

Three Essays on Sequencing and Routing Problems

by

Agustín Bompadre

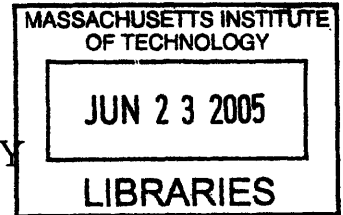
Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

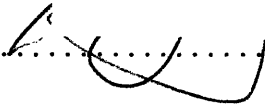
at the

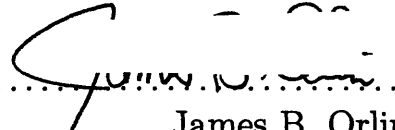
MASSACHUSETTS INSTITUTE OF TECHNOLOGY


June 2005



© Massachusetts Institute of Technology 2005. All rights reserved.

Author 
Sloan School of Management
June 12, 2005

Certified by 
James B. Orlin
Edward Pennell Brooks Professor of Operations Research
Co-director, Operations Research Center
Thesis Supervisor

Accepted by 
John N. Tsitsiklis
Co-director, Operations Research Center

ARCHIVES

Three Essays on Sequencing and Routing Problems

by

Agustín Bompadre

Submitted to the Sloan School of Management
on June 12, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis we study different combinatorial optimization problems. These problems arise in many practical settings where there is a need for finding good solutions fast. The first class of problems we study are vehicle routing problems, and the second type of problems are sequencing problems. We study approximation algorithms and local search heuristics for these problems.

First, we analyze the Vehicle Routing Problem (VRP) with and without split deliveries. In this problem, we have to route vehicles from the depot to deliver the demand to the customers while minimizing the total traveling cost. We present a lower bound for this problem, improving a previous bound of Haimovich and Rinnooy Kan. This bound is then utilized to improve the worst-case approximation algorithm of the Iterated Tour Partitioning (ITP) heuristic when the capacity of the vehicles is constant.

Second, we analyze a particular case of the VRP, when the customers are uniformly distributed i.i.d. points on the unit square of the plane, and have unit demand. We prove that there exists a constant $c > 0$ such that the ITP heuristic is a $2 - c$ approximation algorithm with probability arbitrarily close to one as the number of customers goes to infinity. This result improves the approximation factor of the ITP heuristic under the worst-case analysis, which is 2. We also generalize this result and previous ones to the multi-depot case.

Third, we study a language to generate Very Large Scale Neighborhoods for sequencing problems. Local search heuristics are among the most popular approaches to solve hard optimization problems. Among them, Very Large Scale Neighborhood Search techniques present a good balance between the quality of local optima and the time to search a neighborhood. We develop a language to generate exponentially large neighborhoods for sequencing problems using grammars. We develop efficient generic dynamic programming solvers that determine the optimal neighbor in a neighborhood generated by a grammar for a list of sequencing problems, including the Traveling Salesman Problem and the Linear Ordering Problem. This framework unifies a variety of previous results on exponentially large neighborhoods for the Traveling Salesman Problem.

Thesis Supervisor: James B. Orlin
Title: Edward Pennell Brooks Professor of Operations Research
Co-director, Operations Research Center

Acknowledgments

I would like to thank my advisor, Professor Jim Orlin, for his guidance and support during my four years at MIT. His originality of thought and ability of capturing the essential things have always amazed me.

I would also like to thank Professors Moshe Dror and Andreas Schulz for being part of the thesis committee. Both contributed with valuable inputs to improve my research. Moshe was always ready to listen and to encourage me during my last two years. His intellectual curiosity and passion for Operations Research are a model for me.

I would like to thank all the people who make the ORC a great place to work. The list is quite extensive, and it starts with the soccer lovers. Alexandre, Felipe, Kwong Meng, Mike W, Rags, Victor and Yann were the backbone of the ORC soccer team that made it into the semi-finals of the intramural B league during my third year. My classmates Carol, Elodie, Lincoln, Maxime, Michele, Mike W, Susan and Ping helped me in one way or another during these years. Susan: thanks for teaching me how to stop when I ski, and thus my avoiding a number of life-threatening accidents.

Outside the ORC, I would like to thank Eva and Alexander for being my friends. My family has encouraged me to pursue a PhD, and I thank them too.

Overall, I would like to thank Malena. Her love and support are the driving forces behind all my enterprises, including this thesis.

Contents

1	Introduction	15
1.1	Worst-Case Analysis of Vehicle Routing Problems	15
1.2	Probabilistic Analysis of Unit Demand Euclidean Vehicle Routing Problems	17
1.3	Using Grammars to Generate Very Large Scale Neighborhood Search for Sequencing Problems	18
2	Improved Bounds for Vehicle Routing Solutions	21
2.1	Introduction	21
2.1.1	Capacitated Routing Problems	21
2.1.2	Notation	24
2.2	Lower bounds on the optimal cost of the VRP	25
2.3	Approximation Algorithms for SDVRP	32
2.3.1	Iterated Tour Partitioning (ITP) heuristic	32
2.3.2	Quadratic Iterated Tour Partitioning heuristics	35
2.4	Approximation Algorithm for CVRP	41
2.5	Conclusions	45
3	Probabilistic Analysis of Unit Demand Vehicle Routing Problems	47
3.1	Introduction	47
3.1.1	Unit Demand Vehicle Routing Problem	47
3.1.2	Notation	49
3.2	Lower bounds on the optimal cost of the VRP	50

3.3	Probabilistic analysis	53
3.4	Multi-Depot Vehicle Routing Problem	59
3.4.1	Notation	60
3.5	An algorithm for the Multi-Depot VRP	60
3.6	Lower bounds on the optimal cost of Multi-Depot VRP	61
3.7	Probabilistic analysis of MDVRP	64
3.7.1	Probabilistic analysis of lower bounds	64
3.7.2	Probabilistic analysis of an algorithm for MDVRP	67
3.8	Conclusions	68
4	Using Grammars to Generate Very Large Scale Neighborhoods for Sequencing Problems	69
4.1	Introduction	69
4.2	Notation and terminology	71
4.2.1	Sequencing Problems	71
4.2.2	Neighborhoods	74
4.2.3	Very Large Scale Neighborhood Search	75
4.2.4	Grammar terminology	76
4.2.5	Sequence Grammar	79
4.3	Neighborhoods Generated by Sequence Grammars	80
4.3.1	Polynomial Neighborhoods	80
4.3.2	Exponential Neighborhoods	81
4.3.3	Exponential Neighborhoods as Sequencing Neighborhood Grammars	85
4.3.4	Compound Neighborhoods	97
4.3.5	Context-Sensitive sequence grammars	98
4.4	Algorithms	99
4.4.1	A generic solution procedure for TSP grammars using dynamic programming	99
4.4.2	Incrementality	105

4.4.3	Running times for the Generic Algorithm on TSP grammars	106
4.4.4	A generic solution procedure for Linear Ordering Grammars	111
4.4.5	A generic solution procedure for Minimum Latency Grammars	112
4.4.6	A generic solution procedure for Scheduling Grammars	114
4.4.7	A generic solution procedure for Weighted Bipartite Matching Problem with Side Constraint Grammars	118
4.4.8	The Shortest Hyperpath Problem	120
4.4.9	Determining whether a permutation σ is in a sequence grammar	122
4.4.10	Context-Sensitive Sequence Grammars for the TSP	123
4.5	Complexity questions	127
4.5.1	Hierarchy of Sequence Grammars	128
4.5.2	Sufficient condition for a Context-Free grammar to be unambiguous	137
4.5.3	Ambiguity, Intersection and Inclusion for regular grammars	138
4.5.4	Testing properties of Sequence Grammars	142
4.6	Inverse Neighborhood	147
4.6.1	Inverse Balas-Simonetti Neighborhood	148
4.6.2	Optimizing in the inverse Neighborhood of a Sequence Grammar for the TSP	150
4.6.3	Running times for optimizing on the inverse neighborhood	153
4.7	Conclusions and Open problems	155

List of Figures

- 2-1 Quadratic Iterated Tour Partitioning heuristic 36
- 2-2 Quadratic Unequal Iterated Tour Partitioning heuristic 42
- 3-1 Assigning customers to their closest depot. 61
- 4-1 Hierarchy of sequence grammars 128

List of Tables

2.1	Approximation ratio for SDVRP for small Q when $\alpha = 1$	39
2.2	Approximation ratio for SDVRP for small Q when $\alpha = \frac{3}{2}$	39
2.3	Approximation ratio for CVRP for small Q when $\alpha = 1$	45
2.4	Approximation ratio for CVRP for small Q when $\alpha = \frac{3}{2}$	45
4.1	Running times for the generic dynamic programming algorithm for TSP neighborhoods.	107

Chapter 1

Introduction

In this thesis we study approximation algorithms and heuristics for different combinatorial optimization problems. The first class of problems we study are vehicle routing problems. For these problems, we present new lower bounds for the cost of the optimal solutions, and we perform a worst-case and a probabilistic analysis of algorithms.

The second class of problems we study are sequencing problems. We analyze local search heuristics for these problems. In particular, we develop a modeling language based on grammars to generate Very Large Scale Neighborhood (VLSN) for sequencing problems.

1.1 Worst-Case Analysis of Vehicle Routing Problems

In the Vehicle Routing Problem (VRP), there are n customers with demand to be satisfied by a fleet of unlimited and identical vehicles, which are located at a depot. The route of each vehicle starts and ends at the depot, and each vehicle cannot deliver more than its capacity Q . The cost of a solution is the sum of the travel cost of each vehicle. The problem considered is to route vehicles to deliver the demand of every customer while minimizing the overall cost.

A number of variations of the VRP have been studied in the literature. Dror and Trudeau [25] analyzed the Split Delivery Vehicle Routing Problem (SDVRP), where the demand of

a customer can be delivered by more than one vehicle (split delivery). Since a customer i with demand q_i can be considered as q_i customers with unit demand and zero interdistance, the SDVRP can be reduced to a unit demand vehicle routing problem, which is also known as the Capacitated Vehicle Routing Problem with equal demand (ECVRP). However, we observe that this reduction is pseudopolynomial.

When split demand is not allowed the problem is known as the Capacitated Vehicle Routing Problem (CVRP). We use the term “Vehicle Routing Problem” when we do not want to distinguish among these specific problems.

The complexity of solving the VRP depends on the capacity of the vehicles, and on the distance metric. For example, when the capacity of every vehicle is 2, the VRPs can be solved in polynomial time by transforming it to a minimum weight matching problem (see Asano et al. [9]). However, the ECVRP is APX-complete for any $Q \geq 3$ (see Asano et al. [8]), that is, there exists $\delta > 0$ such that no $1 + \delta$ approximation algorithm exists unless $P = NP$. The VRPs are closely related to the traveling salesman problem (TSP), and therefore many results for the TSP have been extended or adapted for the VRPs.

If the capacity is fixed, the Euclidean ECVRP admits a PTAS (polynomial time approximation scheme). That is, for every $\epsilon > 0$ there exists a polynomial-time algorithm with approximation guarantee at most $1 + \epsilon$. Haimovich and Rinnooy Kan [35] presented the first PTAS for the Euclidean ECVRP. The running time of their PTAS is (doubly) exponential in Q . In particular, it does not lead to a PTAS for the TSP. Later, Asano et al. [9] presented a faster PTAS, using the PTAS developed by Arora [7] for the Euclidean TSP.

In this paper, we study the VRPs in the metric case, when the capacity Q is fixed. We improve previous approximation results for the VRPs in this case. Improving the approximation ratio for fixed Q has a practical interest since some problems that arise in practice have small Q (see e.g. Bell et al. [14]).

Haimovich and Rinnooy Kan [35] started the worst-case and the probabilistic analysis of VRP. They presented a simple and very useful lower bound of the optimal cost of the VRP. They also presented the Iterated Tour Partitioning (ITP) heuristic for the ECVRP.

This heuristics was later improved and generalized to the CVRP by Altinkemer and Gavish [4, 5]. The primary contributions we present on Chapter 2 are the following. We provide nonlinear valid lower bounds for the optimal cost of VRP problems. For the VRPs mentioned before, we modify the two heuristics of Altinkemer and Gavish [4, 4] and we obtain better approximation algorithms for the SDVRP, the ECVRP and the CVRP, when the capacity Q is fixed. For the case when Q is not fixed, we also present an implementation of the Iterated Tour Partitioning (ITP) heuristic by Altinkemer and Gavish [5] for the SDVRP that runs in polynomial time. The previous implementation runs in pseudopolynomial time.

1.2 Probabilistic Analysis of Unit Demand Euclidean Vehicle Routing Problems

In Chapter 3, we perform a probabilistic analysis of the Euclidean ECVRP. In this problem, all customers have unit demand and are modeled as points in the unit square. The customers are independent identically distributed points. The ITP heuristic, described in Chapter 2, is analyzed in this setting. We also analyze the multi-depot VRP, where a number of depots are fixed in advance. Two papers that deal with the multi-depot VRP are the ones by Li and Simchi-Levi [44], and Stougie [55]. Li and Simchi-Levi [44] performed a worst-case analysis of the multi-depot vehicle routing problem when the distance satisfy the triangle inequality, and showed how to reduce it to a single-depot case with triangle inequality. Their analysis is general, and does not take special advantage of particular cases (e.g., the Euclidean case). Stougie [55] studied a two-stage multi-depot problem, where on the first stage we decide how many depots to build (and where), and on the second stage we deal with a multi-depot VRP. His analysis is probabilistic, since the customers of the second stage are i.i.d. points in the unit square. The objective in his problem is to minimize the sum of the costs of both stages.

Not only Haimovich and Rinnooy Kan [35] proposed the ITP heuristic and performed a worst-case analysis, but they also analyzed it in the probabilistic setting as well. They

show that, under technical conditions on Q and the number of customers, the ITP heuristic is asymptotically optimal. However, under no extra assumptions on Q , the ITP is still a 2-approximation algorithm.

The primary contributions of Chapter 3 are as follows. We present a probabilistic analysis of a lower bound proposed in Chapter 2, and we show that it improves upon the lower bound of Haimovich and Rinnooy Kan [35]. Based on this analysis, we improve the approximation bound of the ITP heuristic, showing that it is a $2 - \hat{c}$ approximation algorithm (for a constant $\hat{c} > 0$). for the VRP. In the second part of the chapter we extend these results to the multi-depot case.

1.3 Using Grammars to Generate Very Large Scale Neighborhood Search for Sequencing Problems

In Chapter 4, we study how to use grammar to define local search algorithms for a list of sequencing problems.

The traveling salesman problem and the linear ordering problem are examples of sequencing problems. Both problems are NP -hard, and thus there is a need of algorithms that find a good solution fast. Local search algorithms (see e.g., the book by Aarts and Lenstra [1]), and in particular Very Large Scale Neighborhood (VLSN) search, have been proposed and tested on a number of combinatorial optimization problems such as the sequencing problems studied in this chapter. Good sources on VLSN search are the papers by Ahuja et al. [3], Deĭneko and Woeginger [47], and Gutin et al. [34]. The former two papers are devoted to VLSN for the TSP, and thus they are closely related to the problems and techniques studied in this chapter.

Many of the VLSN defined for the TSP, that are surveyed in [47], rely on dynamic programming recursions to find the optimal solution in an neighborhood efficiently. In Chapter 4, we define the *sequence grammars* for sequencing problems, and show that this framework that unifies all these results. This approach is also useful for a list of sequencing problems

such as the linear ordering problem, the minimum latency problem, and a bipartite matching problem with side constraints. We also provide a generic dynamic programming algorithm for finding the best tour in a grammar induced neighborhood for each of these sequencing problems. For the TSP, when specialized to the neighborhoods given in Deĭneko and Woeginger [47], our generic algorithm achieves the same running time as the special purpose dynamic programs, except for the twisted sequence neighborhood. In that case, our generic DP improves upon the previous best bound by a factor of n .

