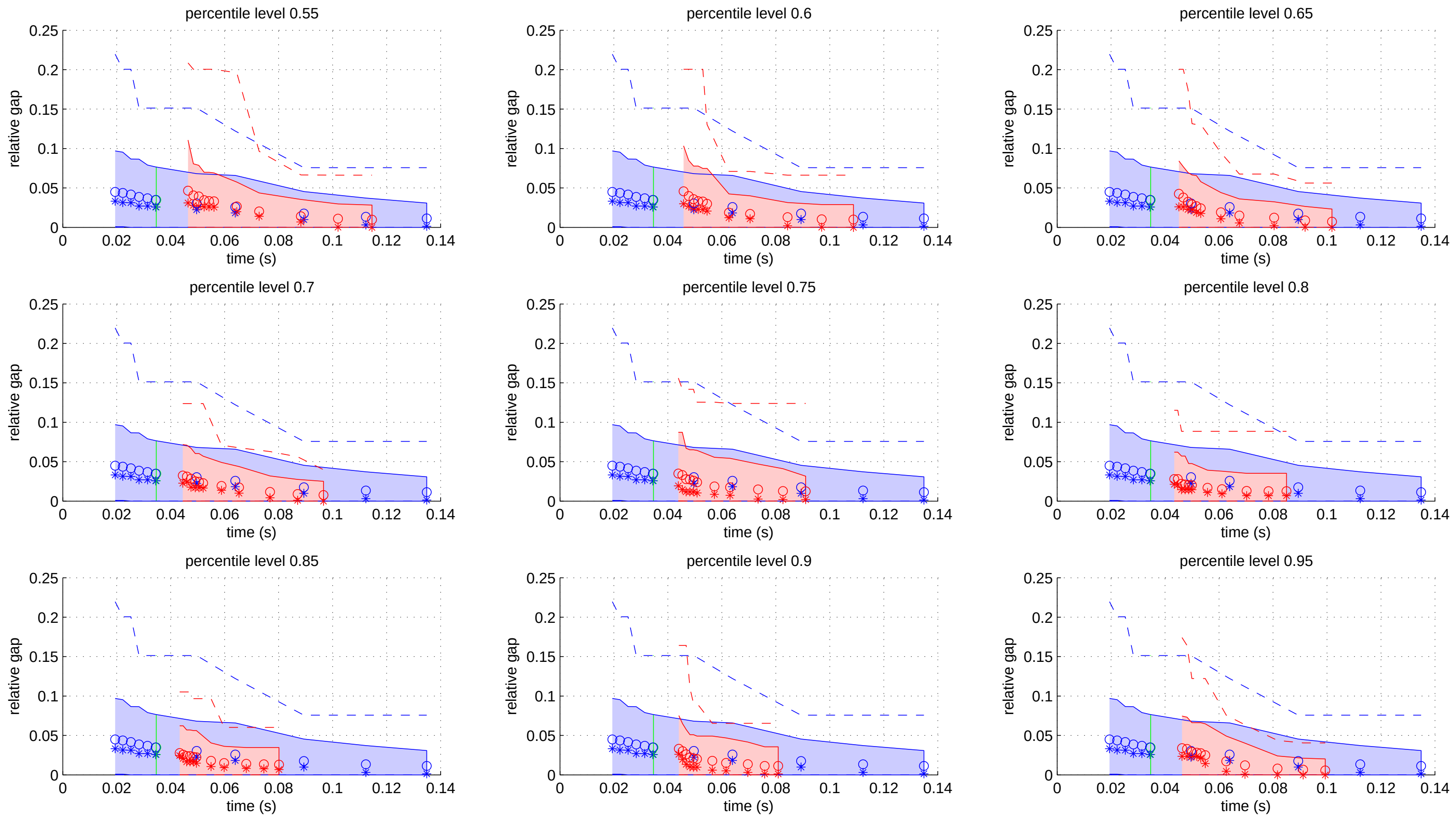


Figure C.3.13: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 500. Instances with 15 customers.

Two-Phase DP Prototype

under

Realistic Conditions

15 Customers

Phase 1 Beam of 1000

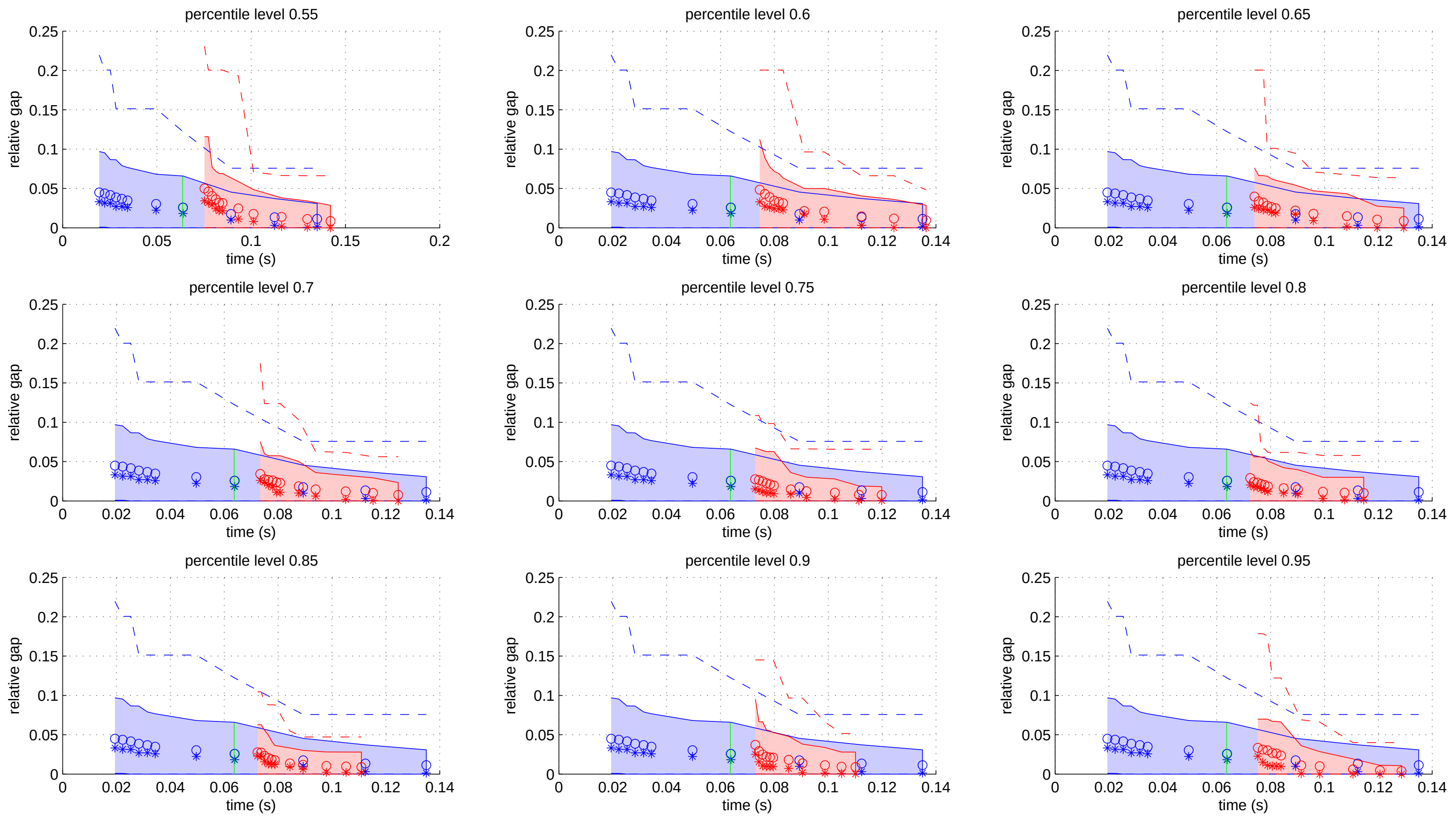


Figure C.3.14: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 1000. Instances with 15 customers.

C.3.4

Two-Phase DP Prototype

under

Realistic Conditions

25 Customers

Phase 1 Beam of 250

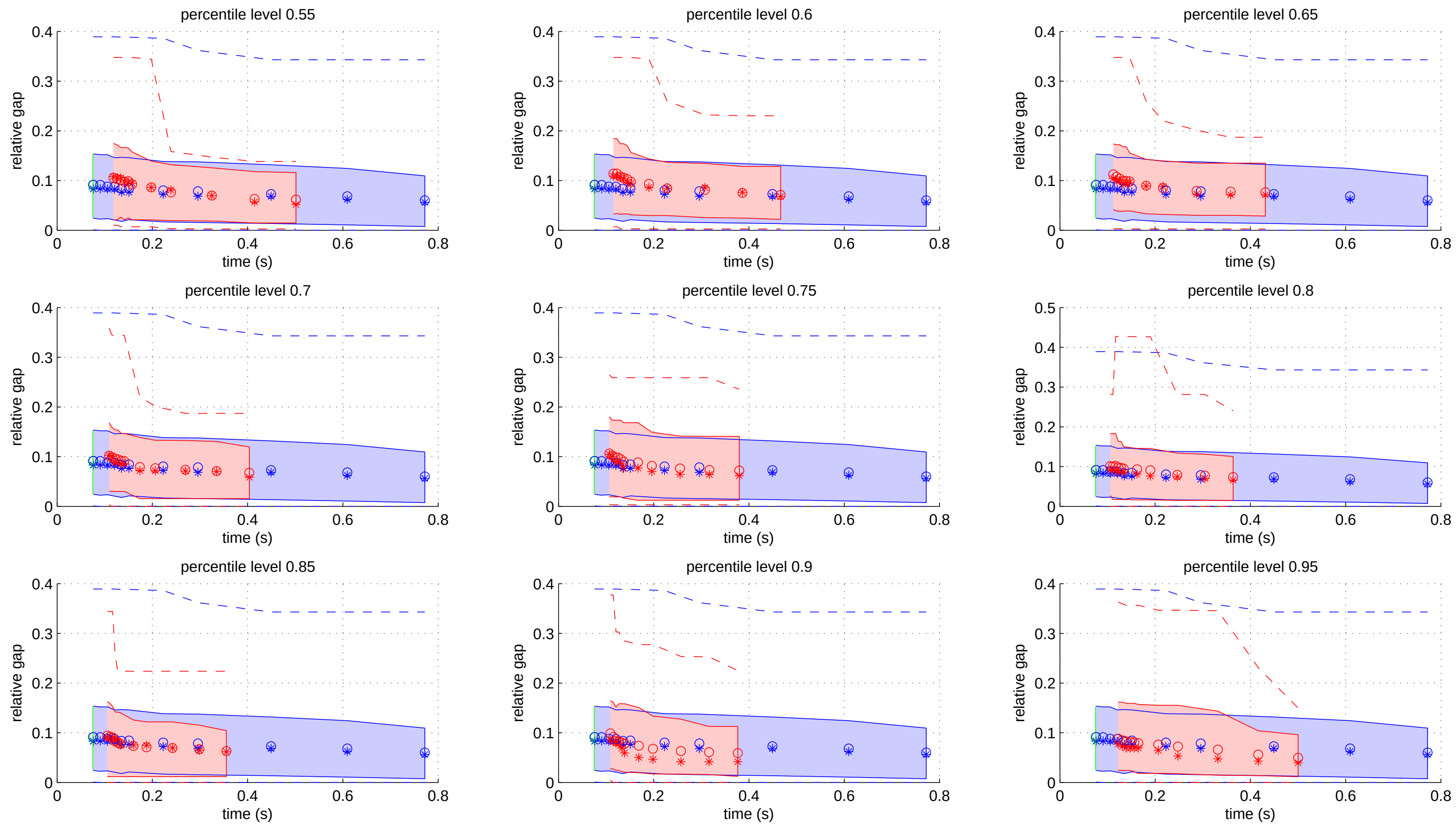


Figure C.3.15: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 250. Instances with 25 customers.