The description of a variety of VLSN for sequencing problems using grammars we provide in Chapter 4 is a first step towards the development of a modeling language for VLSN search.

We also provide efficient algorithms for enumerating the size of neighborhoods and for deciding whether two neighborhoods are distinct in the case that the generating grammars are context free and unambiguous. We prove a number of theoretical results about the hierarchy of sequence grammars. In particular, we prove that a context free sequence grammar that generates the complete neighborhood must have an exponential number of production rules.

Chapter 2

Improved Bounds for Vehicle Routing Solutions

2.1 Introduction

2.1.1 Capacitated Routing Problems

Let $G = (V, E)$ be an undirected graph, where each edge (i, j) has a travel cost $c_{ij} \geq 0$. We assume that the cost matrix satisfies the triangle inequality. Node 1 is the depot node and the rest of the nodes $2, \dots, n$ are customers. Each node i different from 1 has demand $q_i \in \mathbb{N}$. There are identical vehicles with capacity $Q \in \mathbb{N}$ which are going to deliver the demand of each customer i . The route of each vehicle starts and ends at node 1, and each vehicle cannot deliver more than its capacity Q . The cost of a solution is the sum of the travel cost of each vehicle. The problem considered is to route vehicles to deliver the demand of every customer while minimizing the overall cost. If the demand of a customer can be delivered by more than one vehicle (split delivery), the problem is known as the Split Delivery Vehicle Routing Problem (SDVRP) (see Dror and Trudeau [25]). Since a customer i with demand q_i can be considered as q_i customers with unit demand and zero interdistance, the SDVRP can be reduced to a unit demand vehicle routing problem (this reduction is pseudopolynomial but it can be done implicitly), which is also known as the Capacitated Vehicle Routing Problem

with equal demand (ECVRP). When split demand is not allowed the problem is known as the Capacitated Vehicle Routing Problem (CVRP). When we do not want to distinguish between these variations, we talk about a Vehicle Routing Problem (VRP).

The complexity of solving the VRP depends on Q and on the travel cost. When $Q = 2$, the VRP can be solved in polynomial time by transforming it to a minimum weight matching problem (see Asano et al. [9]). However, the problem is NP -hard for any $Q \geq 3$. If Q is fixed, the Euclidean ECVRP admits a PTAS (polynomial time approximation scheme). The first PTAS for this case appeared in Haimovich and Rinnooy Kan [35]. The running time of this PTAS is (doubly) exponential in Q . In particular, it does not lead to a PTAS for the TSP. Subsequently, Asano et al. [9] improved the running time using the PTAS for the Euclidean TSP. The general metric case is APX-complete for any $Q \geq 3$ (see Asano et al. [8]), that is, there exists $\delta > 0$ such that no $1 + \delta$ approximation algorithm exists unless $P = NP$. Improving the approximation ratio for fixed Q has a practical interest since some problems that arise in practice have small Q (see Bell et al. [14]). Moreover, as Anily and Bramel [6] point out, the transportation of cars, industrial machinery or handicapped children are some examples where the capacity of the vehicles is small.

The worst case analysis and the probabilistic analysis of the VRP started with the work by Haimovich and Rinnooy Kan [35]. In their paper, a lower bound on the cost of the VRP is presented. Based on this bound, they derived approximation results. Subsequent papers (e.g., Altinkemer and Gavish [4, 5], Li and Simchi-Levi [44]) rely on this bound to improve or generalize approximation results for the VRP. Anily and Bramel [6] present approximation algorithms for a related problem, the capacitated TSP with pickups and deliveries. When applied to the SDVRP, its algorithm $MATCH^k$ has a worse case bound than the algorithm by Altinkemer and Gavish [5] for any capacity Q but $Q = 4$. The primary contributions of this chapter are as follows:

1. We provide nonlinear valid inequalities that are useful for improving bounds for VRP problems.

2. We improve the approximation results of Altinkemer and Gavish [4, 5] for the VRP with triangle inequality, with and without split deliveries. This improvement, though slight, does resolve a long standing open question of whether any improvement was possible.

3. We present an implementation of the Iterated Tour Partitioning (ITP) heuristic by Altinkemer and Gavish [5] for SDVRP that runs in polynomial time (the previous implementation runs in pseudopolynomial time when the capacity Q is part of the input).

When customers have unit demand (i.e., ECVRP), the ITP heuristic by Altinkemer and Gavish [5] receives an α -optimal traveling salesman tour as part of the input and outputs a solution with cost at most $1 + (1 - \frac{1}{Q})\alpha$ times the optimal solution. In particular, its approximation ratio depends on the approximability of the TSP. With respect to the approximability of the TSP with triangle inequality, the current best ratio is $\alpha = \frac{3}{2}$, obtained by Christofides' algorithm (see [20]). When the nodes are points on the plane and the travel cost is the Euclidean distance, α can be $1 + \epsilon$ for any $\epsilon > 0$ (see Arora [7] or Mitchell [46]). When customers have unequal demand and split deliveries are not allowed (CVRP), the Unequal-weight Iterated Tour Partitioning (UITP) heuristic by Altinkemer and Gavish [4] is a $2 + (1 - \frac{2}{Q})\alpha$ approximation algorithm assuming that Q is even and an α -optimal traveling salesman tour is part of the input. The survey by Haimovich and Rinnooy Kan [36] analyzes these algorithms from a worst case and from a probabilistic perspectives. An implementation of ITP for SDVRP runs in $O(S(n, Q))$ time, where $S(n, Q)$ is the time to sort n integers in the range $[1, Q]$, as we show in Section 2.3. In particular, this implies that UITP also runs in $O(S(n, Q))$ time. In the previous literature the SDVRP is reduced to the ECVRP via a pseudopolynomial transformation before using ITP.

We present the Quadratic Iterated Tour Partitioning (QITP) heuristics for the SDVRP and the Quadratic Unequal Iterated Tour Partitioning (QUITP) heuristics for the CVRP. The approximation ratio of the former is $1 - a(\alpha, Q) + (1 - \frac{1}{Q})\alpha$ and the approximation ratio of the latter is $2 - b(\alpha, Q) + (1 - \frac{2}{Q})\alpha$ (Q even in this case). The functions $a(\alpha, Q), b(\alpha, Q)$ are the improvements over the approximation ratio of ITP and UITP respectively. They

satisfy that $a(\alpha, Q), b(\alpha, Q) \geq \frac{1}{3Q^3}$ for any $\alpha \geq 1$, any $Q \geq 3$. When $\alpha = \frac{3}{2}$ and $Q \geq 3$, they satisfy that $a(\frac{3}{2}, Q) \geq \frac{1}{4Q^2}, b(\frac{3}{2}, Q) \geq \frac{1}{3Q^2}$. The running time of these algorithms is $O(n^2 \log n)$. The increase on the running time with respect to the running times of ITP and UITP is not the bottleneck operation when we consider that all these algorithms receive an α -optimal tour as part of their input and that the current best approximation algorithm for TSP runs in $O(n^3)$ time. The mechanics of QITP (QUITP respectively) are as follows: if the new quadratic lower bound is significantly larger than the old lower bound, then ITP (UITP respectively) improves its approximation guarantee ratio since we found a stronger lower bound. If the new quadratic lower bound is not substantially larger than the old one, we still gather some information to construct a solution of less cost.

The rest of the chapter is organized as follows: in Subsection 2.1.2 we introduce the notation to use throughout the chapter. In Section 2.2, we present some known and some new lower bounds on the optimal cost of the VRP. In Section 2.3, we present the QITP for the SDVRP, and we also present an implementation of ITP for SDVRP that runs in $O(S(n, Q))$ time. In Section 2.4, we present the QITP for the CVRP. Finally, we summarize our conclusions in Section 2.5.

2.1.2 Notation

A solution of a VRP is denoted by (K, v_k, d_i^k) where K is the number of vehicles used, v_k denotes the routing of the k th vehicle and d_i^k denotes the demand delivered by vehicle k to customer i . When split deliveries are not allowed, d_i^k is either 0 or q_i^k . The routing cost of vehicle k is denoted by $c(v_k)$. We assign to the terms routing and subtour the same meaning. We let $R = \sum_{i=2}^n 2c_{1i} \frac{q_i}{Q}$ denote the *radial distance* as per Haimovich and Rinnooy Kan [35]. We let $c(TSP)$ denote the cost of an optimal tour on G .

When we transform a Split Delivery Vehicle Routing Problem into a unit-demand problem, we replace each customer i with demand q_i by a clique of q_i customers with unit demand each and zero interdistance. Then we say that these q_i nodes represent the same original customer.

2.2 Lower bounds on the optimal cost of the VRP

In this section we present some known lower bounds and new lower bounds on the optimal cost of the VRP. To simplify notation, some of the bounds are presented for the unit-demand VRP. The bounds also hold for SDVRP and CVRP since we can transform (relax) a SDVRP (CVRP) into a unit-demand VRP. The main result of this section is the lower bound given in Theorem 2.2.14 which improves the lower bound given by Haimovich and Rinnooy Kan [35]. In the next Sections, we make use of this bound to improve the approximation ratio of the algorithms for SDVRP and CVRP by Altinkemer and Gavish [5] and [4] respectively.

The following lemma gives a lower bound on the cost of routing a vehicle.

Lemma 2.2.1. *Let W be any subtour that passes through the depot. Then,*

$$\sum_{(i,j) \in W} c_{ij} \geq \sum_{i \in W \setminus 1} \frac{2c_{1i}}{|W| - 1}.$$

Proof. By triangular inequality, $\sum_{(i,j) \in W} c_{ij} \geq \max\{2c_{1i} : i \in W \setminus 1\}$. Moreover, $\max\{2c_{1i} : i \in W \setminus 1\} \geq \sum_{i \in W \setminus 1} \frac{2c_{1i}}{|W| - 1}$ since the maximum c_{1i} among $i \in W \setminus 1$ is at least the average c_{1i} among $i \in W \setminus 1$. \square

This lemma implies the following lemma from Haimovich and Rinnooy Kan [35].

Lemma 2.2.2. ([35]) *The cost $c(v_k)$ of routing a vehicle v_k with capacity Q that delivers d_i^k units to customer i is at least $\sum_{i=2}^n 2c_{1i} \frac{d_i^k}{Q}$.*

Proof. Replace each customer i by d_i^k customers with zero interdistance. Therefore, the routing of vehicle v_k can be viewed as a subtour W that passes through the depot and through $\sum_{i=2}^n d_i^k \leq Q$ customers. By applying Lemma 2.2.1 to subtour W , we obtain the desired inequality. \square

Lemma 2.2.3. ([35]) *The cost $c(TSP)$ of an optimal tour is a lower bound on the cost of the VRP.*

Proof. Given an optimal solution of the VRP, we can construct a tour with lesser or equal cost by merging all subtours into a single tour by avoiding nodes already visited. The cost of the resulting tour is at most the cost of the VRP because of the triangle inequality. \square

From the previous two lemmas we obtain the following lower bound on the VRP (see Haimovich and Rinnooy Kan [35] or [36]).

Lemma 2.2.4. ([35]) *The cost of the optimal solution of the VRP is at least*

$$\max\left\{\sum_{i=2}^n 2c_{1i} \frac{q_i}{Q}; c(TSP)\right\}.$$

For customers i and j , let $W(i, j)$ be the subtour that passes through customers i, j and the depot only. If $i = j$ then $W(i, j)$ is the subtour including i and the depot only. Its cost is $c(W(i, j))$. The following lemma is key to our analysis.

Lemma 2.2.5. *Let W be any subtour that passes through the depot. Then,*

$$\sum_{(i,j) \in W} c_{ij} \geq \sum_{i,j \in W \setminus 1} \frac{c(W(i, j))}{(|W| - 1)^2}. \quad (2.1)$$

Proof. The proof is similar to the one of Lemma 2.2.1. By triangular inequality, $\sum_{(i,j) \in W} c_{ij} \geq \max\{c(W(i, j)) : i, j \in W \setminus 1\}$. Moreover, $\max\{c(W(i, j)) : i, j \in W \setminus 1\} \geq \sum_{i,j \in W \setminus 1} \frac{c(W(i, j))}{(|W| - 1)^2}$ since the maximum $c(W(i, j))$ among $i, j \in W \setminus 1$ is at least the average $c(W(i, j))$ among $i, j \in W \setminus 1$. \square

The lower bound obtained in this lemma can be expressed as follows.

Corollary 2.2.6. *Let W be any subtour that passes through the depot. Then,*

$$\sum_{(i,j) \in W} c_{ij} \geq \sum_{i \in W \setminus 1} \frac{2c_{1i}}{(|W| - 1)} + \sum_{i,j \in W \setminus 1} \frac{c_{ij}}{(|W| - 1)^2}. \quad (2.2)$$

Proof. Since the cost of subtour $W(i, j)$ is equal to $c_{1i} + c_{ij} + c_{1j}$, the lower bound (2.1) can

be rewritten as follows.

$$\begin{aligned} & \sum_{i,j \in W \setminus 1} \frac{c(W(i,j))}{(|W| - 1)^2} = \\ & \sum_{i,j \in W \setminus 1} \frac{c_{1i}}{(|W| - 1)^2} + \sum_{i,j \in W \setminus 1} \frac{c_{ij}}{(|W| - 1)^2} + \sum_{i,j \in W \setminus 1} \frac{c_{1j}}{(|W| - 1)^2} = \\ & \sum_{i \in W \setminus 1} \frac{2c_{1i}}{(|W| - 1)} + \sum_{i,j \in W \setminus 1} \frac{c_{ij}}{(|W| - 1)^2}. \end{aligned}$$

□

This corollary implies the following bound on the cost of the VRP.

Corollary 2.2.7. *Given a solution (K, v_k, d_i^k) of the VRP, its cost is at least*

$$\sum_{k=1}^K \sum_{i=2}^n 2c_{1i} \frac{d_i^k}{\sum_{t=2}^n d_t^k} + \sum_{k=1}^K \sum_{i,j \in \{2, \dots, n\}} c_{ij} \frac{d_i^k d_j^k}{(\sum_{t=2}^n d_t^k)^2}. \quad (2.3)$$

Proof. It is enough to prove that the cost of vehicle v_k is at least

$$\sum_{i=2}^n 2c_{1i} \frac{d_i^k}{\sum_{t=2}^n d_t^k} + \sum_{i,j \in \{2, \dots, n\}} c_{ij} \frac{d_i^k d_j^k}{(\sum_{t=2}^n d_t^k)^2}.$$

We can view each customer i as d_i^k customers with zero interdistance and unit demand. Thus, the routing of vehicle v_k can be viewed as a subtour W that passes through the depot and through $|W| - 1 = \sum_{t=2}^n d_t^k$ unit-demand customers. Expressing inequality (2.2) in terms of the original customers, we obtain that

$$\sum_{i=2}^n 2c_{1i} \frac{d_i^k}{\sum_{t=2}^n d_t^k} + \sum_{i,j \in \{2, \dots, n\}} c_{ij} \frac{d_i^k d_j^k}{(\sum_{t=2}^n d_t^k)^2}.$$

□

In the rest of the Section we develop a lower bound on $c(VRP)$ that can be computed efficiently for any Q . Moreover, it will be used on the following Sections to improve the approximation ratio of the algorithms by Altinkemer and Gavish when Q is constant.

We give some definitions first. The following definition is valid for the unit-demand VRP. It also applies to the SDVRP (CVRP) after we transform (relax) the instance into a unit-demand VRP. Alternatively, Definition 2.2.9 generalizes Definition 2.2.8 to the unequal-demand case without the need of transforming the instance into a unit-demand VRP.

Definition 2.2.8. *Given customer i , let $i(1), \dots, i(n-1)$ be the customers ordered by its proximity to i , that is, $i(1) = i$ and $c_{i,i(s)} \leq c_{i,i(s+1)}$ for all $1 \leq s \leq n-1$. Assuming $q_i = 1$ for each customer i , let $g(i) = \min_{1 \leq t \leq \min\{Q, n-1\}} \{2c_{1i} \frac{1}{t} + \sum_{1 \leq j \leq t} c_{i,i(j)} \frac{1}{t^2}\}$ and let $F(i) = \{i(1), i(2), \dots, i(s_i)\}$ where s_i is the argument integer number $1 \leq t \leq \min\{Q, n-1\}$ that minimizes $2c_{1i} \frac{1}{t} + \sum_{1 \leq j \leq t} c_{i,i(j)} \frac{1}{t^2}$.*

Informally, $g(i)$ tries to capture the radial cost and the interdistance cost associated to customer i , and the nodes of $F(i)$ are the customers that i would select to share a vehicle with in order to minimize $g(i)$. The sum of $g(i)$ for all customers i is a relaxation of the quadratic cost (2.3) that is at least the radial cost, as Lemma 2.2.11 will show. The cost $g(i)$ can be seen as a way of distributing a lower bound on the routing solution cost. There is a more natural way of distributing a lower bound on the vehicle routing cost to customers. (again we assume unit-demand VRP). Let $\mu(i)$ be

$$\mu(i) = \min \left\{ \frac{c(W)}{(|W| - 1)} : W \text{ is a subtour that contains } i \text{ and } 1, \text{ and } |W| \leq Q + 1 \right\}.$$

Then, $\sum_{i=2}^n \mu(i)$ is a lower bound on the cost of the unit-demand VRP. This bound also applies to SDVRP and CVRP after we transform (relax) the problem into a unit-demand VRP. This bound can be computed in polynomial time when Q is constant. Interestingly, the bounds $\sum_{i=2}^n \mu(i)$ and $\sum_{i=2}^n g(i)$ are not comparable. That is, there are instances where $\sum_{i=2}^n \mu(i)$ is greater than $\sum_{i=2}^n g(i)$, and vice versa.

The concept of associating to each customer i a radial cost plus a cost related to the average distance to its closest neighbors appeared in Dror and Ball [24]. When customers don't have unit demand, we can transform the problem to an equal demand problem and define $g(i)$ of an original customer as the sum of $g(j)$ of the unit demand customers that replace

it. Alternatively, we can generalize the definition of $g(i)$ in order to avoid the pseudopolynomial transformation to the unit demand case. For each customer i , let $i(1), \dots, i(n-1)$ be the customers ordered by its proximity to i , that is, $i(1) = i$ and $c_{i,i(s)} \leq c_{i,i(s+1)}$ for all $1 \leq s \leq n-1$. Let $T(i)$ be the integer such that $\sum_{l=1}^{T(i)-1} q_{i(l)} < Q$ and $\sum_{l=1}^{T(i)} q_{i(l)} \geq Q$. If $Q > \sum_{l=1}^n q_{i(l)}$ we define $T(i) = n$. For each $1 \leq t \leq T(i)$, let

$$\bar{q}_{i(t)} = \begin{cases} q_i & \text{if } t = 1, \\ q_{i(t)} & \text{if } 1 < t < T(i), \\ Q - \sum_{l=1}^{t-1} q_{i(l)} & \text{if } 1 < t = T(i). \end{cases}$$

Let $Q_i(t) = \min\{Q; \sum_{l=1}^t \bar{q}_{i(l)}\}$. The following definition generalizes $g(i)$ for the unequal-demand case.

Definition 2.2.9. *Given a customer i , let $i(1), \dots, i(n-1)$ be the customers ordered by its proximity to i , that is, $i(1) = i$ and $c_{i,i(s)} \leq c_{i,i(s+1)}$ for all $1 \leq s \leq n-1$. Let $g(i) = \min_{1 \leq t \leq n-1} \{2c_{1i} \frac{\bar{q}_{i(1)}}{Q_i(t)} + \sum_{1 \leq j \leq t} c_{i,i(j)} \frac{\bar{q}_{i(1)} \bar{q}_{i(j)}}{Q_i(t)^2}\}$ and let $F(i) = \{i(1), i(2), \dots, i(s_i)\}$ where s_i is the argument integer number $1 \leq t \leq T(i)$ that minimizes $\min_{1 \leq t \leq T(i)} \{2c_{1i} \frac{\bar{q}_{i(1)}}{Q_i(t)} + \sum_{1 \leq j \leq t} c_{i,i(j)} \frac{\bar{q}_{i(1)} \bar{q}_{i(j)}}{Q_i(t)^2}\}$.*

This definition is equivalent to the previous one after we transform the problem into a unit demand problem. More precisely, the following proposition holds. Its proof easily follows from Definitions 2.2.8 and 2.2.9.

Proposition 2.2.10. *Let I be an instance of a VRP. For each customer i , let $g(i)$ be defined as in Definition 2.2.9. We define an instance I' of a unit demand VRP by replacing each customer i with demand q_i by q_i customers i_1, \dots, i_{q_i} with unit demand and zero interdistance. For each customer i_t in the transformed problem, let $g(i_t)$ be defined as in Definition 2.2.8. Then, for each original customer i in I , $g(i) = \sum_{t=1}^{q_i} g(i_t)$.*

Sorting the customers with respect to the distance to i takes $O(n \log n)$ time. For each customer i , the computation of $g(i), F(i)$ can be done in $O(n)$ time. Therefore, the compu-

tation of $g(i), F(i)$ for all i takes $O(n^2 \log n)$ time. The term $\sum_{i \neq 1} g(i)$ is at least the radial cost and is at most the cost of the VRP as the next lemma shows.

Lemma 2.2.11.

$$R \leq \sum_{i=2}^n g(i) \leq c(\text{VRP}).$$

Proof. To simplify the proof we assume unit demand. It is clear from the definition of $g(i)$ that $c_{1i} \frac{1}{Q} \leq g(i)$, and therefore $\sum_{i \neq 1} 2c_{1i} \frac{1}{Q} = R \leq \sum_{i \neq 1} g(i)$.

Since we assume unit demand, each customer is visited by exactly one vehicle. In order to prove the inequality $\sum_{i=2}^n g(i) \leq c(\text{VRP})$, it is enough to prove that for each vehicle v , the routing cost of v is at least the sum of the cost $g(i)$ for every customer i visited by v . In other words, it is enough to prove that for every subtour W that visits at most Q customers, the inequality $c(W) \geq \sum_{i \in W \setminus 1} g(i)$ holds. By Corollary 2.2.6,

$$c(W) = \sum_{(i,j) \in W} c_{ij} \geq \sum_{i \in W \setminus 1} \frac{2c_{1i}}{(|W| - 1)} + \sum_{i,j \in W \setminus 1} \frac{c_{ij}}{(|W| - 1)^2}. \quad (2.4)$$

The following inequalities hold by definition of $g(i)$.

$$\frac{2c_{1i}}{(|W| - 1)} + \sum_{j \in W \setminus 1} \frac{c_{ij}}{(|W| - 1)^2} \geq \frac{2c_{1i}}{(|W| - 1)} + \sum_{1 \leq j \leq |W| - 1} \frac{c_{i,i(j)}}{(|W| - 1)^2} \geq g(i). \quad (2.5)$$

Combining equations (2.4) and (2.5) we prove that $c(W) \geq \sum_{i \in W \setminus 1} g(i)$ holds. \square

Lemma 2.2.11 shows that $\sum_{i \neq 1} g(i)$ is at least the radial cost. The next two examples give a sense of how big the difference between these two lower bounds can be. The first example shows a family of instances where $\sum_{i \neq 1} g(i) = R$, whereas the second example shows that $\sum_{i \neq 1} g(i)$ can be 50% better than the radial cost.

Example 2.2.12. For each $Q > 0$ we consider the following instance on the plane. Let the depot be the origin point $(0, 0)$ in the plane, and for each $1 \leq r \leq Q$ let customer i_r be located in the point $(\cos \frac{2\pi r}{Q}, \sin \frac{2\pi r}{Q})$. Each customer has demand Q . It is easy to see that $R = 2Q$ and that $g(i_r) = \frac{2Q}{Q}$. Therefore, $R = \sum g(i) = 2Q$ in this example.

Example 2.2.13. For each $Q > 0$ we consider the following instance on the real line. Let the depot be the origin point 0, for each $1 \leq r \leq Q - 1$ let customer i_r be also located in the origin point 0, and let customer i_Q be located in the point Q . Each customer has unit demand. In this case, the radial cost is equal to $\frac{2Q}{Q} = 2$. For each $1 \leq r \leq Q - 1$, $g(i_r) = 0$. It is easy to see that $g(i_Q) = \frac{2Q}{Q} + \frac{(Q-1)Q}{Q^2}$, and therefore $\sum g(i) \rightarrow 3 = 1.5R$ as $Q \rightarrow \infty$.

We summarize the result of this section with a lower bound of the cost of the VRP that is at least as good as the one derived in Lemma 2.2.4.

Theorem 2.2.14. The cost of the optimal solution of the VRP is at least

$$\max\left\{\sum_{i \neq 1} g(i); c(TSP)\right\}.$$

In the next sections we will work with the following auxiliary undirected graph.

Definition 2.2.15. For each customer i , let $F(i)$ be the set defined in Definition 2.2.9. Let $\bar{G} = (V - \{1\}, \bar{E})$ be the undirected graph formed by customers $\{2, \dots, n\}$ connected with edges $\bar{E} = \{(i, j) : i \in V - \{1\}, j \in F(i)\}$.

The set $F(i)$ is set of customers which customer i would like to share a vehicle with in order to minimize its cost $g(i)$. That is, the graph \bar{G} connects customers i, j whenever j belongs to $F(i)$ or i belongs to $F(j)$. We define the following quantities.

Definition 2.2.16. Let C_r be a connected component of \bar{G} . We denote by $q(C_r) = \sum_{i \in C_r} q_i$ its cumulative demand. We denote by $R(C_r) = \sum_{i \in C_r} \frac{2c_{1i}q_i}{\min\{Q; q(C_r)\}}$ its modified radial cost. We define $G(C_r) = \sum_{i \in C_r} g(i)$, and $H(C_r) = \sum_{i \in C_r} \sum_{j \in F(i)} \frac{c_{ij}}{\min\{Q^2; q(C_r)^2\}}$. Let C_1, \dots, C_m be the connected components of \bar{G} and let $F = \sum_{r=1}^m H(C_r)$.

These quantities satisfy the following properties, which are elementary and stated without proof.

Proposition 2.2.17. Let C_1, \dots, C_m be the connected components of \bar{G} . Let R be the radial cost. Then,

1. $R(C_r) + H(C_r) \leq G(C_r)$ for all $1 \leq r \leq m$.
2. $\sum_{r=1}^m G(C_r) = \sum_{i \neq 1} g(i)$.
3. Let $\beta > 0$ be a number such that $\sum_{i \neq 1} g(i) \leq (1 + \beta)R$. Then, $F = \sum_{r=1}^m H(C_r) \leq \beta R$.

2.3 Approximation Algorithms for SDVRP

2.3.1 Iterated Tour Partitioning (ITP) heuristic

We start by presenting the ITP heuristic by Altinkemer and Gavish [5] for the SDVRP. The original version runs in $O(Qn)$ time. Therefore, it runs in linear time when Q is fixed. However, it is a pseudopolynomial algorithm when Q is part of the input. We later improve the running time to $S(n, Q)$, which is the minimum time to sort n integers in the range $[1, Q]$. For example, the running time of a sorting algorithm like heapsort is $O(n \log n)$, so $S(n, Q) = O(n \log n)$. If $Q = O(n^p)$ for some fixed p , $S(n, Q)$ is $O(n)$ since radix sort runs in $O(n)$ time in this case (see the book by Cormen et al. [23] for a description of sorting algorithms).

The ITP heuristic receives a tour $1 - i_1 - i_2 - \dots - i_{n-1} - 1$ of cost at most $\alpha c(TSP)$ as part of the input and outputs a solution of the SDVRP within $1 + (1 - \frac{1}{Q})\alpha$ the optimal solution. On the original implementation of ITP, we replace each customer i with demand q_i by q_i customers with unit demand and zero interdistance. In the transformed graph the number of customers is $m = \sum_{i=2}^n q_i$. Let $1 - j_1 - \dots - j_m - 1$ be the tour on the transformed graph that results from replacing in the original tour $1 - i_1 - i_2 - \dots - i_{n-1} - 1$ each customer i by q_i consecutive customers with unit demand and zero interdistance. For each $1 \leq t \leq Q$, we define the solution route route_t on the transformed graph, as follows: route_t is the union of the subtours $v_1^t = 1 - j_1 - \dots - j_t - 1, v_2^t = 1 - j_{t+1} - \dots - j_{t+Q} - 1, v_3^t = 1 - j_{t+Q+1} - \dots - j_{t+2Q} - 1, \dots, v_{\lceil \frac{m-t}{Q} \rceil + 1}^t = 1 - j_{\lceil \frac{m-t}{Q} \rceil - 1} - \dots - j_m - 1$. That is, route_t transforms the tour $1 - j_1 - \dots - j_m - 1$ into subtours with Q customers each (except possibly the first and the last subtour). The solution route_{t+1} is different from

route_t in that the beginning and/or ending customers visited by each vehicle are shifted one position. With some abuse of notation, we also denote by route_t the induced solution in the original graph.

Denoting depot node 1 as $j_0 = j_{m+1}$, the sum of the cost of these solutions is

$$\begin{aligned} \sum_{t=1}^Q c(\text{route}_t) &= (Q+1)(c_{1j_1} + c_{1j_m}) + 2 \sum_{p=2}^{m-1} c_{1j_p} + (Q-1) \sum_{p=1}^{m-1} c_{j_p j_{p+1}} = \\ &= 2 \sum_{p=1}^m c_{1j_p} + (Q-1) \sum_{p=0}^m c_{j_p j_{p+1}} \leq \\ &= \sum_{i=2}^n 2c_{1i}q_i + (Q-1)\alpha c(TSP). \end{aligned}$$

The average cost of these solutions is at most $R + (1 - \frac{1}{Q})\alpha c(TSP)$. At least one of these solutions considered has cost at most the average. Since $\max\{R; c(TSP)\}$ is a lower bound on the optimal cost of the SDVRP, one of the solutions considered is within $1 + (1 - \frac{1}{Q})\alpha$ the optimal cost.

A straightforward implementation of ITP runs in $O(Qn)$ time. When the capacity Q is part of the input, this running time is pseudopolynomial. In what follows, we present an implementation of ITP that runs in polynomial time. The following lemma implies that we need to compute the cost of $O(n)$ solutions out of the Q solutions $\text{route}_1, \dots, \text{route}_Q$.

Lemma 2.3.1. *The set of costs of solutions $\{c(\text{route}_1), \dots, c(\text{route}_Q)\}$ has $O(n)$ elements.*

Proof. We observe that $v_k^{t+1} = 1 - j_{t+(k-2)Q+2}^-, \dots, -j_{t+(k-1)Q+2} - 1$, the routing of the k th vehicle in solution route_{t+1} , is the same as $v_k^t = 1 - j_{t+(k-2)Q+1}^-, \dots, -j_{t+(k-1)Q+1} - 1$, except for $j_{t+(k-2)Q+1}$ and $j_{t+(k-1)Q+2}$, the first and last customers visited by v_k^t and v_k^{t+1} respectively. However, if $j_{t+(k-2)Q+1}$ and $j_{t+(k-2)Q+2}$ represent the same original customer, and if $j_{t+(k-1)Q+1}$ and $j_{t+(k-1)Q+2}$ also represent the same original customer, the costs of the two vehicles are the same. In this case v_k^t and v_k^{t+1} visit the same original customers in the same order (what changes is the amount of demand delivered by these vehicles).