Two-Phase DP Prototype under Realistic Conditions

25 Customers

Phase 1 Beam of 500

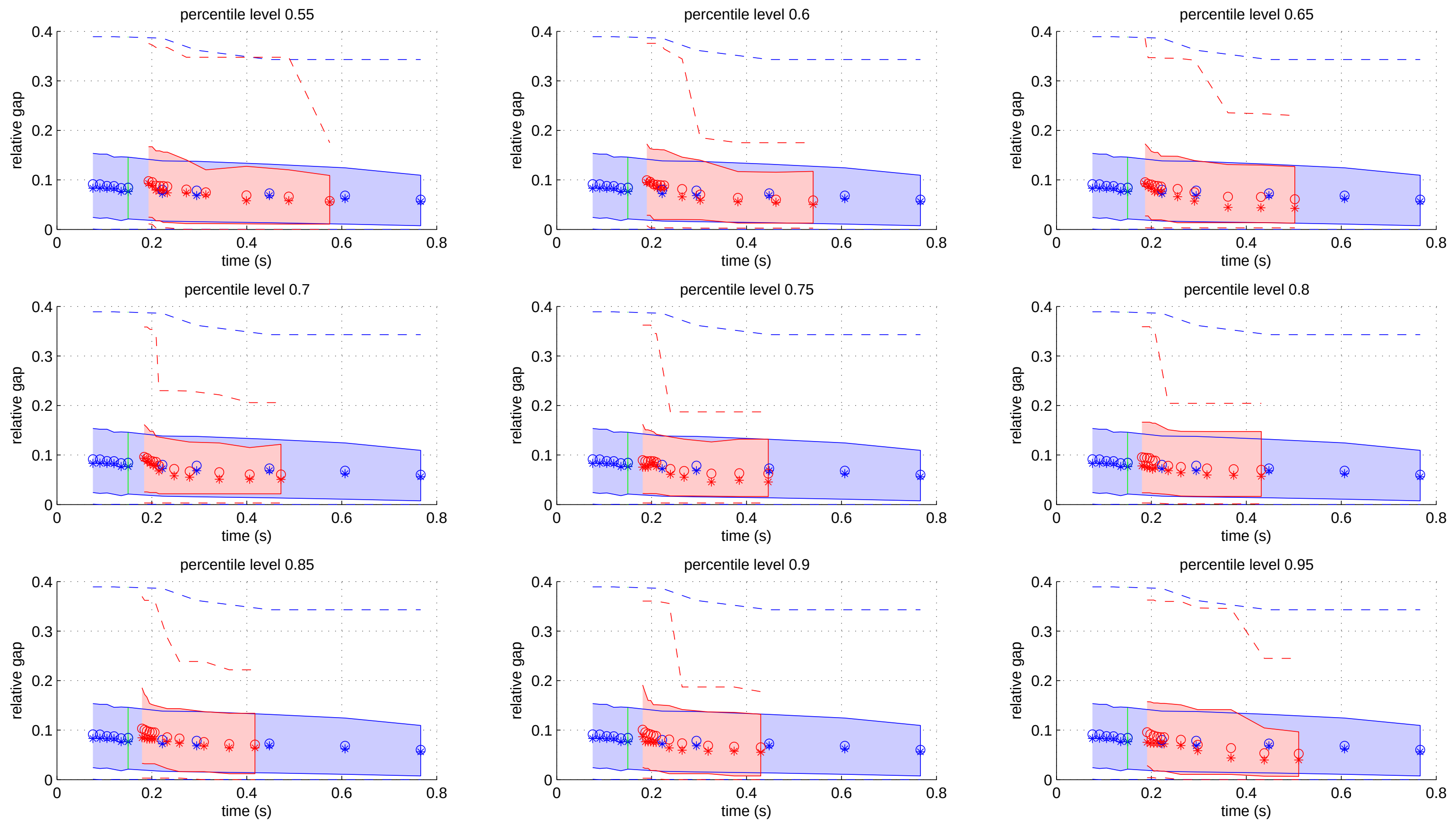


Figure C.3.16: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 500. Instances with 25 customers.

Two-Phase DP Prototype under Realistic Conditions

25 Customers

Phase 1 Beam of 1000

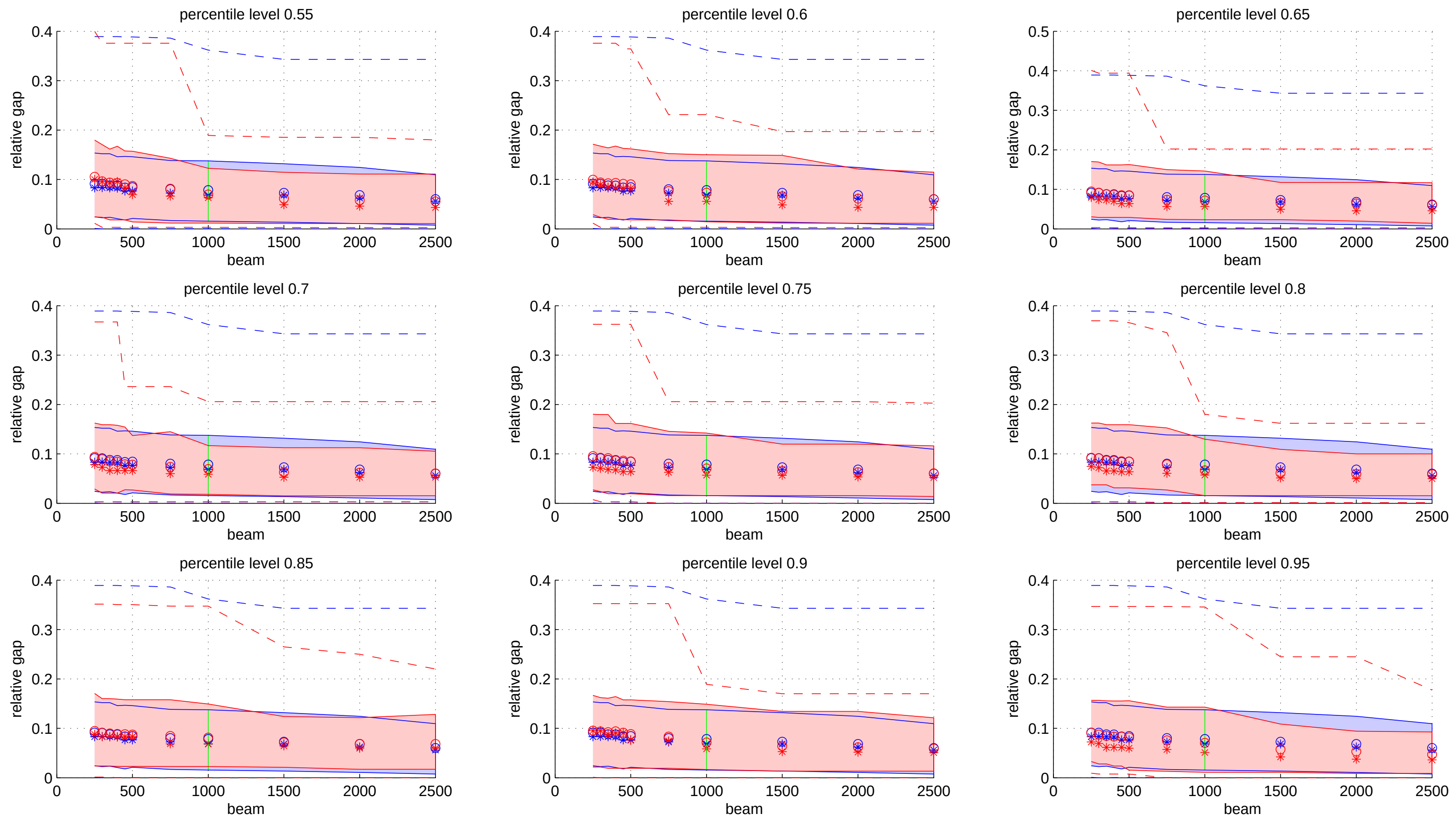


Figure C.3.17: Relative gap from optimality for different beam widths for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 1000. Instances with 25 customers.

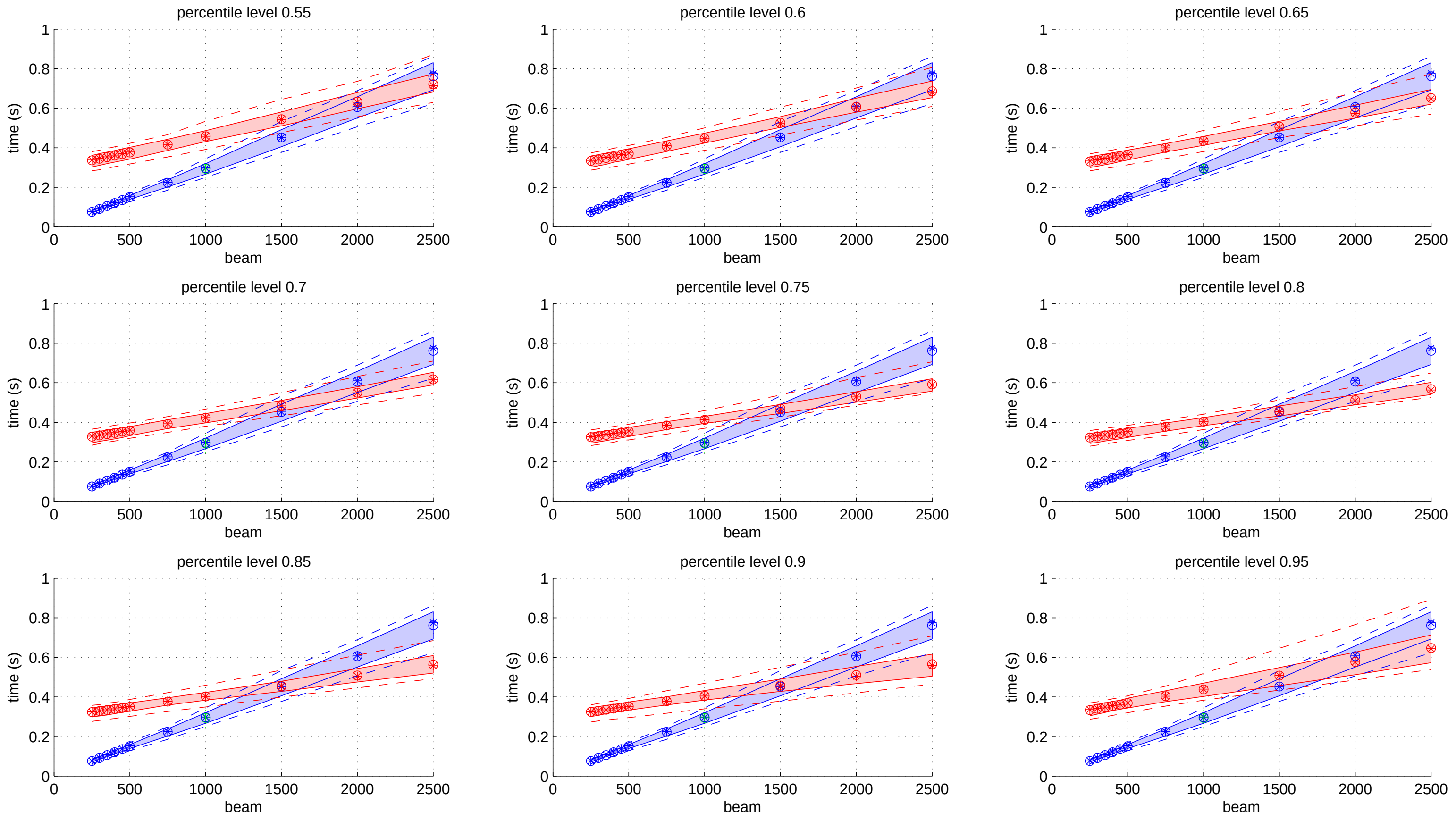


Figure C.3.18: Computational times for different beam widths for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 1000. Instances with 25 customers.

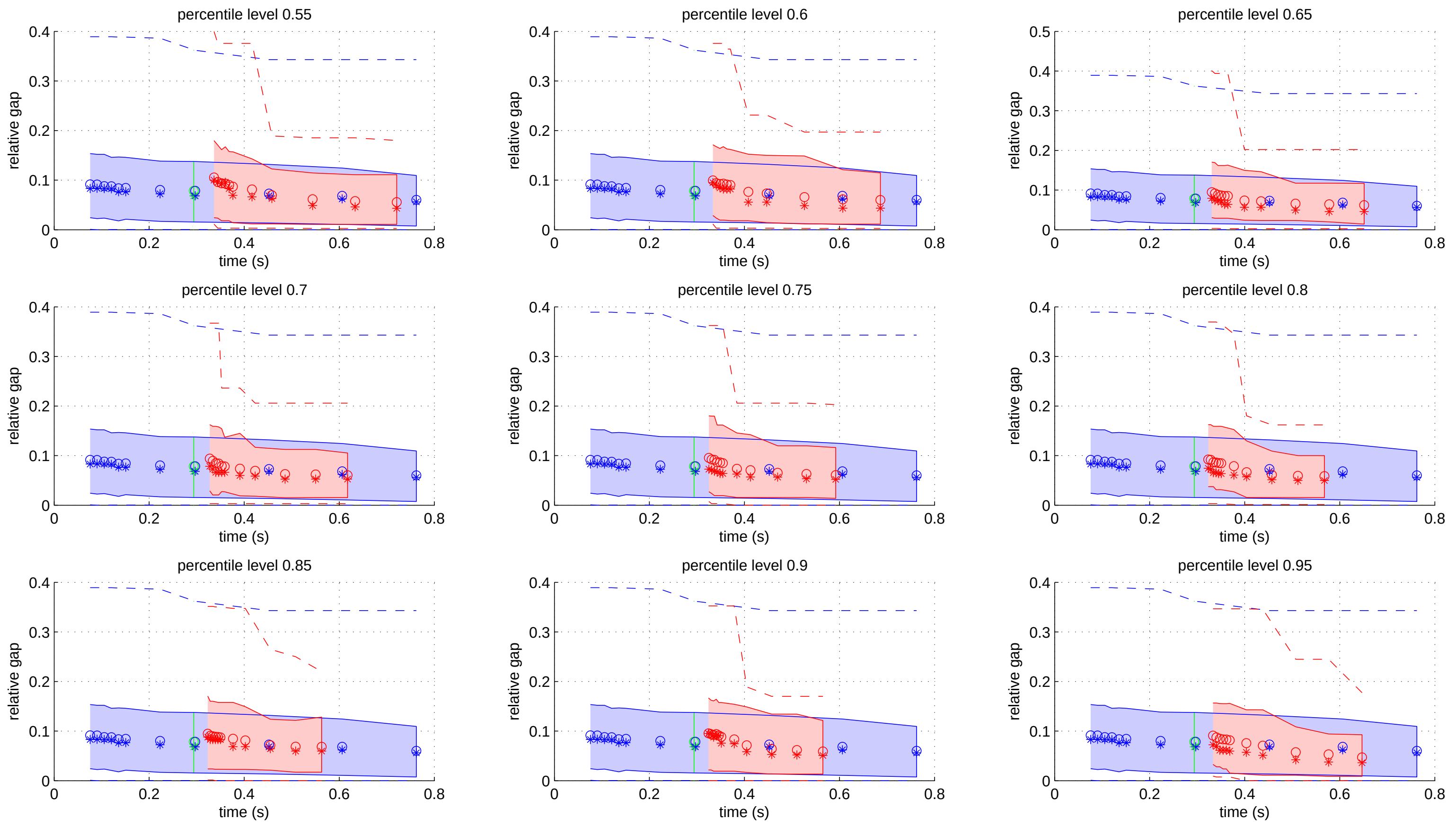


Figure C.3.19: Relative gap from optimality for different average computational times for Restricted DP (blue) and Two-Phase Guided Restricted DP (red). Phase 1 has beam width 1000. Instances with 25 customers.