For each original customer i_p , we say that a number $1 \leq t \leq Q$ is a *starting number* of

customer i_p when the first vehicle that visits customer i_p in solution route_t does not visit the previous (original) customer i_{p-1} . That is, $1 \leq t \leq Q$ is a starting number of i_p if $\sum_{s=1}^{p-1} q_s - t$ is divisible by Q . It is easy to see that for each fixed customer i , there is exactly one starting number. Therefore, the number of different starting numbers is at most n .

We say that a number $1 = t \leq Q$ is a *breaking point* if either $t = 1$, or the costs of solutions $\text{route}_{t-1}, \text{route}_t$ are different. It is clear that the total number of different costs $c(\text{route}_1), \dots, c(\text{route}_Q)$ is at most the total number of breaking points. A necessary condition for $t \neq 1$ to be a breaking point, is that the cost of v_k^{t-1} is different from the cost of v_k^t for some k . This happens only if v_k^{t-1}, v_k^t have different starting or ending original customers. In other words, t is a breaking point only if either t or $t - 1$ is the starting number of some original customer i_p . This implies that the total number of breaking points is also $O(n)$. \square

The next lemma says that we can compute efficiently the best solution route_t .

Lemma 2.3.2. *The solution of the ITP heuristics can be computed in $O(S(n, Q))$ time, where $S(n, Q)$ is the time to sort n integers in the range $[1, Q]$.*

Proof. We use the same terminology as in the proof of Lemma 2.3.1. As it is proven in that lemma, it is enough to compute the costs of solutions route_t for t a breaking point. A necessary condition for $t \neq 1$ to be a breaking point is that, either t or $t - 1$ is a starting or ending number of some original customer i_p . Therefore, we compute the starting and ending number of every original customer i_p . Overall, this operation takes $O(n)$ time. We sort these numbers and we store them on a list L . This operation takes $O(S(n, Q))$ time. For each $t \in L$, let S_t be the set of original customers i_p with either t or $t - 1$ as starting number. Since each customer belongs to at most two sets S_t , the computation of the sets S_t takes $O(n)$ time. The sum $\sum_{t \in L} |S_t|$ is also $O(n)$. It is easy to see that the computation of $c(\text{route}_{t_{p+1}})$ can be done in $O(n)$ time. Given t_p, t_{p+1} two consecutive numbers in the sequence L , we claim that the cost of $\text{route}_{t_{p+1}}$ can be computed from the cost of route_{t_p} in $O(|S_{t_p}|)$. First, we can express $c(\text{route}_{t_{p+1}})$ as

$$c(\text{route}_{t_{p+1}}) = c(\text{route}_{t_p}) + \left(\sum_k c(v_k^{t_{p+1}}) - c(v_k^{t_p}) \right). \quad (2.6)$$

Second, the number of pairs of vehicles $v_k^{t_p}, v_k^{t_{p+1}}$ with different costs is $O(|S_{t_p}|)$. Finally, for each k , the difference $c(v_k^{t_{p+1}}) - c(v_k^{t_p})$ can be computed in $O(1)$ time since these routings differ in the first and the last customer they visit, at most. Therefore, we can compute the cost of route $_t$ for all t in the sequence L in $O(\sum_{t \in L} |S_t|) = O(n)$ time. \square

We summarize our presentation of the ITP heuristic with the following theorem.

Theorem 2.3.3. *Given a tour T of cost C_T as part of the input, the ITP heuristic outputs a solution for the SDVRP of cost at most*

$$R + (1 - \frac{1}{Q})C_T.$$

Given an α -optimal tour as part of the input, the ITP heuristic is a $1 + (1 - \frac{1}{Q})\alpha$ approximation algorithm for the SDVRP. The ITP heuristic runs in $O(S(n, Q))$ time.

2.3.2 Quadratic Iterated Tour Partitioning heuristics

The approximation ratio of the ITP heuristic for SDVRP is based on the lower bound of the optimal cost given in Lemma 2.2.4. In what follows we describe the Quadratic Iterated Tour Partitioning (QITP) heuristics. Its approximation ratio relies on the lower bound given in Theorem 2.2.14. It uses the ITP heuristics as a subroutine that receives a tour of cost C_T and outputs a valid solution for the SDVRP with cost at most $R + (1 - \frac{1}{Q})C_T$. Let β be a threshold value to be fixed afterwards. The QITP for the SDVRP described in this Section is as follows.

We divide the analysis of QITP into two cases; when $\sum_{i \neq 1} g(i) \geq (1 + \beta)R$, and when $\sum_{i \neq 1} g(i) < (1 + \beta)R$.

Let γ be

$$1 - \frac{\beta}{\beta + 1} + (1 - \frac{1}{Q})\alpha. \tag{2.7}$$

If $\sum_{i \neq 1} g(i) \geq (1 + \beta)R$, then the ITP heuristic improves its approximation ratio since the

<p>INPUT: Instance I of SDVRP, an α-optimal tour for I.</p> <p>OUTPUT: A solution of the SDVRP.</p> <ol style="list-style-type: none"> 1) Compute the radial cost R, and $F(i)$, $g(i)$ for all customer i; 2) If $\sum_{i \neq 1} g(i) \geq (1 + \beta)R$ then output solution by the ITP heuristic else begin 3) Compute graph \bar{G}; (see Definition 2.2.15) 4) Compute the connected components C_1, \dots, C_m of graph \bar{G}; 5) For each $1 \leq r \leq m$ Compute a subtour T_r for C_r; (see Lemma 2.3.4) 6) For each $1 \leq r \leq m$ such that $q(C_r) \geq Q + 1$ Use the ITP heuristic to transform the subtour T_r into a routing for C_r; 7) Output the solution generated in steps 5) and 6); <p>end</p>
--

Figure 2-1: Quadratic Iterated Tour Partitioning heuristic

lower bound of Theorem 2.2.14 is sharper than the one of Lemma 2.2.4. To be more precise, the ITP heuristic, which outputs a solution with cost at most $R + (1 - \frac{1}{Q})\alpha c(TSP)$, is now a γ -approximation algorithm since

$$\frac{R + (1 - \frac{1}{Q})\alpha c(TSP)}{\max\{\sum_{i \neq 1} g(i); c(TSP)\}} \leq \frac{R}{\sum_{i \neq 1} g(i)} + \frac{(1 - \frac{1}{Q})\alpha c(TSP)}{c(TSP)} \leq \gamma$$

If $\sum_{i \neq 1} g(i) < (1 + \beta)R$, let us consider the graph $\bar{G} = (V - \{1\}, \bar{E})$ formed by the customers connected with edges $\bar{E} = \{(i, j) : i \in V - \{\text{depot node } 1\}, j \in F(i)\}$ as defined in Definition 2.2.15. The following lemma shows how to construct subtours for each connected component of \bar{G} .

Lemma 2.3.4. *Let C_r be a connected component of \bar{G} with cumulative demand $q(C_r)$. Let $H(C_r)$ be defined as in Definition 2.2.16. There exists a subtour through the depot and all customers of C_r of cost at most*

$$\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} + 2(1 - \frac{1}{q(C_r)})Q^2 H(C_r). \quad (2.8)$$

Proof. To simplify notation we assume that each customer has unit demand. As mentioned before, we can replace each customer i with demand q_i by q_i customers with unit demand.

Let $s = q(C_r)$ be the number of customers of C_r . We relabel the customers of C_r as i_1, \dots, i_s following the appearance order in a depth first search in C_r . The arcs of the depth first search tree are of the form (i, j) with $i, j \in C_r$ and $j \in F(i)$ or $i \in F(j)$. Therefore the cost of the depth first search tree is at most $\sum_{i \in C_r, j \in F(i)} c_{i,j} \leq Q^2 H(C_r)$. If we duplicate all arcs of the depth first search tree we can construct an Eulerian subtour that visits customers i_1, \dots, i_s (in this order) of cost at most $2Q^2 H(C_r)$. By triangular inequality, the subtour $i_1 - \dots - i_s - i_1$ constructed from this subtour has cost at most $2Q^2 H(C_r)$. For $1 \leq t \leq s$, we consider the subtour $1 - i_t - i_{t+1} - \dots - i_{t+s} - 1$ (where indices are cyclic). The sum of the cost of these subtours is $2 \sum_{j=1}^s c_{1i_j} + (s-1) \sum_{j=1}^s c_{i_j i_{j+1}}$, where $i_{s+1} = i_1$. Therefore, the average cost is at most

$$2 \sum_{j=1}^s c_{1i_j} \frac{1}{s} + (1 - \frac{1}{s}) \sum_{j=1}^s c_{i_j i_{j+1}} \leq \sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} + 2(1 - \frac{1}{q(C_r)}) Q^2 H(C_r).$$

At least one of the $s = q(C_r)$ solutions considered has cost at most the average cost. \square

The following two lemmas show how to transform the subtour obtained in Lemma 2.3.4 into a valid set of routings.

Lemma 2.3.5. *Let C_r be a connected component of \bar{G} with cumulative demand $q(C_r)$ at most Q . There exists a routing that meets the demand of customers of C_r of cost at most $R(C_r) + 2(1 - \frac{1}{Q})Q^2 H(C_r)$.*

Proof. Since the cumulative demand of C_r is less than the capacity Q , the subtour obtained in Lemma 2.3.4 is a valid routing for C_r . The cost of this routing is at most $R(C_r) + 2(1 - \frac{1}{Q})Q^2 H(C_r)$ since $R(C_r) = \sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)}$, and $(1 - \frac{1}{q(C_r)})Q^2 H(C_r) \leq (1 - \frac{1}{Q})Q^2 H(C_r)$ when the cumulative demand $q(C_r)$ is at most Q . \square

Lemma 2.3.6. *Let C_r be a connected component of \bar{G} with cumulative demand $q(C_r)$ at least $Q + 1$. There exists a routing that meets the demand of customers of C_r of cost at most $(2 - \frac{2}{Q+1})R(C_r) + 2(1 - \frac{1}{Q})Q^2 H(C_r)$.*

Proof. By Lemma 2.3.4, there exists a subtour through the depot and customers of C_r of cost at most $\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} + 2(1 - \frac{1}{q(C_r)})Q^2H(C_r)$. Applying the ITP heuristics to this subtour, we construct a routing for the customers of C_r of cost at most

$$\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{Q} + (1 - \frac{1}{Q}) \left(\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} + 2(1 - \frac{1}{q(C_r)})Q^2H(C_r) \right). \quad (2.9)$$

Since $q(C_r) \geq Q + 1$, the inequality $\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} \leq (1 - \frac{1}{Q+1})R(C_r)$ holds. Therefore, the cost of the routing obtained from ITP is at most

$$(2 - \frac{2}{Q+1})R(C_r) + 2(1 - \frac{1}{Q})Q^2H(C_r).$$

□

Lemmas 2.3.5 and 2.3.6 imply that for each component C_r of \bar{G} there exists a routing that delivers the demand of all its customers (and does not visit any customer outside C_r) with cost at most

$$(2 - \frac{2}{Q+1})R(C_r) + 2(1 - \frac{1}{Q})Q^2H(C_r). \quad (2.10)$$

Therefore, the total cost of the routings for all connected components C_1, \dots, C_m is at most

$$(2 - \frac{2}{Q+1})R + 2(1 - \frac{1}{Q})Q^2F. \quad (2.11)$$

If $\sum_{i \neq 1} g(i) \leq (1 + \beta)R$ the inequality $F \leq \beta R$ holds (see Proposition 2.2.17). In this case, The ratio of this upper bound with the lower bound $\sum_i g(i)$ is at most

$$(2 - \frac{2}{Q+1}) + 2\beta(1 - \frac{1}{Q})Q^2. \quad (2.12)$$

To summarize, if $\sum_{i \neq 1} g(i) \geq (1 + \beta)R$ the algorithm outputs the solution given by ITP heuristics with approximation ratio given by equation (2.7). If $\sum_{i \neq 1} g(i) \leq (1 + \beta)R$ the algorithm constructs a solution with approximation ratio given by equation (2.12). Combining

Capacity Q	3	4	5	6
Approximation ratio of ITP(1) [5]	1.6667	1.7500	1.800	1.8333
Approximation ratio of QITP(1)	1.6540	1.7440	1.7968	1.8314

Table 2.1: Approximation ratio for SDVRP for small Q when $\alpha = 1$.

Capacity Q	3	4	5	6
Approximation ratio of ITP($\frac{3}{2}$) [5]	2.0000	2.1250	2.2000	2.2500
Approximation ratio of QITP($\frac{3}{2}$)	1.9629	2.1044	2.1872	2.2413

Table 2.2: Approximation ratio for SDVRP for small Q when $\alpha = \frac{3}{2}$.

both ratios, the algorithm has an approximation guaranteed ratio of

$$\max\left\{\left(2 - \frac{2}{Q+1}\right) + 2\beta\left(1 - \frac{1}{Q}\right)Q^2; 1 - \frac{\beta}{\beta+1} + \left(1 - \frac{1}{Q}\right)\alpha\right\}.$$

It remains to select β optimally. The term (2.7) is a decreasing function of β and the term (2.12) is an increasing function of β . When $\beta = 0$, the term (2.7) is smaller than the term (2.12) whereas when β is large enough the converse is true. Therefore, the value of β that gives the best ratio is the one that equalizes both terms, that is, the positive root of the polynomial

$$p(\beta) = (2Q^2 - 2Q)\beta^2 + \left(2 - \alpha + \frac{\alpha}{Q} - \frac{2}{Q+1} + 2Q^2 - 2Q\right)\beta + \left(1 - \alpha + \frac{\alpha}{Q} - \frac{2}{Q+1}\right).$$

Let $\beta^*(\alpha, Q)$ be the optimal β and let $a(\alpha, Q) = \frac{\beta^*(\alpha, Q)}{\beta^*(\alpha, Q)+1}$. Then, the algorithm has an approximation ratio of $1 - a(\alpha, Q) + \left(1 - \frac{1}{Q}\right)\alpha$.

Tables (2.1) and (2.2) depict the approximation ratio for small values of Q when $\alpha = 1$ and $\alpha = \frac{3}{2}$ respectively.

The following theorem summarizes the main result of this Section.

Theorem 2.3.7. *Given an α -optimal tour as part of the input, the Quadratic Iterated Tour Partitioning heuristics is a $1 - a(\alpha, Q) + \left(1 - \frac{1}{Q}\right)\alpha$ approximation algorithm for the SDVRP. The value $a(\alpha, Q)$ is at least $\frac{1}{3Q^3}$ for any $\alpha \geq 1$ and any $Q \geq 3$. For $\alpha = \frac{3}{2}$ the value of $a(\frac{3}{2}, Q)$ is at least $\frac{1}{4Q^2}$ for any $Q \geq 3$. The running time of this algorithm is $O(n^2 \log n)$.*

Proof. We start by showing that $a(\alpha, Q) \geq \frac{1}{3Q^3}$ for any $\alpha \geq 1$ and Q sufficiently large. Let us select a (suboptimal) value $\beta = \frac{1}{3Q^3-1}$. The algorithm has an approximation ratio which is the maximum between the bounds (2.7) and (2.12). With $\beta = \frac{1}{3Q^3-1}$, it is easy to see that the bound (2.7) is at most

$$2 - \frac{2}{Q+1} + 2\beta(1 - \frac{1}{Q})Q^2 \leq 2 - \frac{2}{Q+1} + \frac{2}{3Q}. \quad (2.13)$$

With $\beta = \frac{1}{3Q^3-1}$, bound (2.12) is at least

$$1 - \frac{\frac{1}{3Q^3-1}}{\frac{1}{3Q^3-1} + 1} + (1 - \frac{1}{Q})\alpha = 1 - \frac{1}{3Q^3} + (1 - \frac{1}{Q})\alpha \quad (2.14)$$

$$\geq 2 - \frac{1}{3Q^3} - \frac{1}{Q}. \quad (2.15)$$

It is easy to see that the lower bound (2.15) dominates the upper bound (2.13) for Q sufficiently large. More precisely, the lower bound (2.15) is bigger than the upper bound (2.13) when $Q \geq 6$. Therefore the approximation ratio of the algorithm is at most equation (2.14) for any $\alpha \geq 1$, any $Q \geq 6$ and thus $a(\alpha, Q) \geq \frac{1}{3Q^3}$ for any $\alpha \geq 1$, any $Q \geq 6$.