C.3.5

Random Solutions Prototype

15 Customers

10^3 **Solutions**

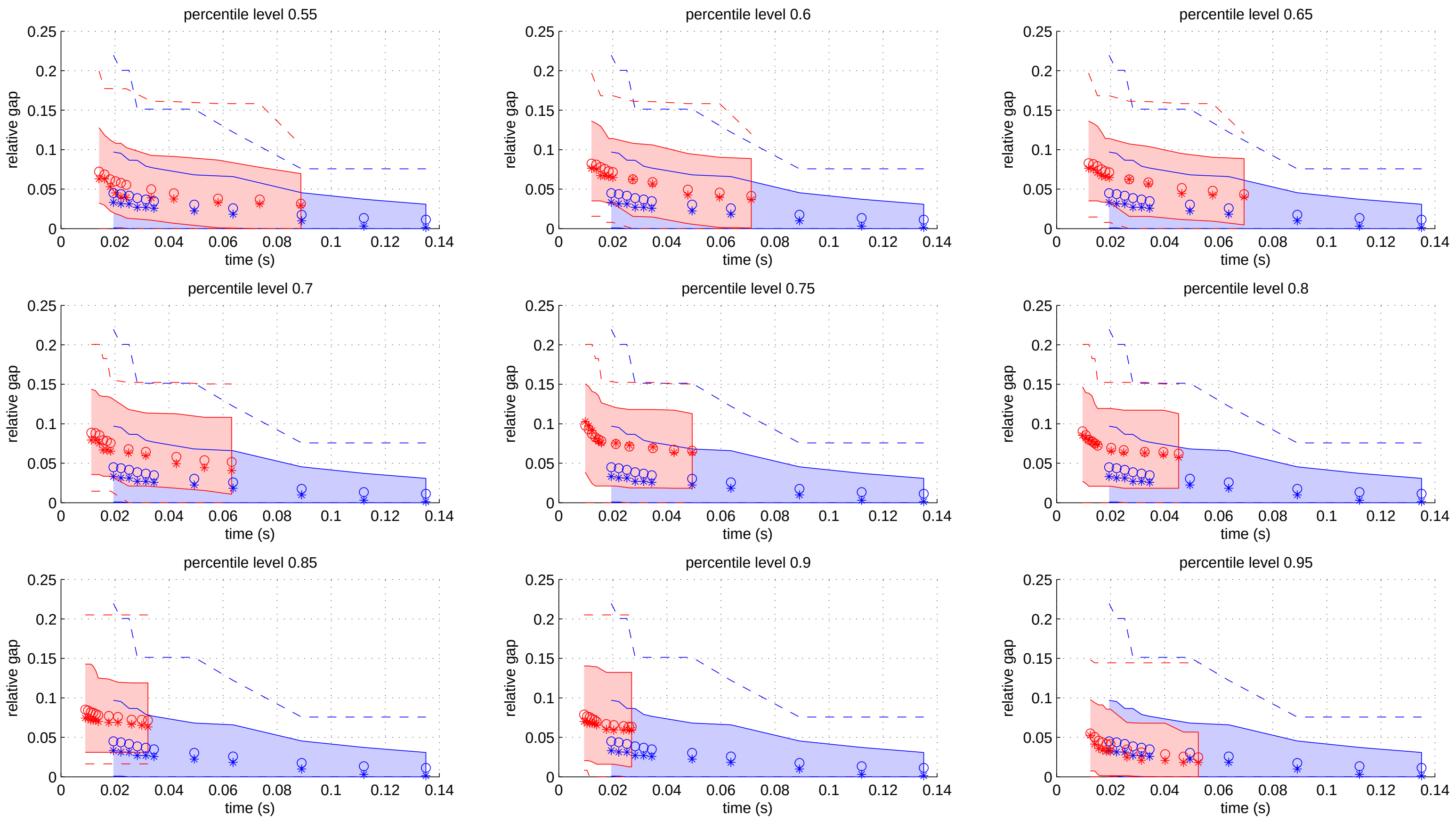


Figure C.3.20: Relative gap from optimality for different average computational times for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^3 random solutions are generated. Instances with 15 customers.

Random Solutions Prototype

15 Customers

10^4 **Solutions**

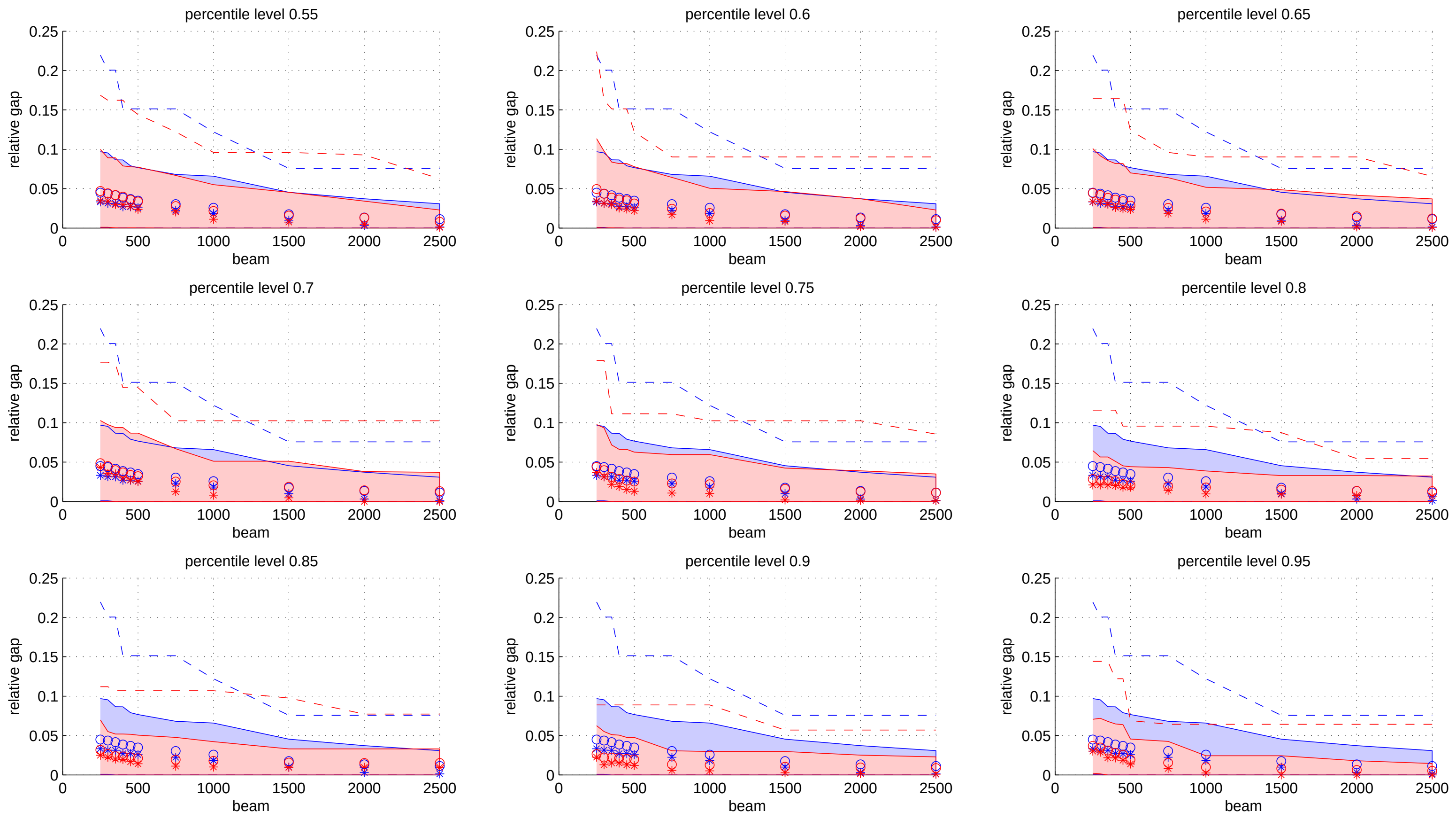


Figure C.3.21: Relative gap from optimality for different beam widths for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^4 random solutions are generated. Instances with 15 customers.

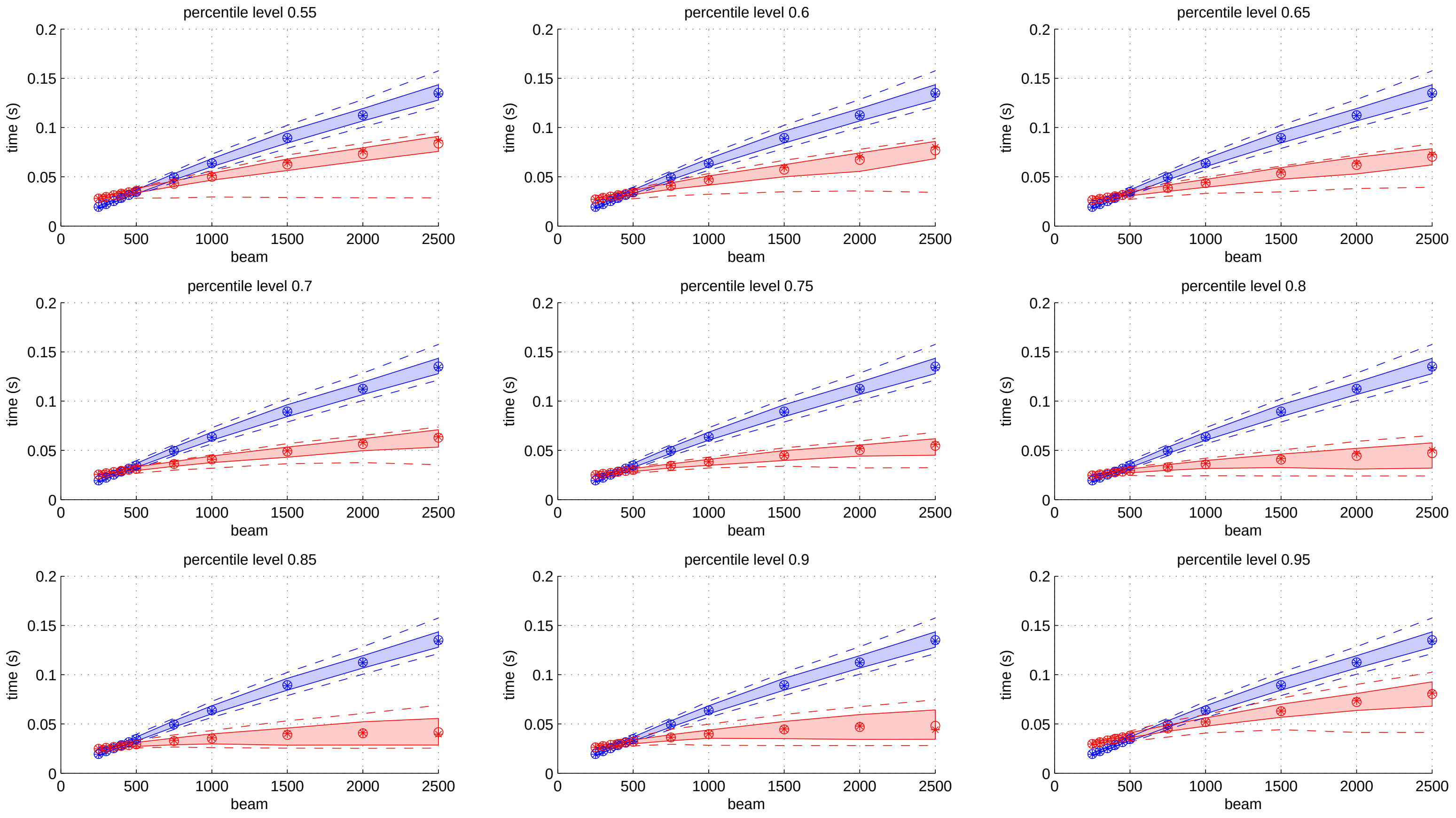


Figure C.3.22: Computational times for different beam widths for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^4 random solutions are generated. Instances with 15 customers.