When $\alpha = \frac{3}{2}$, a similar argument shows that the value $a(\frac{3}{2}, Q)$ is at least $\frac{1}{4Q^2}$ for Q sufficiently large. In this case we select $\beta = \frac{1}{4Q^2-1}$. Then, bound (2.7) is at most $2 - \frac{2}{Q+1} + \frac{1}{2}$ and bound (2.12) is at least $\frac{5}{2} - \frac{3}{2Q} - \frac{1}{4Q^2}$. It is easy to see that bound (2.12) is at least bound (2.7) when $Q \geq 4$ and therefore $a(\frac{3}{2}, Q) \geq \frac{1}{4Q^2}$ for $Q \geq 4$.

It remains to show that $a(\alpha, Q) \geq \frac{1}{3Q^3}$ for any $\alpha \geq 1$, any $3 \leq Q \leq 6$ and that $a(\frac{3}{2}, Q) \geq \frac{1}{4Q^2}$ for $\alpha = \frac{3}{2}$, any $3 \leq Q \leq 6$. Table 2.2 shows the approximation ratio of QITP when β is selected optimally. It follows from Table 2.2 that $a(\frac{3}{2}, Q) \geq \frac{1}{4Q^2}$ for $3 \leq Q \leq 6$. In order to show that $a(\alpha, Q) \geq \frac{1}{3Q^3}$ for any $\alpha \geq 1$, any $3 \leq Q \leq 6$, we claim that the function $a(\alpha, Q) = \frac{\beta^*(\alpha, Q)}{\beta^*(\alpha, Q)+1}$ is nondecreasing in α . In order to prove this claim, is enough to prove that the optimal value $\beta^*(\alpha, Q)$ is a nondecreasing function of α . That is, $\beta^*(\alpha', Q) \geq \beta^*(\alpha, Q)$ when $\alpha \leq \alpha'$. To prove that $\beta^*(\alpha', Q) \geq \beta^*(\alpha, Q)$ when $\alpha \leq \alpha'$, we remind that $\beta^*(\alpha, Q)$ is defined so that the bounds (2.7) and (2.12) are equal. If we change α by a bigger value α'

(while we keep Q and β fixed), the term (2.12) increases whereas the term (2.7) does not change. In order to recover the equality between bounds (2.7) and (2.12), we must increase β since term (2.7) decreases and (2.12) increases when β increases. Therefore, the optimal parameter $\beta^*(\alpha', Q)$ must be greater than the optimal parameter $\beta^*(\alpha, Q)$ when $\alpha \leq \alpha'$.

It follows from Table 2.1 that $a(1, Q) \geq \frac{1}{3Q^3}$ for $3 \leq Q \leq 6$. Since the function $a(\alpha, Q) = \frac{\beta^*(\alpha, Q)}{\beta^*(\alpha, Q)+1}$ is nondecreasing in α , $a(\alpha, Q) \geq \frac{1}{3Q^3}$ for any $\alpha \geq 1$ and $3 \leq Q \leq 6$. This completes the proof of the approximation ratio of QITP.

With respect to the running time of the algorithm, computing $F(i)$ and $g(i)$ for all customer i takes $O(n^2 \log n)$. The ITP heuristic runs in $O(S(n, Q)) = O(n \log n)$ as we showed in Subsection 2.3.1. The auxiliary graph \bar{G} has n nodes and $O(n^2)$ edges. Computing \bar{G} and its connected components takes $O(n^2)$ time. Computing a subtour for each connected component using depth first search takes $O(n^2)$ time. Computing the heuristic solution in \bar{G} takes $O(S(n, Q)) = O(n \log n)$ since we run the ITP heuristics on each connected component of \bar{G} . Therefore, the total running time is $O(n^2 \log n)$. \square

We observe that Christofides' algorithm for the TSP has an approximation ratio of $\frac{3}{2}$ and runs in $O(n^3)$. Therefore, it dominates the running time of our algorithm.

When Q is fixed, the QITP heuristic uniformly improves upon the worst case analysis of the ITP heuristic. It is currently the best approximation algorithm for the SDVRP with triangle inequality, except when $Q = 4$. In this case, the algorithm MATCH^k by Anily and Bramel [6] has a better approximation ratio. When $Q = 4$, MATCH^k has a 1.75 approximation ratio.

2.4 Approximation Algorithm for CVRP

When split delivery of customer's demand is not allowed, the UITP heuristic by Altinkemer and Gavish [4] has an approximation guarantee ratio of $2 + (1 - \frac{2}{Q})\alpha$, assuming that Q is even. This is not restrictive since we can always scale Q and q_i to make Q even. We state the result of UITP heuristic.

Theorem 2.4.1. ([4]) Assume the capacity Q of vehicles is even. Given a tour T of cost C_T as part of the input, the UITP heuristic outputs a solution for the CVRP of cost at most

$$2R + (1 - \frac{2}{Q})C_T.$$

In particular, given an α -optimal tour as part of the input, the UITP heuristic is a $2 + (1 - \frac{2}{Q})\alpha$ approximation algorithm for the CVRP.

The UITP heuristics uses ITP heuristics as a subroutine. The call to ITP is the most expensive operation performed by UITP and therefore its running time is also $O(S(n, Q)) = O(n \log n)$. In this Section we present the Quadratic Unequal Iterated Tour Partitioning (QUITP), which is an adaptation of the algorithm from the previous Section to solve the CVRP using UITP as a subroutine. We will use UITP as a black box that receives a tour of cost C_T and outputs a valid routing for the CVRP with cost at most $2R + (1 - \frac{2}{Q})C_T$. For each customer i we compute $g(i), F(i)$, as defined in Definition 2.2.9.

Let β' be a threshold value to be fixed afterwards. The QUITP heuristics for the CVRP is as follows.

INPUT: Instance I of CVRP, an α -optimal tour for I .
 OUTPUT: A solution of the CVRP.

- 1) Compute the radial cost R , and $F(i), g(i)$ for all customer i ;
- 2) If $\sum_{i \neq 1} g(i) \geq (1 + \beta')R$ then output solution by the UITP heuristic
 else begin
- 3) Compute graph \bar{G} ; (see Definition 2.2.15)
- 4) Compute the connected components C_1, \dots, C_m of graph \bar{G} ;
- 5) For each $1 \leq r \leq m$
 Compute a subtour T_r for C_r ; (see Lemma 2.3.4)
- 6) For each $1 \leq r \leq m$ such that $q(C_r) \geq Q + 1$
 Use the UITP heuristic to transform the subtour T_r into a routing for C_r ;
- 7) Output the solution generated in steps 5) and 6);
- end

Figure 2-2: Quadratic Unequal Iterated Tour Partitioning heuristic

Let γ' be

$$2 - \frac{2\beta'}{\beta' + 1} + \left(1 - \frac{2}{Q}\right)\alpha. \quad (2.16)$$

If $\sum_{i \neq 1} g(i) \geq (1 + \beta')R$, then the UITP heuristic becomes a γ' -approximation algorithm because the lower bound of Theorem 2.2.14 is sharper than the one of Lemma 2.2.4.

If $\sum_{i \neq 1} g(i) \leq (1 + \beta')R$, we proceed in a similar way as in the previous Section, except that a solution constructed must not split demand. Let $\bar{G} = (V - \{1\}, \bar{E})$ be the undirected graph formed by the customers connected with edges $\bar{E} = \{(i, j) : i \in V - \{\text{depot node } 1\}, j \in F(i)\}$. Let C_1, \dots, C_m be its connected components. By Lemma 2.3.5, we can use one vehicle to deliver the demand of customers of a connected component C_r with cumulative demand $q(C_r) \leq Q$ with cost at most

$$R(C_r) + 2\left(1 - \frac{1}{Q}\right)Q^2H(C_r). \quad (2.17)$$

If a connected component C_r has cumulative demand $q(C_r) \geq Q + 1$, the following lemma holds.

Lemma 2.4.2. *Let C_r be a connected component of \bar{G} with cumulative demand $q(C_r)$ at least $Q + 1$. There exists a routing that meets the demand of customers of C_r without split of cost at most*

$$\left(3 - \frac{2}{Q} - \frac{1}{Q + 1} + \frac{2}{Q(Q + 1)}\right)R(C_r) + 2\left(1 - \frac{2}{Q}\right)Q^2H(C_r).$$

Proof. By Lemma 2.3.4 there exists a subtour through the depot and customers of C_r of cost at most $\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} + 2\left(1 - \frac{1}{q(C_r)}\right)Q^2H(C_r)$. Applying the UITP heuristics to this subtour (see Theorem 2.4.1) we construct a routing for the customers of C_r of cost at most

$$\sum_{i \in C_r} 4c_{1,i} \frac{q_i}{Q} + \left(1 - \frac{2}{Q}\right) \left(\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} + 2\left(1 - \frac{1}{q(C_r)}\right)Q^2H(C_r) \right). \quad (2.18)$$

Since $q(C_r) \geq Q + 1$, the inequality $\sum_{i \in C_r} 2c_{1,i} \frac{q_i}{q(C_r)} \leq (1 - \frac{1}{Q+1})R(C_r)$ holds. Therefore the cost of the routing obtained from UITP for C_r is at most

$$(3 - \frac{2}{Q} - \frac{1}{Q+1} + \frac{2}{Q(Q+1)})R(C_r) + 2(1 - \frac{2}{Q})Q^2H(C_r).$$

□

Given a connected component C_r , Lemmas 2.3.5 and 2.4.2 imply that we can construct a routing for all its customers with cost at most $(3 - \frac{2}{Q} - \frac{1}{Q+1} + \frac{2}{Q(Q+1)})R(C_r) + 2(1 - \frac{1}{Q})Q^2H(C_r)$. Therefore, the total cost of the routings for all connected components C_1, \dots, C_m of \bar{G} constructed following Lemmas 2.3.5 and 2.4.2 is at most

$$(3 - \frac{2}{Q} - \frac{1}{Q+1} + \frac{2}{Q(Q+1)})R + 2(1 - \frac{1}{Q})Q^2F. \quad (2.19)$$

If $\sum_{i \neq 1} g(i) \leq (1 + \beta')R$, the ratio of this upper bound with $\sum_{i \neq 1} g(i)$ is at most

$$3 - \frac{2}{Q} - \frac{1}{Q+1} + \frac{2}{Q(Q+1)} + 2(1 - \frac{1}{Q})Q^2\beta' \quad (2.20)$$

Combining equations (2.16) and (2.20), the algorithm has an approximation guaranteed ratio of

$$\max\{(3 - \frac{2}{Q} - \frac{1}{Q+1} + \frac{2}{Q(Q+1)}) + 2(1 - \frac{1}{Q})Q^2\beta'; 2 - \frac{2\beta'}{\beta'+1} + (1 - \frac{2}{Q})\alpha\}.$$

As in the previous Section, the optimal β' is the positive root of a quadratic function. Tables (2.3) and (2.4) depict the approximation ratio for small values of Q when $\alpha = 1$ and $\alpha = \frac{3}{2}$ respectively.

The following theorem has a similar proof as Theorem 2.3.7 of the previous Section.

Theorem 2.4.3. *Assume Q is even. Given an α -optimal tour, the Quadratic Unequal Iterated Tour Partitioning heuristics is a $2 - b(\alpha, Q) + (1 - \frac{2}{Q})\alpha$ approximation algorithm for the CVRP. The value $b(\alpha, Q)$ is at least $\frac{1}{3Q^3}$ for any $\alpha \geq 1$, any $Q \geq 4$. For $\alpha = \frac{3}{2}$ and*

Capacity Q	4	6	8	10
Approximation ratio of UITP(1) [4]	2.5000	2.6667	2.7500	2.8000
Approximation ratio of QUITP(1)	2.4923	2.6636	2.7485	2.7992

Table 2.3: Approximation ratio for CVRP for small Q when $\alpha = 1$.

Capacity Q	4	6	8	10
Approximation ratio of UITP($\frac{3}{2}$) [4]	2.7500	3.0000	3.1250	3.2000
Approximation ratio of QUITP($\frac{3}{2}$)	2.7234	2.9863	3.1170	3.1948

Table 2.4: Approximation ratio for CVRP for small Q when $\alpha = \frac{3}{2}$.

any $Q \geq 4$, the value $b(\frac{3}{2}, Q)$ is at least $\frac{1}{3Q^2}$. Its running time is $O(n^2 \log n)$.

2.5 Conclusions

We present a new quadratic lower bound on the cost of a solution of the VRP which improves the radial cost lower bound. We also present a relaxation of this lower bound that improves the bound by Haimovich and Rinnooy Kan [35] and can be computed in polynomial time. Based on this lower bound we develop the Quadratic Iterated Tour Partitioning and the Quadratic Unequal Iterated Tour Partitioning heuristics for the SDVRP and CVRP respectively that improve the approximation ratio of the algorithms by Altinkemer and Gavish when the capacity Q is fixed. The running time of the new algorithms is $O(n^2 \log n)$, which is not a bottleneck operation. To be more precise, we observe that all the mentioned algorithms for VRP receive an α -optimal traveling tour as part of the input and that the current best approximation algorithm for the TSP with triangle inequality runs in $O(n^3)$ time.

We show that an implementation of ITP for the SDVRP runs in $O(S(n, Q))$ where $O(S(n, Q))$ is the time to sort n integers in the range $[1, Q]$. The original implementation of ITP runs in $O(Qn)$ time and therefore it is pseudopolynomial when Q is part of the input.

With respect to open problems, we observe that the lower bound we use in our analysis (Lemma 2.2.11) is a relaxation of the bound given in Corollary 2.2.7. A more straightforward

quadratic relaxation of Corollary 2.2.7 may not be easily computable. It would be interesting to find a stronger relaxation of this bound that can also be computed in polynomial time since it could lead to better approximation algorithms.

Chapter 3

Probabilistic Analysis of Unit Demand Vehicle Routing Problems

3.1 Introduction

3.1.1 Unit Demand Vehicle Routing Problem

We study the Euclidean VRP where customers x_1, \dots, x_n and depot y_1 are points in the plane, and the distance between points is the Euclidean distance. Each customer has unit demand. The vehicles are identical, and with capacity $Q \in \mathbb{N}$. The route of each vehicle starts and ends at the depot y_1 . Each vehicle cannot deliver more than its capacity Q . The cost of a solution is the sum of the traversing cost of each vehicle. In the problems we consider the objective is to route vehicles to deliver the demand of every customer while minimizing the overall cost.

Except for some special cases, the VRP is an NP-hard problem. In their seminal paper [35], Haimovich and Rinnooy Kan provided a worst case and a probabilistic analyses of the VRP (see also [36]). In [35], a lower bound on the cost of VRP with metric distance is proved which is the maximum between the cost of a TSP and the so-called *radial cost*. When Q is fixed, the Euclidean VRP admits a polynomial time approximation scheme (PTAS). This means that for any $\epsilon > 0$ there exists a $1 + \epsilon$ -approximation algorithm. The first PTAS for

this case appeared in [35]. Subsequently, [9] improved its running time using the PTAS for the Euclidean TSP (see [7] or [46]).