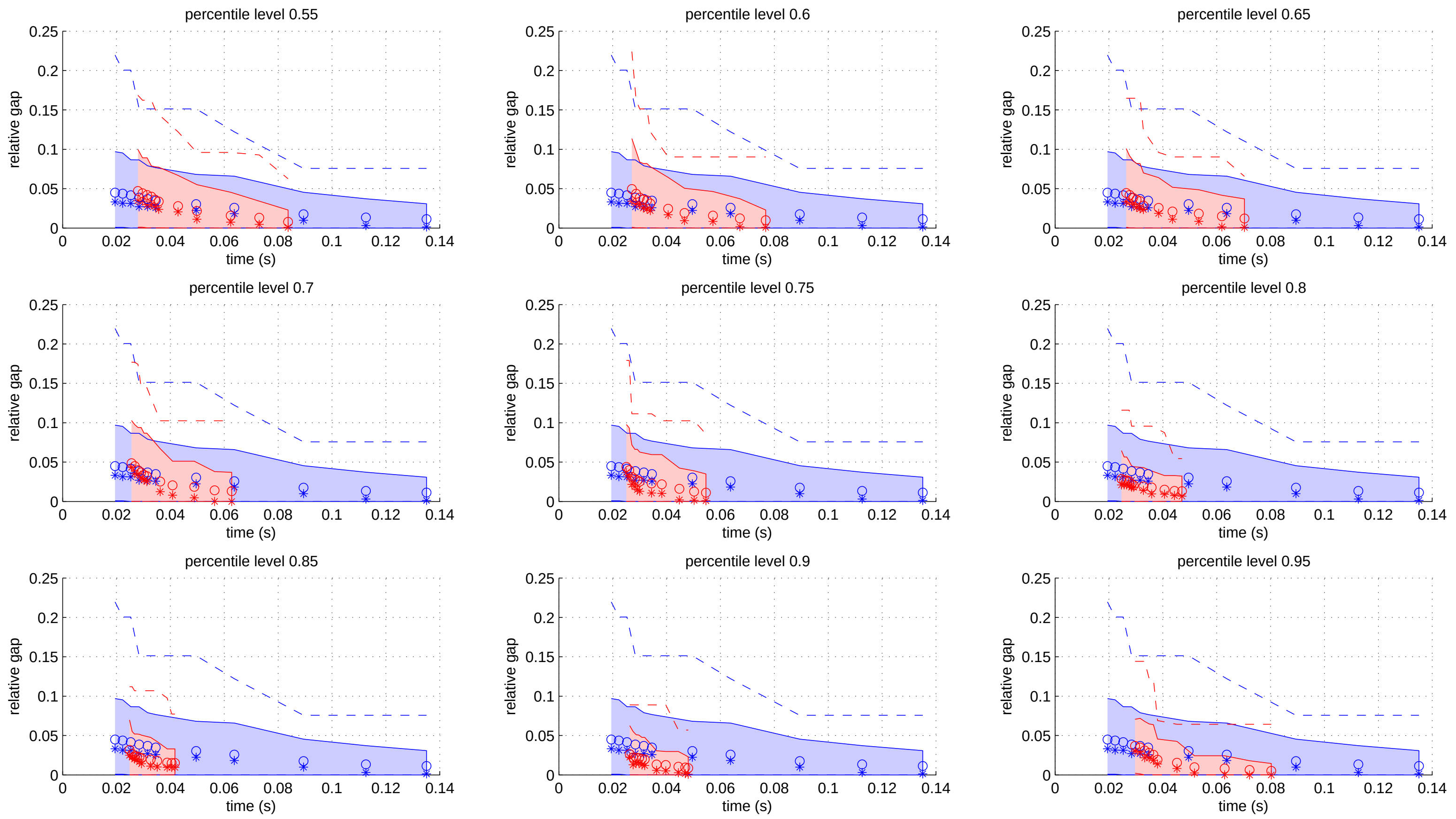


Figure C.3.23: Relative gap from optimality for different average computational times for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^4 random solutions are generated. Instances with 15 customers.

Random Solutions Prototype

25 Customers

10^5 Solutions

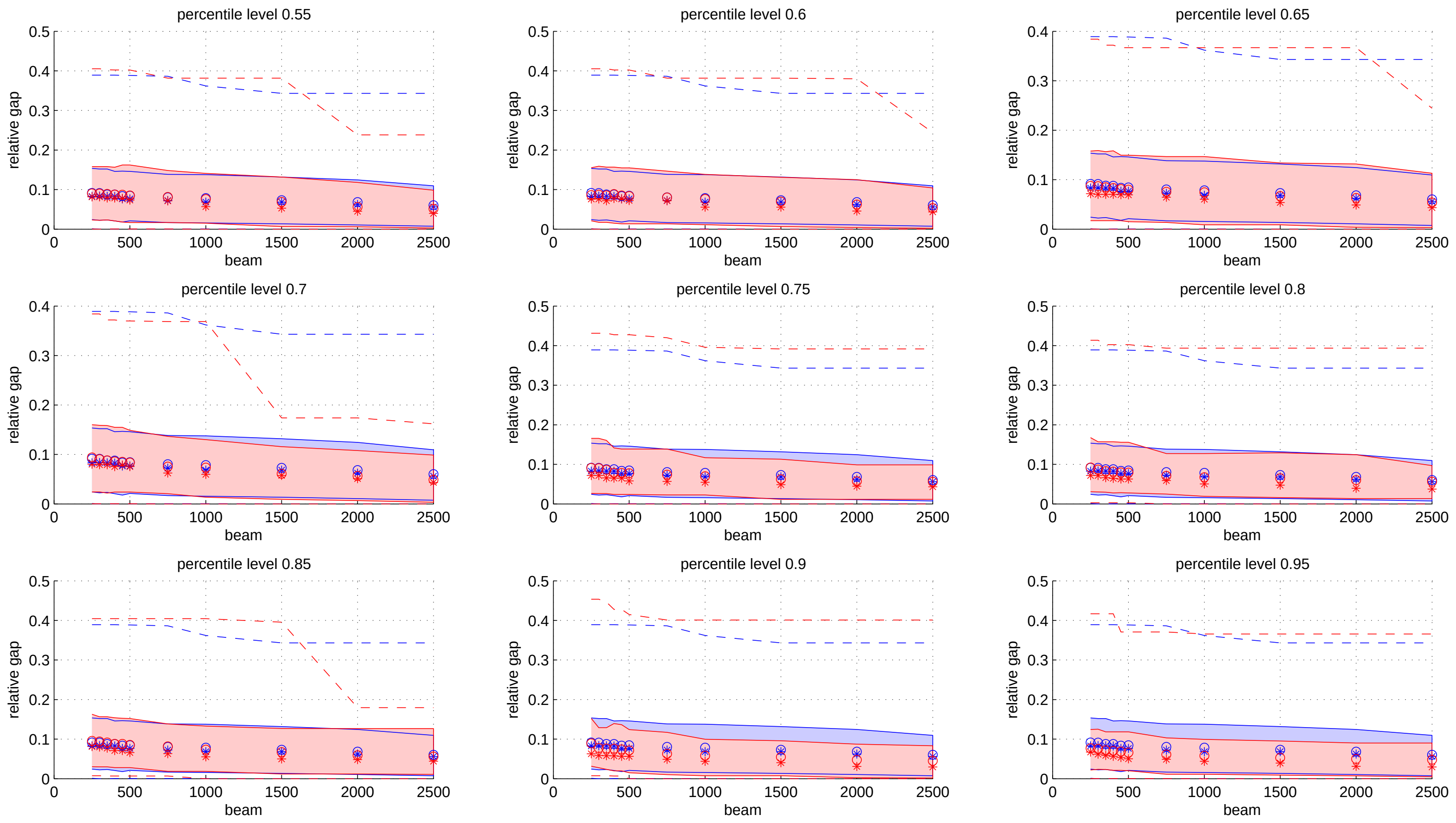


Figure C.3.24: Relative gap from optimality for different beam widths for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^5 random solutions are generated. Instances with 25 customers.

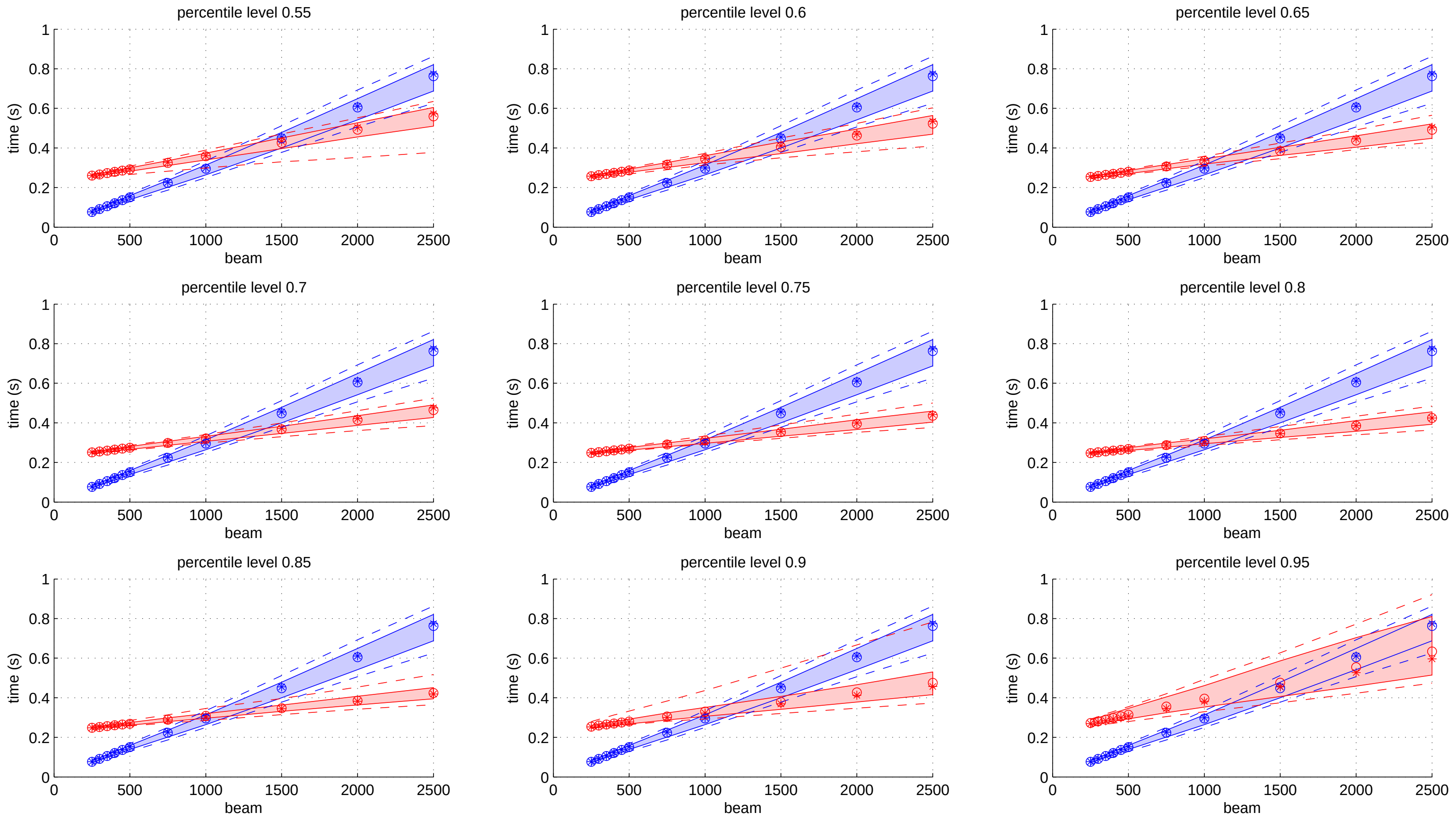


Figure C.3.25: Computational times for different beam widths for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^5 random solutions are generated. Instances with 25 customers.

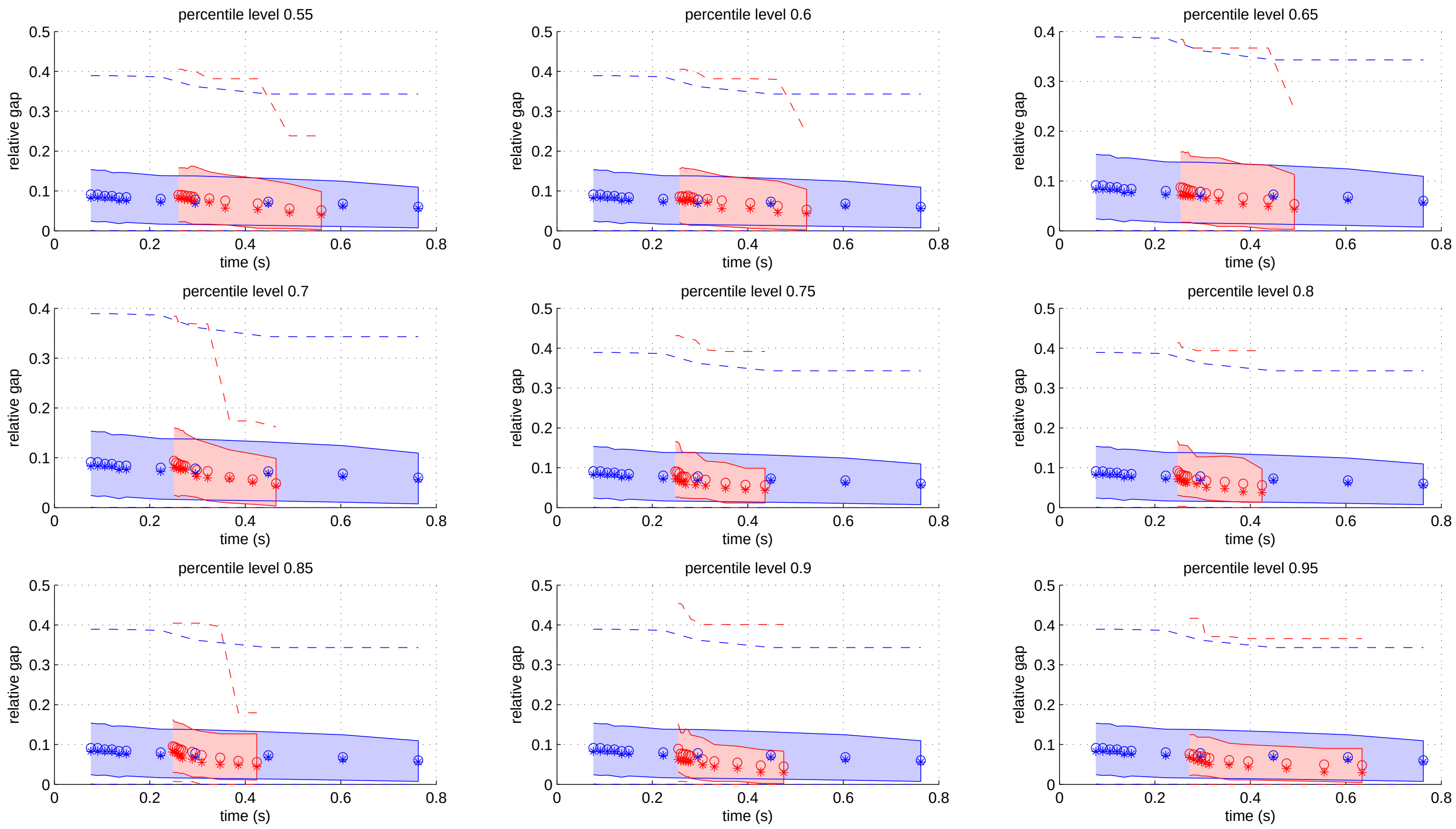


Figure C.3.26: Relative gap from optimality for different average computational times for Restricted DP (blue) and Random Solutions Guided Restricted DP (red). 10^5 random solutions are generated. Instances with 25 customers.