In [35], the authors also analyzed the problem from a probabilistic point of view when the locations of customers are i.i.d. points and Q is a function of the number of customers. The analysis showed that the lower bound is asymptotically optimal when $Q = o(\sqrt{n})$ or $n^2 = o(Q)$. As a result, the Iterated Tour Partitioning heuristic (ITP) by Haimovich and Rinnooy Kan [35] becomes asymptotically optimal in both cases. For the rest of the cases, the ITP heuristic is within a factor of 2 of the optimal cost.

The primary contributions of this chapter are as follows:

1. We improve the approximation bound of Haimovich and Rinnooy Kan [35] for the VRP. This improvement, though slight, does resolve a long standing open question of whether any improvement was possible.

2. We provide nonlinear valid inequalities (Lemma 3.2.3) that are useful for improving bounds for VRP problems.

3. We show that for n points uniformly distributed in the unit square and for every p with $0 < p \leq 1$, there is a constant $c(p)$ such that

$$\lim_{n \rightarrow \infty} \text{P}(\text{at least } pn \text{ points have a neighbor at distance } \leq \frac{c(p)}{\sqrt{n}}) = 0.$$

4. We extend the probabilistic analysis of VRP to the multi-depot case.

To be more precise, we show that when customers are uniformly distributed points in the square $[0, 1]^2$ and the distance is the Euclidean distance, there exists a constant $\hat{c} > 0$ such that the lower bound we give is within a factor of $2 - \hat{c}$ of the optimal cost for any Q with probability arbitrarily close to 1 as the number of customers goes to infinity. In this way, the ITP heuristic is asymptotically a $2 - \hat{c}$ approximation algorithm for any Q . Our analysis also shows a further improvement on the approximation ratio when $Q = \Theta(\sqrt{n})$. These results are generalized in the second part of the chapter to the multi-depot case, where the location of the depots is fixed in advance. Related papers are Li and Simchi-Levi [44] and Stougie

[55]. We introduce a natural generalization of the Iterated Tour Partitioning heuristic (ITP) to the multi-depot case and we show that the results proved for the single-depot case carry through.

The rest of the chapter is organized as follows: in Section 3.2, we present lower bounds on the optimal cost of the VRP. In Section 3.3, we analyze the value of the lower bound in a probabilistic setting, and we prove our main results about the approximation ratio of ITP. We introduce the Multi-Depot Vehicle Routing Problem (MDVRP) in Section 3.4. In Section 3.5 we present an algorithm for the MDVRP that generalizes the ITP heuristic. In Section 3.6 we present lower bounds for the MDVRP. In Section 3.7 we analyze the lower bounds and the algorithm for MDVRP in a probabilistic setting. We summarize our conclusions in Section 3.8.

3.1.2 Notation

Unless otherwise stated, customers and depot are points in the plane. The location of i th customer is denoted by x_i for any $1 \leq i \leq n$. The depot is located at y_1 . The set of customers is denoted by $X^{(n)}$. The distance between customers i, j is denoted by c_{ij} or by $c_{x_i x_j}$, the distance between a customer i and depot y_1 is $c_{y_1, i}$. A solution of a VRP is denoted by (K, v_k, d_i^k) where K is the number of vehicles used, v_k denotes the routing of the k th vehicle and $d_i^k \in \{0, 1\}$ denotes whether the k th vehicle visits customer i or not. The routing cost of vehicle k is denoted by $c(v_k)$. $R = \sum_{i=1}^n 2c_{y_1, i}/Q$ is the so-called *radial cost*. We denote by $c(VRP)$ or by $c(VRP(X^{(n)}))$ the cost of an optimal VRP. We let $c(TSP)$ or $c(TSP(X^{(n)}))$ denote the cost of an optimal travelling salesman tour. Given a probability space and a probability measure, the probability that event A occurs is denoted by $P(A)$. The probability of event A conditioned on event B is $P(A|B)$. The complement of event A is \bar{A} . We use upper case letters (e.g., X) to denote random variables and lower case letters (e.g., x) to denote a realization of a random variable.

3.2 Lower bounds on the optimal cost of the VRP

We assume in this section that all distances satisfy the triangle inequality. We refer to such problems as metric. The Iterated Tour Partitioning heuristic $ITP(\alpha)$ for the unit demand VRP defined in [35] (see also [5]) receives an α -optimal TSP as part of the input and outputs a solution with cost at most $R + \alpha(1 - \frac{1}{Q})c(TSP)$. Lemma 2.2.4 also implies that the $ITP(\alpha)$ heuristic is a $1 + \alpha(1 - \frac{1}{Q})$ approximation algorithm for the unit demand VRP. In the Euclidean setting there exists a PTAS for the TSP (see [7], [46]). Therefore the ITP is a $2 - \frac{1}{Q}$ approximation algorithm in this case. We denote by $c(VRP^{ITP})$ the cost of the solution generated by the Iterated Tour Partitioning heuristic when it receives an optimal TSP as part of the input. When Q is not fixed, namely it is part of the input, the approximation ratio is asymptotically 2. We state this as a lemma.

Lemma 3.2.1. *The Iterated Tour Partitioning heuristic $ITP(1)$ is a 2-approximation algorithm.*

The following lemma is similar to Corollary 2.2.7. We introduce some notation first.

Definition 3.2.2. *Given a routing $v_k = y_1 - i_1^k - i_2^k - \dots - i_{s^k}^k - y_1$ that starts and ends at depot y_1 , we associate to it the sequence of customers $l^k = i_1^k - i_2^k - \dots - i_{s^k}^k$. The length of a sequence is the number of customers visited. For any customers i, j , let l_{ij}^k be the length of the path in l^k from i to j if both i and j are visited by v_k . Otherwise, $l_{ij}^k = 0$. Given a solution (K, v_k, d_i^k) of the VRP, its associated sequences l_1, \dots, l_K are the associated sequences of v_1, \dots, v_K .*

Since the distance matrix is symmetric, $l_{ij}^k = l_{ji}^k$.

Lemma 3.2.3. *Given a solution (K, v_k, d_i^k) of the metric VRP, its cost is at least*

$$\sum_{k=1}^K \sum_{i=1}^n 2c_{y_1, i} \frac{d_i^k}{\sum_{t=1}^n d_t^k} + \sum_{k=1}^K \sum_{i, j \in \{1, \dots, n\}} l_{ij}^k \frac{d_i^k d_j^k}{(\sum_{t=1}^n d_t^k)^2}. \quad (3.1)$$

Proof. To simplify notation, let us define $z_i^k = \frac{d_i^k}{\sum_{t=1}^n d_t^k}$.

Since $\sum_{j=1}^n z_j^k = 1$, we can rewrite (3.1) as

$$\begin{aligned}
& \sum_{k=1}^K \sum_{i=1}^n 2c_{y_1,i} z_i^k + \sum_{k=1}^K \sum_{i,j \in \{1,\dots,n\}} l_{ij}^k z_i^k z_j^k = \\
& \sum_{k=1}^K \sum_{i=1}^n 2c_{y_1,i} z_i^k \left(\sum_{j=1}^n z_j^k \right) + \sum_{k=1}^K \sum_{i,j \in \{1,\dots,n\}} l_{ij}^k z_i^k z_j^k = \\
& \sum_{k=1}^K \sum_{i,j \in \{1,\dots,n\}} 2c_{y_1,i} z_i^k z_j^k + \sum_{k=1}^K \sum_{i,j \in \{1,\dots,n\}} l_{ij}^k z_i^k z_j^k = \\
& \sum_{k=1}^K \left(\sum_{i,j \in \{1,\dots,n\}} (c_{y_1,i} + l_{ij}^k + c_{y_1,j}) z_i^k z_j^k \right) \leq \\
& \sum_{k=1}^K \sum_{i,j \in \{1,\dots,n\}} c(v_k) z_i^k z_j^k.
\end{aligned}$$

The last inequality holds because of the following reasoning: if $z_i^k z_j^k = 0$ then $(c_{y_1,i} + l_{ij}^k + c_{y_1,j}) z_i^k z_j^k = c(v_k) z_i^k z_j^k = 0$; otherwise v_k starts and ends at depot y_1 , and visits customers i, j and thus $(c_{y_1,i} + l_{ij}^k + c_{y_1,j}) z_i^k z_j^k \leq c(v_k) z_i^k z_j^k$ because of the triangle inequality. We use the equality $\sum_{j=1}^n z_j^k = 1$ again to simplify the last expression:

$$\begin{aligned}
& \sum_{k=1}^K \sum_{i,j \in \{1,\dots,n\}} c(v_k) z_i^k z_j^k = \\
& \sum_{k=1}^K c(v_k) \left(\sum_{i=1}^n z_i^k \right) \left(\sum_{j=1}^n z_j^k \right) = \sum_{k=1}^K c(v_k).
\end{aligned}$$

□

Since any vehicle v_k delivers $\sum_{t=1}^n d_t^k \leq Q$ units of demand, the following observation holds.

Observation 3.2.4. *Given a solution (K, v_k, d_i^k) of the metric VRP, the term $\sum_{k=1}^K \sum_{i=1}^n 2c_{y_1,i} \frac{d_i^k}{\sum_{t=1}^n d_t^k}$ is at least the radial cost $R = \sum_{i=1}^n 2c_{y_1,i} \frac{1}{Q}$.*

Definition 3.2.5. Given a solution (K, v_k, d_i^k) , a vehicle v_k is called half-full if it visits at least $\frac{Q}{2}$ customers. Let $A \subseteq V$ be the set of customers visited by half-full vehicles. A solution satisfies the fullness property if $|A| \geq n - \frac{Q}{2}$.

The following lemma says that there always exists an optimal solution that satisfies the fullness property.

Lemma 3.2.6. There exists an optimal solution (K, v_k, d_i^k) such that $|A| \geq n - \frac{Q}{2}$.

Proof. Let (K, v_k, d_i^k) be an optimal solution such that the associated set A has maximal cardinality. Either there is at most one vehicle that visits at most $\frac{Q}{2}$ of the customers or there are at least two. On the first case, $|A| \geq n - \frac{Q}{2}$. On the second case, we can replace two vehicles that visit at most $\frac{Q}{2}$ of the customers by one vehicle without increasing the routing cost. In this case we found an optimal solution with an associated set A' bigger than A , contradicting the maximality of A . \square

We give a name to the quadratic term on the expression (3.1).

Definition 3.2.7. Given a solution (K, v_k, d_i^k) , let

$$QC(K, v_k, d_i^k) = \sum_{k=1}^K \sum_{i,j \in \{1, \dots, n\}} l_{ij}^k \frac{d_i^k d_j^k}{(\sum_{j=1}^n d_j^k)^2}. \quad (3.2)$$

Let QC be the minimum value of $QC(K, v_k, d_i^k)$ among all optimal solutions (K, v_k, d_i^k) that satisfy the fullness property of **Definition 3.2.5**.

It is apparent that the radial cost plus the quadratic term QC is a lower bound on the cost of a solution (K, v_k, d_i^k) that satisfies the fullness property. Therefore, the following lemma holds.

Lemma 3.2.8. The cost of VRP is at least $\max\{R + QC; c(TSP)\}$.

3.3 Probabilistic analysis

We start giving the result by Beardwood, Halton and Hammersley [13] concerning the asymptotic behavior of the TSP. Let X_1, X_2, \dots be a sequence of i.i.d. points on $[0, 1]^2$. With probability one, the cost of an optimal subtour through the first n points satisfies that

$$\lim_{n \rightarrow \infty} \frac{c(TSP(X^{(n)}))}{\sqrt{n}} = \beta \int f^{1/2} dx = \beta \quad (3.3)$$

where f is the absolutely continuous density of the X_i and $\beta > 0$ is a constant that doesn't depend on the distribution.

From now on, we assume that the customers X_1, \dots, X_n are independent random variables with distribution $U[0, 1]^2$. Although the results proven in this chapter also hold for more general random variables, the restriction to uniform random variables is made in order to simplify proofs. In this case, the ratio between the cost of TSP and \sqrt{n} converges a.s. to a constant $\beta > 0$ (see [13]). The following theorem is proved in [35]. Informally, the result says that the radial cost dominates the TSP when the capacity $Q = o(\sqrt{n})$ and the reverse holds when $n^2 = o(Q)$.

Theorem 3.3.1. *Let X_1, X_2, \dots be a sequence of i.i.d. uniform random points in $[0, 1]^2$ with expected distance μ from the depot, and let $X^{(n)}$ denote the first n points of the sequence.*

- If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = 0$, then

$$\lim_{n \rightarrow \infty} \frac{c(VRP(X^{(n)}))Q}{n} = 2\mu \text{ a.s.}$$

- If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = \infty$, then

$$\lim_{n \rightarrow \infty} \frac{c(VRP(X^{(n)}))}{\sqrt{n}} = \beta \text{ a.s.}$$

where $\beta > 0$ is the constant of [13].

In particular, the ITP heuristic is asymptotically optimal whenever $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = 0$ or $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = \infty$.

Corollary 3.3.2. *If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = 0$ or if $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = \infty$ then $\frac{c(\text{VRP}^{\text{ITP}})}{c(\text{VRP})} = 1$ a.s.*

Informally, the quadratic term (3.2) captures part of the interdistance between customers that is neglected by the radial cost. This cost is related to the distance from a generic customer i to its closest neighbor $j \neq i$. Let p be a parameter. If we define a threshold value and we consider a customer an isolated customer whenever its distance to its closest neighbor is at least the threshold, then the following lemma says that we can define the threshold as a function of p in order to guarantee that the proportion of isolated customers is at least p .

Lemma 3.3.3. *For any $0 < p \leq 1$ there exists a value $c(p) > 0$ such that*

$$\lim_{n \rightarrow \infty} \text{P}(\text{at least } pn \text{ customers have a neighbor at distance } \leq \frac{c(p)}{\sqrt{n}}) = 0. \quad (3.4)$$

Proof. First divide the unit square into $K^2 = \alpha pn$ subsquares, each with a side whose length is $\frac{1}{\sqrt{\alpha pn}}$. We assume that αpn is the square of an integer. This assumption is used to simplify the proof. This proof carries through even if αpn is not a square by defining K as the (unique) integer such that $K^2 \leq \alpha pn < (K + 1)^2$. We will soon choose α as a function of p .

Suppose that n points are dropped at random, corresponding to the selection of n squares at random (with replacement). We will call a selected square *isolated* if no other selected square is a neighbor. (Neighbors include the square and its 8 adjacent squares.) Otherwise, we call it *non-isolated*. Then,

$$\begin{aligned} & \text{P}(\text{at least } pn \text{ customers have a neighbor at distance } \leq \frac{1}{\sqrt{\alpha pn}}) \leq \\ & \text{P}(\text{at least } pn \text{ selected squares have a selected neighbor}) = \\ & \text{P}(\text{at least } pn \text{ selected squares are non-isolated}). \end{aligned} \quad (3.5)$$

Suppose that we select points randomly one at a time. As the k -th point is randomly selected, the probability that it is adjacent to a point that already has been randomly selected is at most $\frac{9(k-1)}{\alpha pn}$, which is bounded above by $\frac{9n}{\alpha pn} = \frac{9}{\alpha p}$. This upper bound is independent of

the locations of the first $k - 1$ points. Therefore, the total number of non-isolated points is bounded by $2X$ where X is the number of successes in Bernoulli distribution where n events occur, and each event has a probability of $\frac{9}{\alpha p}$ of success. The factor of 2 comes from the fact that if point k is adjacent to point $i < k$, then i is also adjacent to k . Then,

$$\begin{aligned} & \text{P(at least } pn \text{ selected squares are non-isolated)} \leq \\ & \text{P(at least } \frac{pn}{2} \text{ successes out of } n \text{ with probability of success of } \frac{9}{\alpha p}). \end{aligned} \tag{3.6}$$

Therefore, if we choose α so that $\frac{18}{\alpha p} < p$ (that is, $\alpha > \frac{18}{p^2}$), then by the law of large numbers, $\lim_{n \rightarrow \infty} \text{P(at least } pn \text{ selected squares are non-isolated)} = 0$. Therefore, any value of $c(p)$ smaller than $\frac{\sqrt{p}}{3\sqrt{2}}$ satisfies equation (3.4). \square

This lemma provides non-trivial lower bounds on the cost of combinatorial optimization problems such as minimum weighted matching, travelling salesman problem and minimum latency problem as the next corollary shows. The minimum latency problem (see [15]) is to find a tour of minimum latency through all customers. The latency of a tour is the sum of waiting times of each customers. We observe that the lower bounds we obtain for minimum weighted matching and travelling salesman problem are of the same order of magnitude as the results from [49] and [13] for these problems respectively. We are not aware of an asymptotic result for the minimum latency problem.