C.3.6

Out-link Prototype

15 Customers

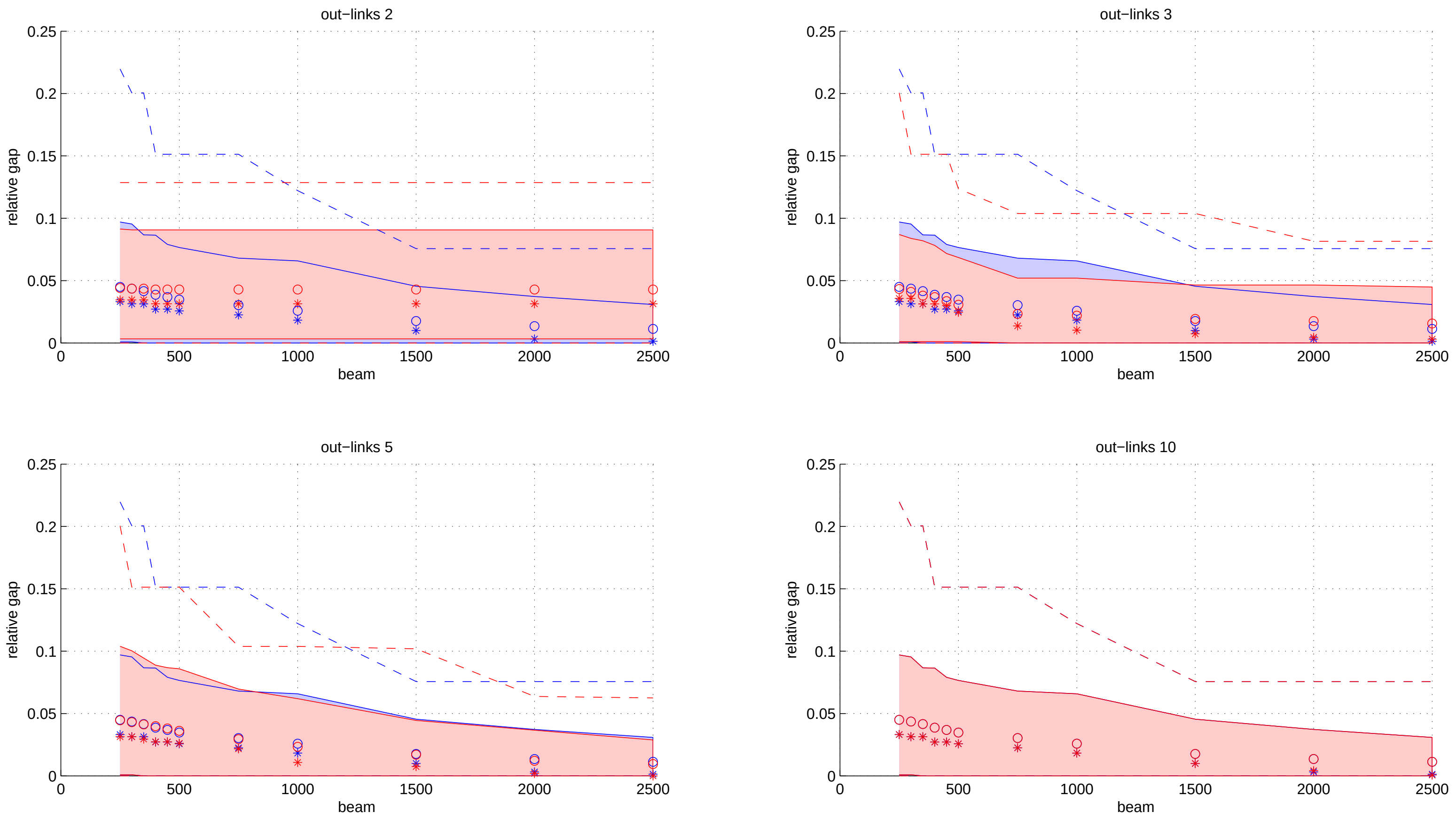


Figure C.3.27: Relative gap from optimality for different beam widths for Restricted DP (blue) and Out-link Guided Restricted DP (red). Instances with 15 customers.

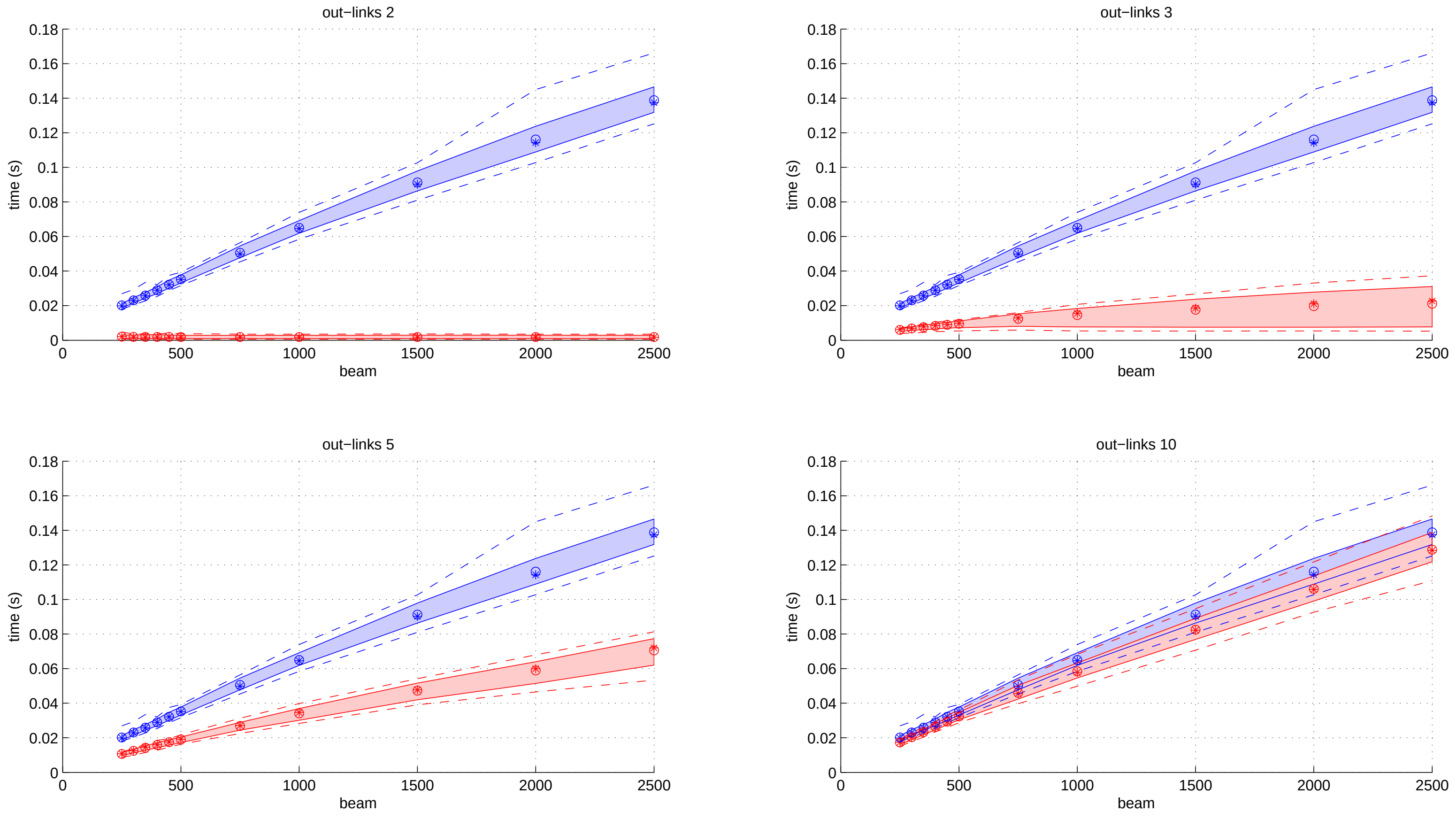


Figure C.3.28: Computational times for different beam widths for Restricted DP (blue) and Out-link Guided Restricted DP (red). Instances with 15 customers.

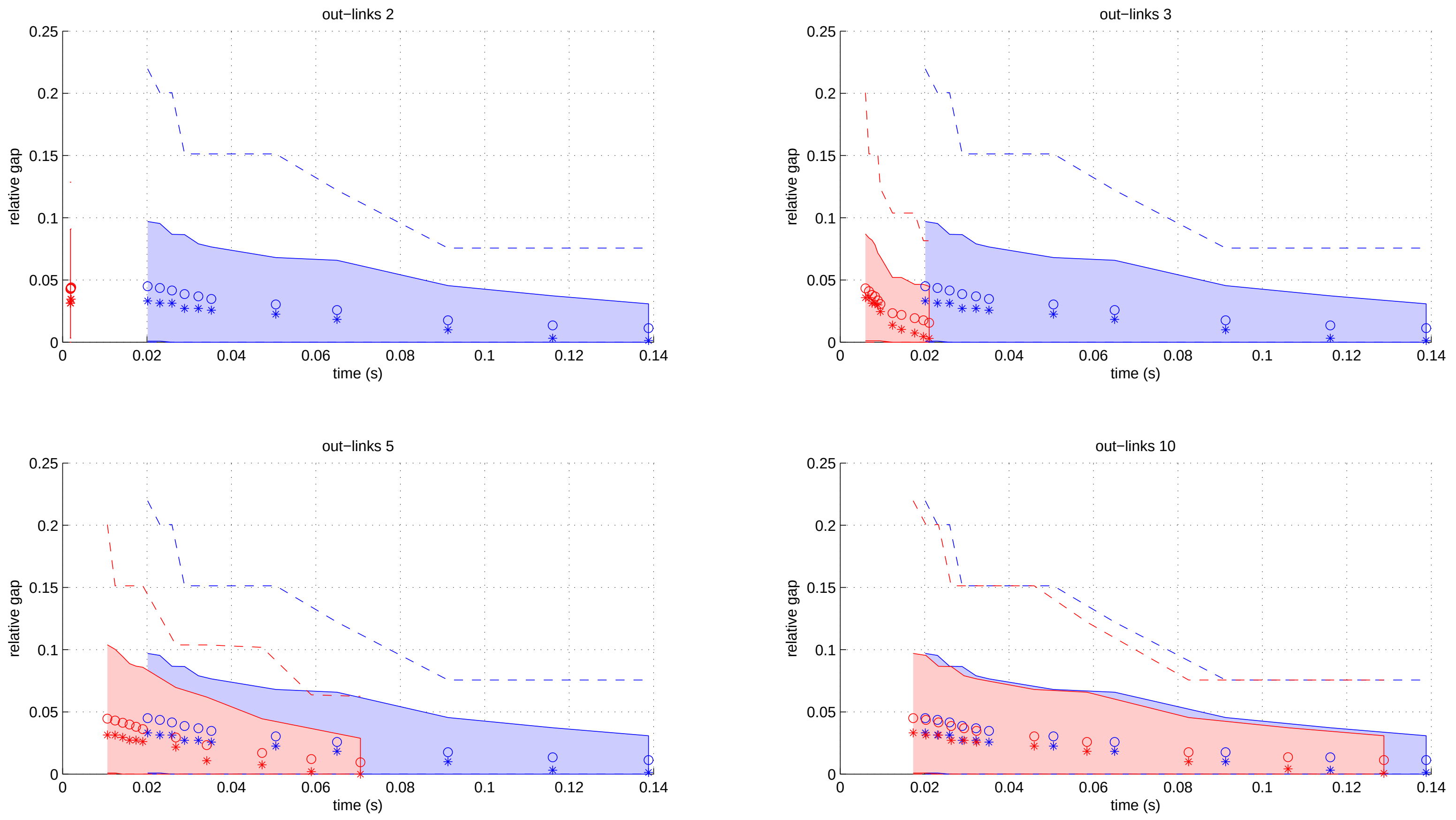


Figure C.3.29: Relative gap from optimality for different average computational times for Restricted DP (blue) and Out-link Guided Restricted DP (red). Instances with 15 customers.

C.3.7

Out-link Prototype

25 Customers

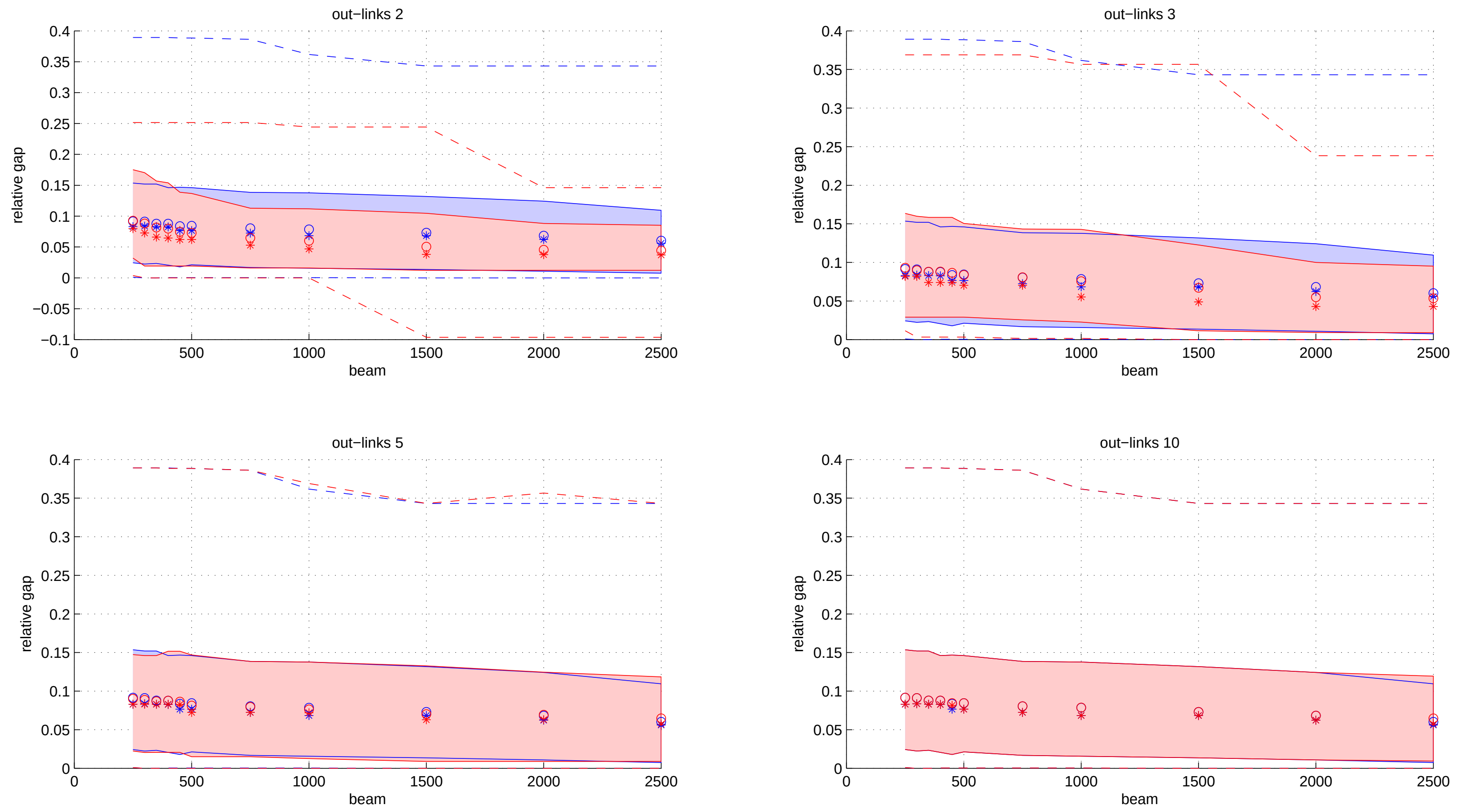


Figure C.3.30: Relative gap from optimality for different beam widths for Restricted DP (blue) and Out-link Guided Restricted DP (red). Instances with 25 customers.

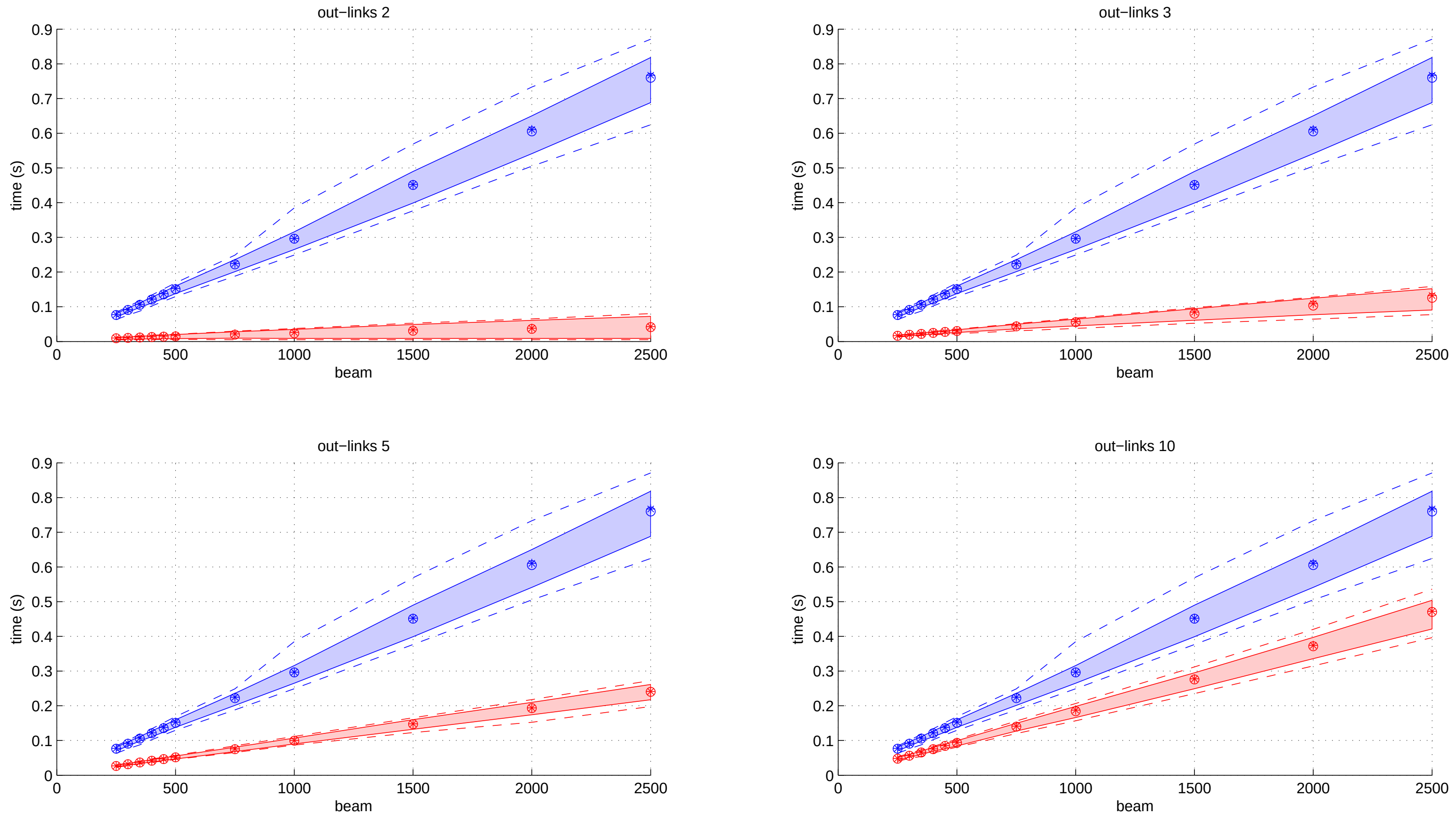


Figure C.3.31: Computational times for different beam widths for Restricted DP (blue) and Out-link Guided Restricted DP (red). Instances with 25 customers.

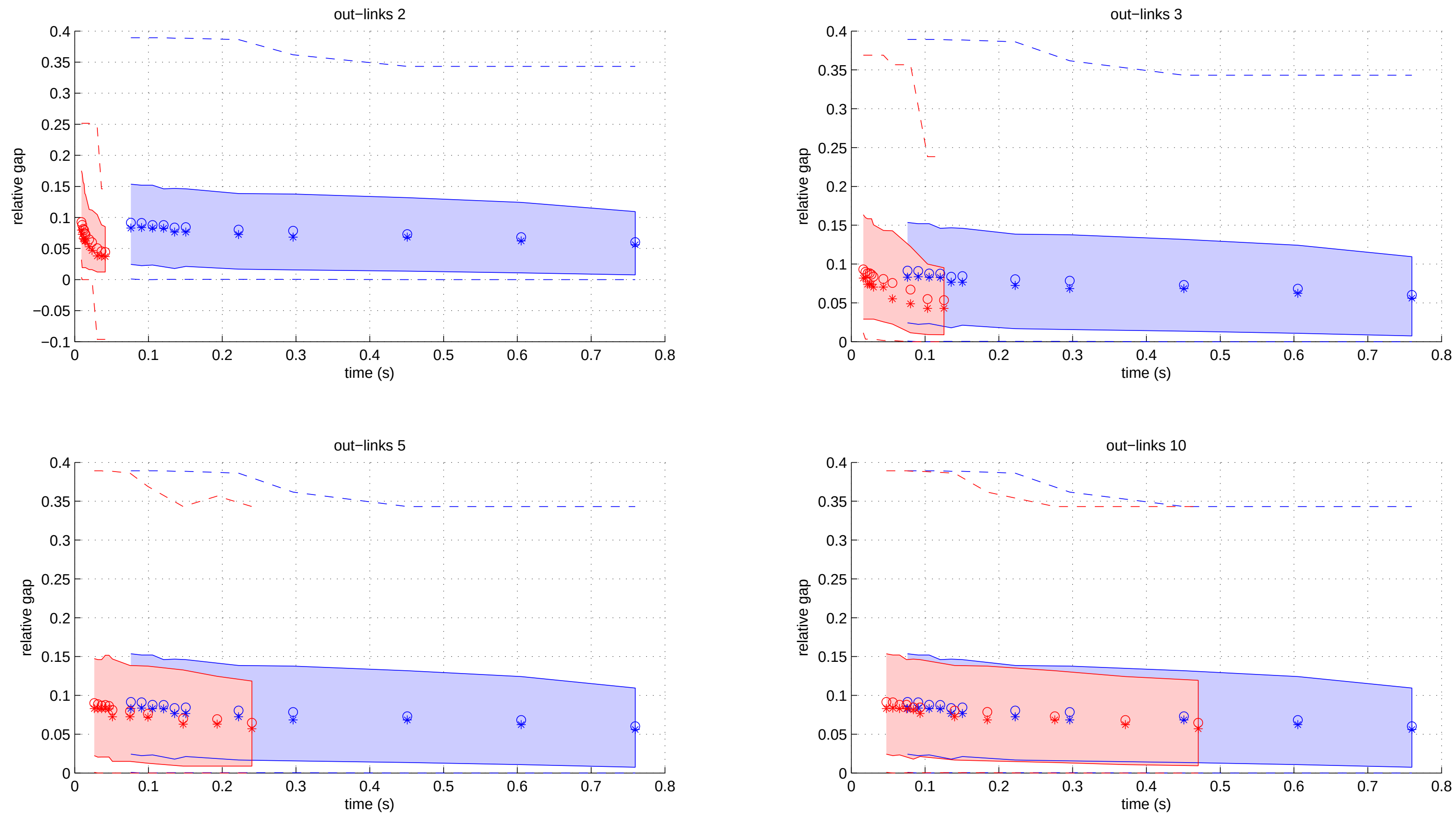


Figure C.3.32: Relative gap from optimality for different average computational times for Restricted DP (blue) and Out-link Guided Restricted DP (red). Instances with 25 customers.

C.4

Literature Benchmarks

Name	**Optimal Solution**	**Reference Solution**		**Two-Phase DP Prototype**		**Random Solutions Prototype**		**Out-link Prototype**	
	value	*value*	*gap*	*value*	*gap*	*value*	*gap*	*value*	*gap*
BERLIN52	7542	**8317**	**0.1028**	8540	0.1323	8594	0.1395	**8317**	**0.1028**
EIL51	426	474	0.1127	**444**	**0.0423**	486	0.1408	*469*	*0.1009*
EIL76	538	**599**	**0.1134**	630	0.1710	637	0.1840	601	0.1171
KROA100	21282	**24460**	**0.1493**	27510	0.2926	31416	0.4762	24649	0.1582
KROB100	22141	28836	0.3024	**26542**	**0.1988**	32931	0.4873	*28210*	*0.2741*
KROC100	20749	**26250**	**0.2651**	26975	0.3001	30374	0.4639	**26250**	**0.2651**
KROD100	21294	**25757**	**0.2096**	27290	0.2816	28142	0.3216	25948	0.2186
KROE100	22068	24833	0.1253	27689	0.2547	27803	0.2599	**24390**	**0.1052**
PR76	108159	135400	0.2519	*134528*	*0.2438*	146735	0.3567	**134028**	**0.2392**
RAT99	1211	**1434**	**0.1841**	1537	0.2692	1550	0.2799	**1434**	**0.1841**
RD100	7910	**8949**	**0.1314**	10140	0.2819	10508	0.3284	8970	0.1340
ST70	675	827	0.2252	**811**	**0.2015**	*815*	*0.2074*	838	0.2415
			0.1811		**0.2225**		**0.3038**		**0.1784**

Table C.4.1: The selected Euclidean TSP benchmark instances with optimal objective values and the results for the reference and the prototypes. Note that the gap shown is the relative gap from optimality. The number in the name of the instance corresponds to the number of nodes (depot and customers).

Name	**Reference Solution**	**Two-Phase DP Prototype**	**Random Solutions Prototype**	**Out-link Prototype**
	time (s)	*time (s)*	*time (s)*	*time (s)*
BERLIN52	6.6089	*4.9380*	*4.7180*	**0.6453**
EIL51	6.4348	*4.8876*	*2.1696*	**0.6653**
EIL76	18.3848	*13.0682*	*5.9394*	**1.3580**
KROA100	37.9137	*35.7781*	*7.6564*	**2.3773**
KROB100	38.4673	*32.7165*	*9.6402*	**2.3368**
KROC100	38.2139	44.3527	*7.0266*	**2.2783**
KROD100	35.6439	36.8104	*8.6592*	**2.1417**
KROE100	38.0335	*37.1698*	*8.3661*	**2.2033**
PR76	18.0644	*13.6237*	*6.0561*	**1.2806**
RAT99	35.7905	*30.3133*	*8.1247*	**2.1532**
RD100	36.9900	*34.3088*	*7.5929*	**2.1624**
ST70	15.3024	*15.1222*	*4.2174*	**1.1077**
	27.1540	**25.2574**	**6.6805**	**1.7258**

Table C.4.2: The computational time for the reference and the prototypes are listed for the selected Euclidean TSP benchmark instances. The number in the name of the instance corresponds to the number of nodes (depot and customers).