Corollary 3.3.4. *The cost of minimum weighted matching and the cost of TSP when points are uniformly distributed in $[0, 1]^2$ are $\Omega(\sqrt{n})$ with probability 1. The cost of minimum latency problem when points are uniformly distributed in $[0, 1]^2$ is $\Omega(n^{1.5})$ with probability 1.*

Proof. For a fixed $0 < p < 1$ and for $c(p)$ such that equation (3.4) holds, minimum weighted matching has cost at least $\frac{c(p)}{2}(1 - p)n = \Omega(\sqrt{n})$ with probability 1. Selecting $p = 1/3$ and $c(p)$ smaller than $\frac{\sqrt{p}}{3\sqrt{2}}$ imply that minimum weighted matching has cost at least $0.04\sqrt{n}$ with probability 1. This bound also holds for the TSP too since minimum weighted matching is a lower bound of TSP.

Given a solution $y_1 - i_1 - i_2 - \dots, -i_n - y_1$ of the minimum latency problem, its cost is

$$\sum_{k=1}^{n-1} c_{i_k} = \sum_{k=0}^{n-1} (n-k) c_{i_k, i_{k+1}}.$$

For a fixed $0 < p < 1$ and for $c(p)$ such that equation (3.4) holds, this cost is at least

$$\sum_{k=(1-p)n}^{n-1} (n-k) c_{i_k, i_{k+1}} \geq \frac{c(p)}{\sqrt{n}} \frac{(1-p)^2 n^2}{2} = \Omega(n^{1.5}).$$

Selecting $p = 1/5$ and $c(p)$ smaller than $\frac{\sqrt{p}}{3\sqrt{2}}$ imply that minimum latency problem has cost at least $0.11n^{1.5}$ with probability 1. \square

Definition 3.3.5. Given a parameter c , we say that a customer i is non-isolated with respect to c if it has a neighbor at distance at most $\frac{c}{\sqrt{n}}$.

Proposition 3.3.6. Given a sequence $l = i_1 - i_2 - \dots - i_s$ with $s > 1$ customers where b of them are non-isolated w.r.t. c , the following inequality holds.

$$\sum_{i,j \in \{i_1, \dots, i_s\}} \frac{l_{ij}}{s^2} \geq \frac{c}{\sqrt{n}} \left(\frac{s}{6} - 2b \right).$$

Proof. We can express the sum $\sum_{i,j \in \{i_1, \dots, i_s\}} \frac{l_{ij}}{s^2}$ as $\sum_{t=1}^{s-1} \frac{2t(s-t)}{s^2} c_{i_t, i_{t+1}}$. Let $\delta(t) = 0$ whenever i_t and i_{t+1} are both non-isolated customers w.r.t. c and 1 otherwise. Then $c_{i_t, i_{t+1}} \geq \frac{\delta(t)c}{\sqrt{n}}$ and therefore

$$\begin{aligned} \sum_{i,j \in \{i_1, \dots, i_s\}} \frac{l_{ij}}{s^2} &= \sum_{t=1}^{s-1} \frac{2t(s-t)}{s^2} c_{i_t, i_{t+1}} \geq \\ \sum_{t=1}^{s-1} \frac{2t(s-t)}{s^2} \frac{\delta(t)c}{\sqrt{n}} &= \left(\sum_{t=1}^{s-1} \frac{2t(s-t)}{s^2} - \sum_{t:\delta(t)=0} \frac{2t(s-t)}{s^2} \right) \frac{c}{\sqrt{n}}. \end{aligned} \quad (3.7)$$

To bound the right hand side of this inequality we will use the identities $\sum_{t=1}^{s-1} t = \frac{s(s-1)}{2}$ and

$\sum_{t=1}^{s-1} t^2 = \frac{(s-1)s(2s-1)}{6} \leq \frac{s^2(s-1)}{3}$. The sum $\sum_{t=1}^{s-1} \frac{2t(s-t)}{s^2}$ is then

$$\sum_{t=1}^{s-1} \frac{2t}{s} - \sum_{t=1}^{s-1} \frac{2t^2}{s^2} \geq (s-1) - \frac{2}{3}(s-1) = \frac{1}{3}(s-1) \geq \frac{s}{6}. \quad (3.8)$$

In order to bound the sum $\sum_{t:\delta(t)=0} \frac{2t(s-t)}{s^2}$ we observe that among all possible distribution of b non-isolated customers in a sequence with s customers, this sum reaches its maximum value when the non-isolated customers are located in the center of the sequence l . The reason is that the weight $\frac{t(s-t)}{s^2}$ is a quadratic convex function of t that reaches its maximum at $t = s/2$. That is, this sum reaches its maximum when the non-isolated customers in the sequence l are $i_a, i_{a+1}, \dots, i_{i+b-1}$ where a is either $\lfloor \frac{s-b}{2} \rfloor$ or $\lfloor \frac{s-b}{2} \rfloor + 1$. (In particular, a is at most $\frac{s-b}{2} + 1$.) Therefore, the sum $\sum_{t:\delta(t)=0} \frac{2t(s-t)}{s^2}$ is at most

$$\sum_{t=a}^{a+b-1} \frac{2t(s-t)}{s^2} \leq \sum_{t=a}^{a+b-1} \frac{2ts}{s^2} = \frac{2b(2a+b-1)}{s} \leq \frac{b(s+1)}{s} \leq 2b. \quad (3.9)$$

Inequalities (3.8) and (3.9) imply that the right hand side of inequality (3.7) is at least $\frac{c}{\sqrt{n}}(\frac{s}{6} - 2b)$. \square

When $Q \geq 2$, the quadratic term QC defined in (3.2) is $\Omega(\sqrt{n})$ with probability 1 as the next lemma shows.

Lemma 3.3.7. *Assuming that $Q \geq 2$, there exists $\bar{c} > 0$ such that*

$$\lim_{n \rightarrow \infty} \mathbb{P}(QC \geq \bar{c}\sqrt{n}) = 1.$$

Proof. Let (K, v_k, d_i^k) be any optimal solution that satisfies the fullness property. By Lemma 3.2.6 there is always an optimal solution that satisfies the fullness property. This implies that at most one vehicle v_k of this solution visits only one customer. We will prove that there exists $\bar{c} > 0$ such that $\lim_{n \rightarrow \infty} \mathbb{P}(QC(K, v_k, d_i^k) \geq \bar{c}\sqrt{n}) = 1$ for any such (K, v_k, d_i^k) , which implies that $\lim_{n \rightarrow \infty} \mathbb{P}(QC \geq \bar{c}\sqrt{n}) = 1$ holds. Let $0 < p < 1, c > 0$ be parameters to be fixed afterwards. Each vehicle v_k of (K, v_k, d_i^k) has associated a sequence l_k with s_k

customers and b_k of them are non-isolated customers w.r.t. c . Proposition 3.3.6 implies that

$$\sum_{k=1}^K \sum_{i,j \in \{1, \dots, n\}} l_{ij}^k \frac{d_i^k d_j^k}{(\sum_{j=1}^n d_j^k)^2} \geq \sum_{k=1}^K \frac{c}{\sqrt{n}} \left(\frac{s_k}{6} - 2b_k \right) \geq \frac{c}{\sqrt{n}} \left(\frac{n}{6} - 2|\{\text{non isolated customers w.r.t. } c\}| \right). \quad (3.10)$$

If we choose p, c such that Lemma 3.4 holds, then

$$\lim_{n \rightarrow \infty} P(|\{\text{non-isolated customers w.r.t. } c\}| \leq pn) = 1$$

and therefore

$$\lim_{n \rightarrow \infty} P\left(\sum_{k=1}^K \sum_{i,j \in \{1, \dots, n\}} l_{ij}^k \frac{d_i^k d_j^k}{(\sum_{j=1}^n d_j^k)^2} \geq \frac{c}{\sqrt{n}} \left(\frac{n}{6} - 2pn \right) \right) = 1.$$

□

Theorem 3.3.8. *There exists a constant $\hat{c} > 0$ such that*

$$\lim_{n \rightarrow \infty} P\left(\frac{c(VRP^{ITP})}{c(VRP)} \leq 2 - \hat{c} \right) = 1.$$

Proof. We know that $c(VRP^{ITP}) \leq R + c(TSP)$. Lemma 3.2.8 says that $c(VRP) \geq \max\{R + QC, c(TSP)\}$. Therefore,

$$\frac{c(VRP^{ITP})}{c(VRP)} \leq \frac{R + c(TSP)}{\max\{R + QC, c(TSP)\}} \leq 2 - \frac{QC}{c(TSP)}. \quad (3.11)$$

We know that the ratio between the cost of TSP and \sqrt{n} converges a.s. to a constant β . By Lemma 3.3.7 we know that $\lim_{n \rightarrow \infty} P(QC \geq \bar{c}\sqrt{n}) = 1$. By setting $\hat{c} = \frac{\bar{c}}{\beta} < \frac{\sqrt{p}(\frac{1}{6} - 2p)}{3\sqrt{2}\beta}$ the following limit holds.

$$\lim_{n \rightarrow \infty} P\left(\frac{QC}{c(TSP)} \geq \hat{c} \right) = 1. \quad (3.12)$$

Equations (3.11) and (3.12) prove the theorem. □

What is \hat{c} ? In the uniform $[0, 1]^2$ setting, the constant $\beta > 0$ is at most $\sqrt{2}$ (see [30]). By fixing $p = \frac{1}{36}$, the value $\frac{\sqrt{p}(\frac{1}{6}-2p)}{3\sqrt{2}\beta}$ (and therefore \hat{c}) is at least 0.0028. A more careful derivation of bounds for constants $c(p)$ in Lemma 3.3.3 and for \hat{c} in Proposition 3.3.6 show that \hat{c} can be 0.01.

A generalization Let $\mu = \mathbb{E}(c_{y_1, X})$ be the expected distance of a customer X to the depot. If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}}$ exists and is equal to a finite value $w > 0$, the strong law of large numbers implies that $\lim_{n \rightarrow \infty} \frac{R}{\frac{2\mu}{w}\sqrt{n}} = 1$ a.s. The approximation ratio of the ITP heuristic satisfies the following.

Theorem 3.3.9. *Let \bar{c} be the constant defined on Lemma 3.3.7. Assume that $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}}$ exists and is equal to $0 < w < \infty$. The approximation ratio of the ITP heuristic satisfies that*

- If $\frac{2\mu}{w} + \bar{c} \leq \beta$,

$$\lim_{n \rightarrow \infty} \mathbb{P}\left(\frac{c(VRP^{ITP})}{c(VRP)} \leq 1 + \frac{\beta}{\frac{2\mu}{w} + \bar{c}} - \frac{\bar{c}}{\frac{2\mu}{w} + \bar{c}}\right) = 1.$$

- If $\frac{2\mu}{w} + \bar{c} < \beta$,

$$\lim_{n \rightarrow \infty} \mathbb{P}\left(\frac{c(VRP^{ITP})}{c(VRP)} \leq 1 + \frac{2\mu}{w\beta}\right) = 1.$$

Proof. The results follow from the ratio

$$\frac{c(VRP^{ITP})}{c(VRP)} \leq \frac{R + c(TSP)}{\max\{R + QC, c(TSP)\}}$$

and the limits $\lim_{n \rightarrow \infty} \frac{c(TSP)}{R+QR} = \frac{\beta}{\frac{2\mu}{w} + \bar{c}}$, $\lim_{n \rightarrow \infty} \frac{QC}{R+QR} = \frac{c}{\frac{2\mu}{w} + \bar{c}}$, $\lim_{n \rightarrow \infty} \frac{R}{c(TSP)} = \frac{2\mu}{w\beta}$. □

3.4 Multi-Depot Vehicle Routing Problem

In this Section we generalize the VRP to a multi-depot scenario. There are n customers and m depots. Each vehicle starts and ends at the same depot. We assume that there is no restriction on the number of vehicles available at each depot.

3.4.1 Notation

We extend the notation from the previous sections. Unless otherwise stated, customers and depots are points in the plane. The location of i th customer is denoted by x_i for any $1 \leq i \leq n$. The location of j th depot is denoted by y_j for any $1 \leq j \leq m$. The set of customers is denoted by $X^{(n)}$. The set of depots is denoted by $Y^{(m)}$. A solution of a multi-depot vehicle routing problem (MDVRP) is denoted by (K, v_k, d_i^k) where K is the number of vehicles used, v_k denotes the routing of the k th vehicle and $d_i^k \in \{0, 1\}$ denotes whether k th vehicle visits customer i or not. For each customer i , let $c_i = \min\{c_{i,y} : y \in Y^{(m)}\}$ be its minimum distance to a depot. $R_{MD} = \sum_{i=1}^n 2c_i/Q$ is the *multi-depot radial cost*. We denote by $c(MDVRP)$ or by $c(MDVRP(X^{(n)}))$ the cost of an optimal MDVRP and by $c(TSP)$ or by $c(TSP(X^{(n)} \cup Y^{(m)}))$ the cost of an optimal travelling salesman tour through all customers and depots. Let $c_\infty(MDVRP)$ denote the cost of the MDVRP when the capacity of vehicles is infinity. Given a vehicle v_k that starts and ends at depot y , and customers i, j , let $v_k - y$ be the path obtained by deleting the depot y from the tour v_k . Let l_{ij}^k be the length of the path in $v_k - y$ from i to j if both i and j are visited by v_k . Otherwise, $l_{ij}^k = 0$.

3.5 An algorithm for the Multi-Depot VRP

We generalize the Iterated Tour Partitioning to the multi-depot case. The Multi-Depot Iterated Tour Partitioning (MDITP(α)) heuristic we propose uses the ITP(α) heuristic as a subroutine. We assign each customer to its closest depot and solve m independent VRP problems. More formally, the MDITP heuristic is as follows:

- 0) $S_j = \emptyset$ for all $1 \leq j \leq m$;
- 1) For each customer x_i find y_j , its closest depot, and let $S_j = S_j \cup \{x_i\}$;
- 2) For each $1 \leq j \leq m$ run the ITP(α) heuristic to approximately solve the VRP problem VRP_j where y_j is the only depot and the set of customers is S_j ;

The cost of the solution produced by this heuristic is the sum of the cost of the solutions

produced by $ITP(\alpha)$ on each VRP_j which is at most $R_{MD} + \alpha \sum_{j=1}^m c(TSP_j)$. In what follows, we denote by $c(MDVRP^{MDITP})$ the cost of this heuristic when we use $ITP(1)$ as a subroutine. This analysis implies the following.