Appendix D

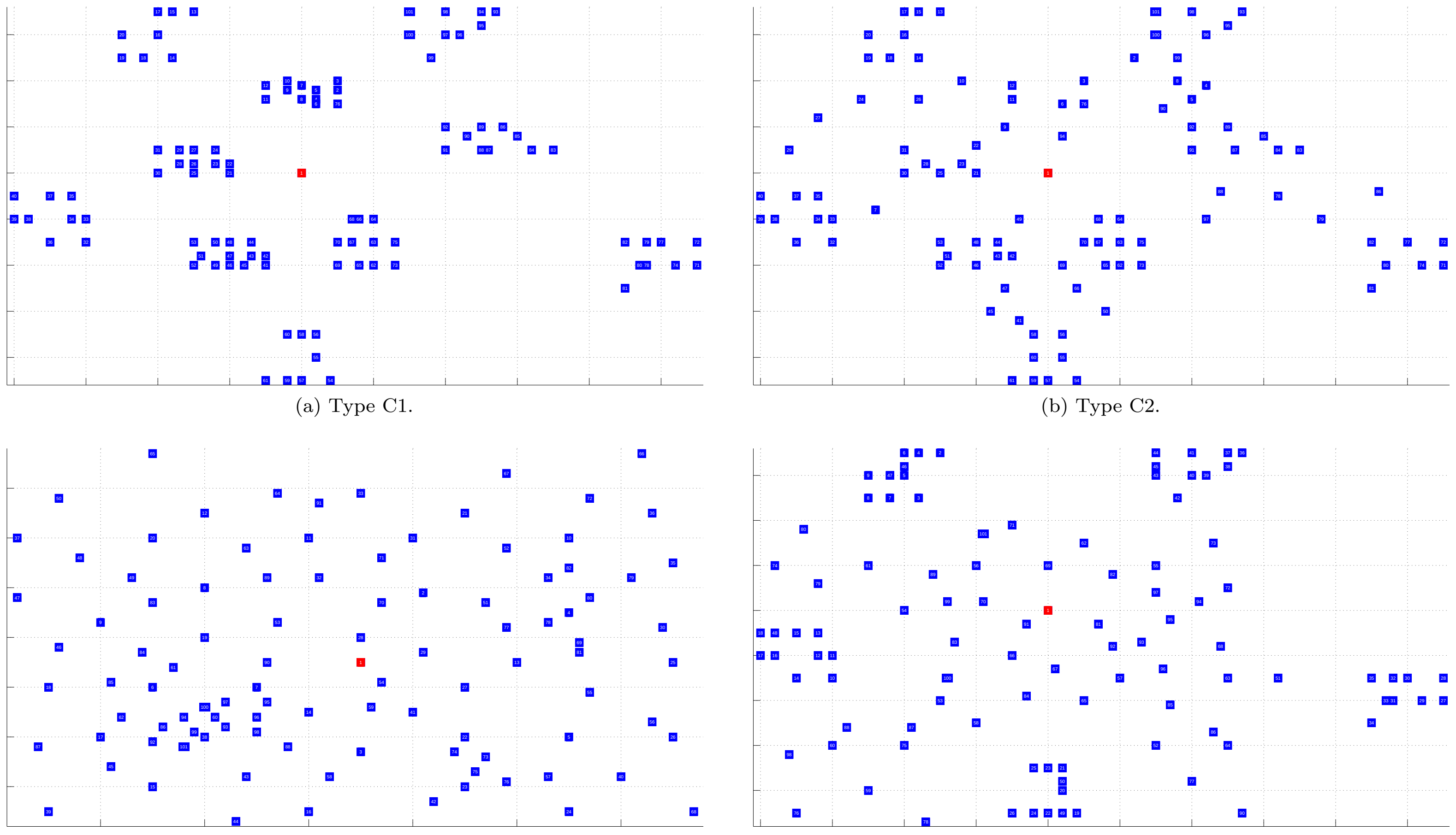
Numerical Results for VRP

D.1 Introduction

The results for the references and the prototype solution methods for the Capacitated Vehicle Routing Problem with Time Windows are shown below. The methods are described in Chapter 4. The Solomon CVRPTW benchmark instances are used for this comparison. The locations of the nodes of these instances are also depicted in this appendix.

The specifications of the system are: 2.60GHz Intel Core i7-3720QM processor, 8GB RAM and a Windows 7 64-bit operating system. Used software are: Matlab 2007b, version 7.5.0.342 and Visual Studio 2010 Professional, version 10.0.40219.1 SP1Rel. The Dynamic Programming algorithms are implemented in C++.

D.2 Instances



(a) Type C1.

(b) Type C2.

(c) Class R.

(d) Class RC.

Figure D.2.1: Locations of nodes of the Solomon CVRPTW instances. The depot is shown in red.

D.3

Literature Benchmarks

D.3.1

Literature Benchmarks

Set Covering LP

Name	**Best**	**Reference 1**			**Reference 2**			**Prototype**		
	value	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*
C101	827.30	920.0581	0.1121	**5.7474**	**828.9369**	**0.0020**	9.9805	**828.9369**	**0.0020**	6.2143
C102	827.30	1034.9419	0.2510	80.6282	865.6583	0.0464	88.4545	**835.5980**	**0.0100**	**21.2869**
C103	826.30	**1015.5618**	**0.2290**	97.4934	1051.9620	0.2731	12.4432	1051.9620	0.2731	**8.5848**
C104	822.90	983.5849	0.1953	102.1952	1195.3427	0.4526	**12.9386**	**862.6565**	**0.0483**	127.5980
C105	827.30	1001.1101	0.2101	17.3528	**828.9369**	**0.0020**	9.8492	**828.9369**	**0.0020**	**6.8478**
C106	827.30	1094.8779	0.3234	**5.9266**	**828.9369**	**0.0020**	10.2664	**828.9369**	**0.0020**	6.9770
C107	827.30	971.9691	0.1749	80.1827	**828.9369**	**0.0020**	10.7266	**828.9369**	**0.0020**	**7.2634**
C108	827.30	972.4816	0.1755	88.5297	**828.9369**	**0.0020**	10.9316	**828.9369**	**0.0020**	**7.5106**
C109	827.30	949.0114	0.1471	107.0149	889.7064	0.0754	**12.3491**	**864.5740**	**0.0451**	19.9521
			0.2020	**65.0079**		**0.0953**	**19.7711**		**0.0429**	**23.5816**
C201	589.10	744.1197	0.2631	14.8697	**603.8793**	**0.0251**	16.6900	**603.8793**	**0.0251**	**11.4010**
C202	589.10	719.7096	0.2217	15.9573	**692.6130**	**0.1757**	16.0619	**692.6130**	**0.1757**	**11.5504**
C203	588.70	666.6374	0.1324	**11.5101**	634.7971	0.0783	18.4482	**629.3111**	**0.0690**	25.0173
C204	588.10	890.9590	0.5150	**10.3525**	**823.8511**	**0.4009**	18.2440	**823.8511**	**0.4009**	13.1529
C205	586.40	734.7980	0.2531	**8.8012**	**608.8857**	**0.0383**	16.8847	**608.8857**	**0.0383**	12.7918
C206	586.00	845.9514	0.4436	128.9109	**690.1640**	**0.1778**	16.4228	**690.1640**	**0.1778**	**12.0916**
C207	585.80	688.4375	0.1752	**8.4295**	**636.5574**	**0.0866**	17.1149	**636.5574**	**0.0866**	12.9866
C208	585.80	861.9577	0.4714	**9.2507**	**647.3751**	**0.1051**	17.5663	**647.3751**	**0.1051**	14.1485
			0.3094	**26.0102**		**0.1360**	**17.1791**		**0.1348**	**14.1425**
			0.2526	**46.6561**		**0.1144**	**18.5513**		**0.0862**	**19.1397**

Table D.3.1: Set Covering LP relaxation results for Solomon CVRPTW benchmark instances (1 of 3).

Name	**Optimal**	**Reference 1**			**Reference 2**			**Prototype**		
	value	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*
R101	1637.70	**1696.2570**	**0.0358**	66.6536	1792.3631	0.0944	**7.8891**	1719.1396	0.0497	29.8241
R102	1466.60	1605.8153	0.0949	71.3376	1748.9952	0.1926	**9.9465**	**1487.9023**	**0.0145**	79.5850
R103	1208.70	1305.6230	0.0802	84.3870	1502.0406	0.2427	**10.1042**	**1254.6178**	**0.0380**	123.6915
R104	971.50	**1052.7138**	**0.0836**	100.2139	1202.8769	0.2382	10.9326	1202.8769	0.2382	**8.1032**
R105	1355.30	1411.7404	0.0416	85.1726	1575.2935	0.1623	**9.8670**	**1361.0957**	**0.0043**	113.1680
R106	1234.60	1310.2327	0.0613	98.0439	1488.2773	0.2055	**10.7173**	**1268.2862**	**0.0273**	137.2100
R107	1064.60	1139.1407	0.0700	101.7933	**1109.3820**	**0.0421**	176.0128	1355.9025	0.2736	**8.2014**
R108	960.88	**969.1672**	**0.0086**	106.6211	971.3092	0.0109	161.9830	1181.2771	0.2294	**8.2814**
R109	1146.90	1198.7518	0.0452	**92.4407**	1180.7693	0.0295	167.0844	**1157.2917**	**0.0091**	131.2294
R110	1068.00	**1111.0677**	**0.0403**	103.1403	1229.1753	0.1509	**11.6152**	1227.2169	0.1491	15.2848
R111	1048.70	1098.5624	0.0475	102.2460	1294.0756	0.2340	**11.2332**	**1082.7297**	**0.0324**	141.1726
R112	982.14	969.1135	**−0.0133**	108.6762	1108.4248	0.1286	11.9137	1108.4248	0.1286	**8.6281**
			0.0497	**93.3939**		**0.1443**	**49.9416**		**0.0995**	**67.0316**
R201	1143.20	1495.9526	0.3086	184.1237	**1372.3570**	**0.2005**	16.0619	**1372.3570**	**0.2005**	**11.8534**
R202	1191.70	1427.8681	0.1982	198.0186	**1331.9837**	**0.1177**	17.5343	**1331.9837**	**0.1177**	**13.2229**
R203	939.54	1466.2582	0.5606	**13.6054**	**1233.3621**	**0.3127**	19.8089	**1233.3621**	**0.3127**	16.0227
R204	825.52	**977.2770**	**0.1838**	214.0262	995.9679	0.2065	24.0598	995.9679	0.2065	**16.5875**
R205	994.42	1749.3088	0.7591	**13.4554**	**1264.2354**	**0.2713**	17.4977	**1264.2354**	**0.2713**	13.5589
R206	906.14	1243.2831	0.3721	224.4616	**1163.4477**	**0.2840**	20.1343	**1163.4477**	**0.2840**	**15.3997**
R207	893.33	1099.2497	0.2305	223.6603	**1034.2097**	**0.1577**	20.1449	**1034.2097**	**0.1577**	**15.3660**
R208	726.75	**881.8782**	**0.2135**	230.7342	894.7692	0.2312	22.0627	894.7692	0.2312	**17.6869**
R209	909.16	1423.8198	0.5661	**13.6049**	**1167.9812**	**0.2847**	19.3985	**1167.9812**	**0.2847**	15.2862
R210	939.34	1231.4419	0.3110	209.3396	**1200.0314**	**0.2775**	25.5190	1256.8760	0.3380	**15.5318**
R211	892.71	**990.2499**	**0.1093**	220.4352	1012.1490	0.1338	22.3528	1012.1490	0.1338	**16.6343**
			0.3466	**158.6786**		**0.2252**	**20.4159**		**0.2307**	**15.1955**
			0.1917	**124.6170**		**0.1830**	**35.8206**		**0.1623**	**42.2404**

Table D.3.2: Set Covering LP relaxation results for Solomon CVRPTW benchmark instances (2 of 3).

Name	**Optimal**	**Reference 1**			**Reference 2**			**Prototype**		
	value	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*
RC101	1619.80	1692.0348	0.0446	**71.2329**	1639.1829	0.0120	138.1216	**1610.6398**	**−0.0057**	107.4525
RC102	1457.40	1508.1593	0.0348	**66.3855**	1476.2278	0.0129	153.5908	**1463.8599**	**0.0044**	118.6899
RC103	1258.00	**1307.7180**	**0.0395**	90.5611	1309.2702	0.0408	155.4139	1596.6180	0.2692	**7.6264**
RC104	1135.48	**1190.0738**	**0.0481**	88.2461	1390.9875	0.2250	11.5520	1390.9875	0.2250	**8.2835**
RC105	1513.70	1612.4433	0.0652	**66.8052**	1546.4459	0.0216	144.7520	**1520.4420**	**0.0045**	109.0309
RC106	1424.73	1388.3492	−0.0255	**91.3406**	1369.1050	−0.0390	157.2343	**1357.1894**	**−0.0474**	127.9212
RC107	1207.80	**1257.4744**	**0.0411**	93.4317	1503.7860	0.2451	10.6241	1503.7860	0.2451	**8.1585**
RC108	1114.20	1128.3682	0.0127	**89.5851**	1100.6278	−0.0122	168.5245	**1098.1686**	**−0.0144**	135.0489
			0.0326	**82.1985**		**0.0633**	**117.4766**		**0.0851**	**77.7765**
RC201	1261.80	1736.5030	0.3762	155.2416	**1571.8889**	**0.2458**	14.1889	**1571.8889**	**0.2458**	**10.0181**
RC202	1092.30	1945.1237	0.7808	**10.8216**	**1435.9169**	**0.3146**	17.3034	**1435.9169**	**0.3146**	12.9641
RC203	1049.62	**1277.7196**	**0.2173**	184.1666	1319.6827	0.2573	18.9174	1319.6827	0.2573	**14.1627**
RC204	798.41	1011.6849	0.2671	168.2929	**996.5987**	**0.2482**	19.2468	**996.5987**	**0.2482**	**15.8210**
RC205	1154.00	1562.7843	0.3542	156.4432	1481.4123	0.2837	**16.0827**	**1459.3413**	**0.2646**	17.5577
RC206	1146.32	1545.0979	0.3479	183.6862	**1358.8069**	**0.1854**	17.2362	**1358.8069**	**0.1854**	**13.5368**
RC207	1061.14	1776.8551	0.6745	**12.4334**	**1273.7763**	**0.2004**	23.1846	1328.6388	0.2521	14.7281
RC208	828.14	1093.5909	0.3205	199.0409	1037.6105	0.2529	**19.5066**	**1031.0002**	**0.2450**	20.1896
			0.4173	**133.7658**		**0.2485**	**18.2083**		**0.2516**	**14.8723**
			0.2249	**107.9822**		**0.1559**	**67.8425**		**0.1683**	**46.3244**

Table D.3.3: Set Covering LP relaxation results for Solomon CVRPTW benchmark instances (3 of 3).

D.3.2

Literature Benchmarks

Set Partition IP

Name	**Optimal**	**Reference 1**			**Reference 2**			**Prototype**		
	value	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*
C101	827.30	881.9389	0.0660	**5.7829**	**828.9369**	**0.0020**	9.9925	**828.9369**	**0.0020**	6.2197
C102	827.30	1156.7253	0.3982	83.9175	921.1193	0.1134	88.5080	**835.5980**	**0.0100**	**21.2981**
C103	826.30	1116.9900	0.3518	99.7294	**1051.9620**	**0.2731**	12.4508	**1051.9620**	**0.2731**	**8.5921**
C104	822.90	1121.4787	0.3628	114.0244	1195.3427	0.4526	**12.9467**	**897.2357**	**0.0903**	127.6994
C105	827.30	1024.8304	0.2388	17.4296	**828.9369**	**0.0020**	9.8568	**828.9369**	**0.0020**	**6.8540**
C106	827.30	1056.5581	0.2771	**5.9682**	**828.9369**	**0.0020**	10.2744	**828.9369**	**0.0020**	6.9846
C107	827.30	1013.1015	0.2246	80.8979	**828.9369**	**0.0020**	10.7346	**828.9369**	**0.0020**	**7.2697**
C108	827.30	1074.6342	0.2990	90.8224	**828.9369**	**0.0020**	10.9393	**828.9369**	**0.0020**	**7.5182**
C109	827.30	1093.6752	0.3220	120.8679	889.7064	0.0754	**12.3565**	**864.5740**	**0.0451**	19.9626
			0.2823	**68.8267**		**0.1027**	**19.7844**		**0.0476**	**23.5998**
C201	589.10	719.9268	0.2221	14.9383	**603.8793**	**0.0251**	16.6976	**603.8793**	**0.0251**	**11.4075**
C202	589.10	729.8636	0.2389	16.2771	**692.6130**	**0.1757**	16.0694	**692.6130**	**0.1757**	**11.5575**
C203	588.70	**621.1695**	**0.0552**	**11.5286**	634.7971	0.0783	18.4546	634.7971	0.0783	25.0291
C204	588.10	848.7887	0.4433	**10.3844**	**823.8511**	**0.4009**	18.2518	**823.8511**	**0.4009**	13.1604
C205	586.40	706.5693	0.2049	**8.8306**	**608.8857**	**0.0383**	16.8912	**608.8857**	**0.0383**	12.7984
C206	586.00	909.9315	0.5528	130.1695	**690.1640**	**0.1778**	16.4297	**690.1640**	**0.1778**	**12.0979**
C207	585.80	688.4375	0.1752	**8.4338**	**636.5574**	**0.0866**	17.1213	**636.5574**	**0.0866**	12.9931
C208	585.80	861.9577	0.4714	**9.2559**	**647.3751**	**0.1051**	17.5727	**647.3751**	**0.1051**	14.1542
			0.2955	**26.2273**		**0.1360**	**17.1860**		**0.1360**	**14.1497**
			0.2885	**48.7799**		**0.1184**	**18.5616**		**0.0892**	**19.1527**

Table D.3.4: Set Partition IP results for Solomon CVRPTW benchmark instances (1 of 3).

Name	Optimal	Reference 1			Reference 2			Prototype		
	value	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*
R101	1637.70	1733.5798	0.0585	67.3280	1792.3631	0.0944	**7.8927**	**1719.1396**	**0.0497**	29.8434
R102	1466.60	1618.6510	0.1037	71.9672	1748.9952	0.1926	**9.9509**	**1487.9023**	**0.0145**	79.6061
R103	1208.70	1385.7178	0.1465	96.9925	1502.0406	0.2427	**10.1138**	**1286.0493**	**0.0640**	124.6682
R104	971.50	**1153.1099**	**0.1869**	150.3742	1202.8769	0.2382	10.9404	1202.8769	0.2382	**8.1111**
R105	1355.30	1521.2751	0.1225	152.6641	1575.2935	0.1623	**9.8756**	**1376.8070**	**0.0159**	113.5053
R106	1234.60	1379.7296	0.1176	102.4082	1488.2773	0.2055	**10.7254**	**1303.2786**	**0.0556**	142.2626
R107	1064.60	1215.9269	0.1421	113.0096	**1149.4111**	**0.0797**	178.2553	1355.9025	0.2736	**8.2108**
R108	960.88	1101.2825	0.1461	505.1398	**1074.0539**	**0.1178**	193.1561	1181.2771	0.2294	**8.2893**
R109	1146.90	1291.1158	0.1257	**126.4762**	1266.6809	0.1044	208.0820	**1174.0775**	**0.0237**	132.3518
R110	1068.00	**1215.2878**	**0.1379**	150.6315	1229.1753	0.1509	**11.6235**	1227.2169	0.1491	15.2960
R111	1048.70	1174.2513	0.1197	116.4556	1294.0756	0.2340	**11.2417**	**1160.1837**	**0.1063**	152.6036
R112	982.14	**1087.3264**	**0.1071**	466.4446	1108.4248	0.1286	11.9221	1108.4248	0.1286	**8.6359**
			0.1262	**176.6576**		**0.1626**	**56.1483**		**0.1124**	**68.6153**
R201	1143.20	1825.2055	0.5966	264.0118	**1372.3570**	**0.2005**	16.0707	**1372.3570**	**0.2005**	**11.8616**
R202	1191.70	1493.5698	0.2533	205.9658	**1331.9837**	**0.1177**	17.5428	**1331.9837**	**0.1177**	**13.2312**
R203	939.54	1349.2474	0.4361	**13.6603**	**1233.3621**	**0.3127**	19.8183	**1233.3621**	**0.3127**	16.0315
R204	825.52	1236.8700	0.4983	283.5752	**995.9679**	**0.2065**	24.0690	**995.9679**	**0.2065**	**16.5965**
R205	994.42	1560.6120	0.5694	**13.5104**	**1256.7316**	**0.2638**	17.5334	**1256.7316**	**0.2638**	13.5911
R206	906.14	1558.1174	0.7195	254.2569	**1163.4477**	**0.2840**	20.1427	**1163.4477**	**0.2840**	**15.4072**
R207	893.33	1323.4876	0.4815	274.0620	**1034.2097**	**0.1577**	20.1532	**1034.2097**	**0.1577**	**15.3745**
R208	726.75	1052.1960	0.4478	236.7438	**894.7692**	**0.2312**	22.0696	**894.7692**	**0.2312**	**17.6945**
R209	909.16	1390.9588	0.5299	**13.6248**	**1167.9812**	**0.2847**	19.4076	**1167.9812**	**0.2847**	15.2944
R210	939.34	1639.6865	0.7456	250.6328	**1200.0314**	**0.2775**	25.5292	1256.8760	0.3380	**15.5406**
R211	892.71	1291.2904	0.4465	244.6869	**1012.1490**	**0.1338**	22.3612	**1012.1490**	**0.1338**	**16.6439**
			0.5204	**186.7937**		**0.2245**	**20.4271**		**0.2300**	**15.2061**
			0.3147	**181.5053**		**0.1922**	**39.0642**		**0.1687**	**43.0718**

Table D.3.5: Set Partition IP results for Solomon CVRPTW benchmark instances (2 of 3).

Name	**Optimal**	**Reference 1**			**Reference 2**			**Prototype**		
	value	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*	*value*	*gap*	*time (s)*
RC101	1619.80	1823.9064	0.1260	**76.7834**	1705.9400	0.0532	138.9806	**1677.8706**	**0.0359**	107.9613
RC102	1457.40	1591.8436	0.0922	**68.1133**	1567.0830	0.0753	154.7585	**1532.7426**	**0.0517**	119.1703
RC103	1258.00	**1413.4984**	**0.1236**	93.5521	1434.7924	0.1405	160.2667	1596.6180	0.2692	**7.6351**
RC104	1135.48	**1290.8162**	**0.1368**	91.9064	1390.9875	0.2250	11.5600	1390.9875	0.2250	**8.2913**
RC105	1513.70	1705.3824	0.1266	**67.6190**	1693.9210	0.1191	145.2237	**1611.1490**	**0.0644**	109.1853
RC106	1424.73	1517.0055	0.0648	**100.5382**	1490.0049	0.0458	159.0010	**1456.5374**	**0.0223**	129.6014
RC107	1207.80	**1444.2364**	**0.1958**	236.8438	1503.7860	0.2451	10.6323	1503.7860	0.2451	**8.1668**
RC108	1114.20	1311.8343	0.1774	**102.5117**	1258.0025	0.1291	181.2375	**1240.3241**	**0.1132**	136.8945
			0.1304	**104.7335**		**0.1291**	**120.2075**		**0.1283**	**78.3632**
RC201	1261.80	1926.2170	0.5266	161.2833	**1571.8889**	**0.2458**	14.1965	**1571.8889**	**0.2458**	**10.0259**
RC202	1092.30	1814.8052	0.6615	**10.8547**	**1435.9169**	**0.3146**	17.3110	**1435.9169**	**0.3146**	12.9720
RC203	1049.62	1326.9412	0.2642	190.3764	**1319.6827**	**0.2573**	18.9264	**1319.6827**	**0.2573**	**14.1705**
RC204	798.41	1082.5630	0.3559	173.4263	**996.5987**	**0.2482**	19.2540	**996.5987**	**0.2482**	**15.8284**
RC205	1154.00	1754.2913	0.5202	160.3699	1481.4123	0.2837	**16.0909**	**1459.3413**	**0.2646**	17.5685
RC206	1146.32	1747.0704	0.5241	186.8380	**1358.8069**	**0.1854**	17.2438	**1358.8069**	**0.1854**	**13.5444**
RC207	1061.14	1673.0339	0.5766	**12.4532**	**1273.7763**	**0.2004**	23.1931	1328.6388	0.2521	14.7357
RC208	828.14	1377.2943	0.6631	211.4114	1037.6105	0.2529	**19.5148**	**1031.0002**	**0.2450**	20.1987
			0.5115	**138.3766**		**0.2485**	**18.2163**		**0.2516**	**14.8805**
			0.3210	**121.5551**		**0.1888**	**69.2119**		**0.1900**	**46.6219**

Table D.3.6: Set Partition IP results for Solomon CVRPTW benchmark instances (3 of 3).

D.3.3

Literature Benchmarks

Overview

Type	**Reference 1**		**Reference 2**		**Prototype**	
	gap	*time (s)*	*gap*	*time (s)*	*gap*	*time (s)*
C1	0.2020	65.0079	0.0953	**19.7711**	**0.0429**	23.5816
C2	0.3094	26.0102	0.1360	17.1791	**0.1348**	**14.1425**
R1	**0.0497**	93.3939	0.1443	**49.9416**	0.0995	67.0316
R2	0.3466	158.6786	**0.2252**	20.4159	0.2307	**15.1955**
RC1	**0.0326**	82.1985	0.0633	117.4766	0.0851	**77.7765**
RC2	0.4173	133.7658	**0.2485**	18.2083	0.2516	**14.8723**
C	0.2526	46.6561	0.1144	**18.5513**	**0.0862**	19.1397
R	0.1917	124.6170	0.1830	**35.8206**	**0.1623**	42.2404
RC	0.2249	107.9822	**0.1559**	67.8425	0.1683	**46.3244**
	0.2197	**96.1975**	**0.1544**	**39.7272**	**0.1409**	**36.3945**

Table D.3.7: Overall Set Covering LP results for Solomon CVRPTW benchmark instances.

Type	**Reference 1**		**Reference 2**		**Prototype**	
	gap	*time (s)*	*gap*	*time (s)*	*gap*	*time (s)*
C1	0.2823	68.8267	0.1027	**19.7844**	**0.0476**	23.5998
C2	0.2955	26.2273	**0.1360**	17.1860	**0.1360**	**14.1497**
R1	0.1262	176.6576	0.1626	**56.1483**	**0.1124**	68.6153
R2	0.5204	186.7937	**0.2245**	20.4271	0.2300	**15.2061**
RC1	0.1304	104.7335	0.1291	120.2075	**0.1283**	**78.3632**
RC2	0.5115	138.3766	**0.2485**	**18.2163**	0.2516	**14.8805**
C	0.2885	48.7799	0.1184	**18.5616**	**0.0892**	19.1527
R	0.3147	181.5053	0.1922	**39.0642**	**0.1687**	43.0718
RC	0.3210	121.5551	**0.1888**	69.2119	0.1900	**46.6219**
	0.3085	**124.0850**	**0.1688**	**41.4538**	**0.1506**	**36.8250**

Table D.3.8: Overall Set Partition IP results for Solomon CVRPTW benchmark instances.