Lemma 3.5.1. *The cost of MDVRP is at most $R_{MD} + \sum_{j=1}^m c(TSP_j)$.*

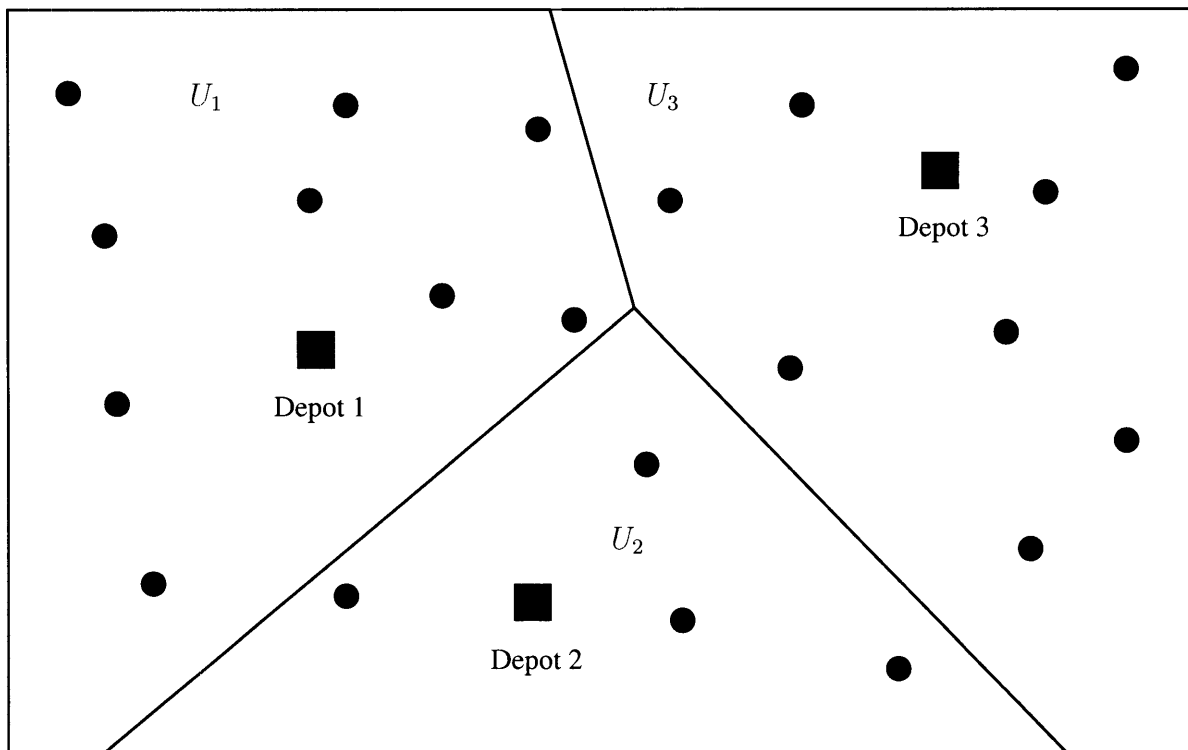


Figure 3-1: Assigning customers to their closest depot.

3.6 Lower bounds on the optimal cost of Multi-Depot VRP

In this Section we generalize the results from Section 3.2 to the multi-depot case. The following lemma generalizes Lemma 2.2.4.

Lemma 3.6.1. *The cost of MDVRP is at least*

$$\max\{R_{MD}; c_\infty(MDVRP)\}.$$

Proof. Let X_k be the set of customers visited by vehicle v_k and let y be the depot from where v_k starts and ends. Then,

$$c(v_k) \geq 2 \max_{i \in X_k} \{c_{i,y}\} \geq 2 \max_{i \in X_k} \{c_i\} \geq 2 \frac{\sum_{i \in X_k} c_i}{|X_k|} \geq \frac{2}{Q} \sum_{i \in X_k} c_i.$$

Summing for all vehicles we obtain that $c(MDVRP) \geq R_{MD}$.

Inequality $c(MDVRP) \geq c_\infty(MDVRP)$ holds since any feasible solution of the MDVRP with vehicles with capacity Q is also feasible for the infinite-capacity MDVRP. \square

The next lemma relates $c_\infty(MDVRP)$ with the cost of a TSP.

Lemma 3.6.2. *Let $c(TSP(Y^{(m)}))$ denote the cost of an optimal subtour through the depots. Then,*

$$c_\infty(MDVRP) \geq c(TSP(X^{(n)} \cup Y^{(m)})) - c(TSP(Y^{(m)})).$$

Proof. We observe that $c_\infty(MDVRP) + c(TSP(Y^{(m)}))$ is the cost of the walking tour formed by the union of the routes of an optimal solution of the infinite-capacity MDVRP plus a subtour through all the depots. This walking tour can be transformed into a TSP through all customers and depots of lesser cost by shortcutting nodes already visited. Therefore, $c_\infty(MDVRP) + c(TSP(Y^{(m)})) \geq c(TSP(X^{(n)} \cup Y^{(m)}))$ holds. \square

The following lemma is a generalization of Lemma 3.2.3.

Lemma 3.6.3. *Given a solution (K, v_k, d_i^k) of the MDVRP, its cost is at least*

$$\sum_{k=1}^K \sum_{i=1}^n 2c_i \frac{d_i^k}{\sum_{t=1}^n d_t^k} + \sum_{k=1}^K \sum_{i,j \in \{1, \dots, n\}} l_{ij}^k \frac{d_i^k d_j^k}{(\sum_{t=1}^n d_t^k)^2}. \quad (3.13)$$

A solution of the MDVRP satisfies the fullness property if $|A| \geq n - m\frac{Q}{2}$. The following lemma says that there always exists an optimal solution that satisfies the fullness property.

Lemma 3.6.4. *There exists an optimal solution (K, v_k, d_i^k) such that $|A| \geq n - m\frac{Q}{2}$.*

Proof. Given an optimal solution (K, v_k, d_i^k) , each depot has at most one vehicle that is not half-full. Otherwise, we can merge the routings of two not half full vehicles and obtain another optimal solution that satisfies the fullness property. \square

We give a name to the quadratic term on the expression (3.13).

Definition 3.6.5. *Given a solution (K, v_k, d_i^k) , let*

$$QC(K, v_k, d_i^k) = \sum_{k=1}^K \sum_{i,j \in \{2, \dots, n\}} l_{ij}^k \frac{d_i^k d_j^k}{(\sum_{j=2}^n d_j^k)^2}. \quad (3.14)$$

Let QC be the minimum value of $QC(K, v_k, d_i^k)$ among all optimal solutions (K, v_k, d_i^k) that satisfy the fullness property of **Definition 3.2.5**.

It is apparent that the multi-depot radial cost plus the quadratic term QC is a lower bound on the cost of any solution (K, v_k, d_i^k) that satisfies the fullness property. Since there is at least one optimal solution that satisfies the fullness property, the following lemma holds.

Lemma 3.6.6. *The cost of MDVRP is at least*

$$\max\{R_{MD} + QC; c(TSP(X^{(n)} \cup Y^{(m)})) - c(TSP(Y^{(m)}))\}.$$

Lemma 3.3.7 also holds for the multi-depot case. The constant \bar{c} of this lemma is the same as the one in Lemma 3.3.7.

Lemma 3.6.7. *Assuming that $\lim_{n \rightarrow \infty} Q = \infty$ and $Q = o(n)$, there exists $\bar{c} > 0$ such that*

$$\lim_{n \rightarrow \infty} P(QC \geq \bar{c}\sqrt{n}) = 1.$$

3.7 Probabilistic analysis of MDVRP

3.7.1 Probabilistic analysis of lower bounds

We analyze the Unit Demand Euclidean MDVRP where the depots and customers are points in the plane. The customers have unit demand. The m depots are fixed in advance whereas the location of the n customers are i.i.d. uniform random variables on $[0, 1]^2$.

Let U_1, \dots, U_m be a disjoint partition of the square $[0, 1]^2$ such that each U_j contains exactly one depot, namely y_j . For each $1 \leq j \leq m$, let $n_j \geq 0$ be the number of customers that belong to U_j and let $X^{(n_j)}$ be the set of customers that belong to U_j . Let $c(TSP_j) := c(TSP(X^{(n_j)} \cup \{y_j\}))$ be the cost of an optimal subtour that visits all customers of $X^{(n_j)}$ and depot y_j . The following result holds.

Lemma 3.7.1.

$$\lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(TSP_j)}{c(TSP)} = 1 \text{ (a.s.)}$$

Proof. For each $1 \leq j \leq m$, let f_j be the restriction of the uniform density to the set U_j . That is,

$$f_j(x) = \begin{cases} 1 & \text{if } x \in U_j, \\ 0 & \text{otherwise.} \end{cases}$$

The function $\frac{f_j}{|U_j|}$ is the density function of a customer conditional that it belongs to U_j . We will prove that

$$\lim_{n \rightarrow \infty} \frac{c(TSP_j)}{\sqrt{n}} = \beta \int f_j^{1/2} dx \text{ (a.s.)} \quad (3.15)$$

for each $1 \leq j \leq m$. Let us fix j . In order to prove equation (3.15) we would like to restrict the experiment to the customers that fell inside U_j and apply the result by Beardwood et al. [13]. However, this has to be done with some care since the number of customers that fell inside U_j is random. We observe that the distribution of customers that fell outside U_j does not affect $c(TSP_j)$. Therefore, the cost of the optimal TSP of the following experiment

is probabilistically the same random variable as $c(TSP_j)$. Let the n customers be i.i.d. points with the following distribution: with probability $p_j = |U_j|$ customer i is located in U_j according to density $\frac{f_j}{p_j}$, with probability $1 - p_j$ customer i is located in depot y_j . The main properties of the new experiment are

1. All the points fell inside U_j .
2. The cost of an optimal tour through all points and depot y_j in the new experiment has the same distribution as the cost $c(TSP_j)$ in the original experiment.

Since the absolutely continuous density part of the distribution of a customer in the new experiment is f_j , the result by Beardwood et al. [13] implies that equation (3.15) holds for each j .

Since for each j , equation (3.15) holds almost surely, altogether they imply that

$$\lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(TSP_j)}{\sqrt{n}} = \beta \int \sum_{j=1}^m f_j^{1/2} dx = \beta \int f^{1/2} dx \text{ (a.s.)}. \quad (3.16)$$

The last equality holds since $f = \sum_{j=1}^m f_j$ and the support of functions f_j are mutually disjoint. Finally, (3.16) and (3.3) imply that

$$\lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(TSP_j)}{c(TSP)} = \lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(TSP_j)/\sqrt{n}}{c(TSP)/\sqrt{n}} = 1 \text{ (a.s.)}$$

□

The following theorem generalizes Theorem 3.3.1 to the multi-depot case.

Theorem 3.7.2. *Let X_1, X_2, \dots be a sequence of i.i.d. uniform random points in $[0, 1]^2$ with expected distance μ to its closest depot, and let $X^{(n)}$ denote the first n points of the sequence.*

1. *If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = 0$, then*

$$\lim_{n \rightarrow \infty} \frac{c(MDVRP(X^{(n)}))Q}{n} = 2\mu \text{ a.s.}$$

2. If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = \infty$, then

$$\lim_{n \rightarrow \infty} \frac{c(\text{MDV}RP(X^{(n)}))}{\sqrt{n}} = \beta \text{ a.s.}$$

where $\beta > 0$ is the constant in [13].

Proof. Lemmas 3.5.1, 3.6.1, 3.6.2 imply that

$$\max\{R_{MD}; c(\text{TSP})(X^{(n)} \cup Y^{(m)}) - c(\text{TSP}(Y^{(m)}))\} \leq c(\text{MDV}RP(X^{(n)}))$$

and that

$$c(\text{MDV}RP(X^{(n)})) \leq R_{MD} + \sum_{j=1}^m c(\text{TSP}_j).$$

The ratio between R_{MD} and n/Q is equal to $\frac{R_{MD}Q}{n} = \sum_{i=1}^n \frac{2c_i Q}{nQ} = \sum_{i=1}^n \frac{2c_i}{n}$. The law of large numbers implies that this ratio converges a.s. to 2μ . If X_1, X_2, \dots are uniformly bounded and $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = 0$, then $\lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(\text{TSP}_j)Q}{n} = \lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(\text{TSP}_j)}{\sqrt{n}} \frac{Q}{\sqrt{n}} = 0$ since $\lim_{n \rightarrow \infty} \frac{\sum_{j=1}^m c(\text{TSP}_j)}{\sqrt{n}} = \lim_{n \rightarrow \infty} \frac{c(\text{TSP})(X^{(n)} \cup Y^{(m)})}{\sqrt{n}}$ is a constant. Therefore,

$$2\mu = \lim_{n \rightarrow \infty} \frac{\max\{R_{MD}; c(\text{TSP})(X^{(n)} \cup Y^{(m)}) - c(\text{TSP}(Y^{(m)}))\}}{n/Q} \leq$$

$$\lim_{n \rightarrow \infty} \frac{c(\text{MDV}RP(X^{(n)}))}{n/Q} \leq \lim_{n \rightarrow \infty} \frac{R_{MD} + \sum_{j=1}^m c(\text{TSP}_j)}{n/Q} = 2\mu$$

in this case.

When $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = \infty$, then $\frac{R_{MD}Q}{\sqrt{n}} = \sum_{i=1}^n \frac{2c_i}{n} \frac{n}{\sqrt{n}Q} = \sum_{i=1}^n \frac{2c_i}{n} \frac{\sqrt{n}}{Q}$. When n goes to infinity, this ratio converges to 0 a.s. The ratio between $c(\text{TSP})(X^{(n)} \cup Y^{(m)})$ and \sqrt{n} converges to β a.s. The ratio between $c(\text{TSP})(Y^{(m)})$ and \sqrt{n} converges to 0 since $c(\text{TSP})(Y^{(m)})$ is a constant. The ratio between $\sum_{j=1}^m c(\text{TSP}_j)$ and \sqrt{n} also converges to β a.s. because of Lemma 3.7.1. Therefore,

$$\beta = \lim_{n \rightarrow \infty} \frac{\max\{R_{MD}; c(\text{TSP})(X^{(n)} \cup Y^{(m)}) - c(\text{TSP}(Y^{(m)}))\}}{\sqrt{n}} \leq$$

$$\lim_{n \rightarrow \infty} \frac{c(\text{MDV}RP(X^{(n)}))}{\sqrt{n}} \leq \lim_{n \rightarrow \infty} \frac{R_{MD} + \sum_{j=1}^m c(\text{TSP}_j)}{\sqrt{n}} = \beta$$

in this case. □

3.7.2 Probabilistic analysis of an algorithm for MDVRP

The following theorem is an extension of Theorems 3.3.8 and 3.3.9.

Theorem 3.7.3. *There exists a constant $\hat{c} > 0$ such that*

$$\lim_{n \rightarrow \infty} \mathbb{P}\left(\frac{c(\text{MDV}RP^{\text{MDITP}})}{c(\text{MDV}RP)} \leq 2 - \hat{c}\right) = 1. \quad (3.17)$$

Moreover,

1. If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = 0$, then

$$\lim_{n \rightarrow \infty} \frac{c(\text{MDV}RP^{\text{MDITP}})}{c(\text{MDV}RP)} = 1 \text{ (a.s.)}$$

2. If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = w > 0$ and $\frac{2\mu}{w} + \bar{c} \leq \beta$,

$$\lim_{n \rightarrow \infty} \mathbb{P}\left(\frac{c(\text{MDV}RP^{\text{MDITP}})}{c(\text{MDV}RP)} \leq 1 + \frac{\beta}{\frac{2\mu}{w} + \bar{c}} - \frac{\bar{c}}{\frac{2\mu}{w} + \bar{c}}\right) = 1.$$

3. If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = w > 0$ and $\frac{2\mu}{w} + \bar{c} < \beta$,

$$\lim_{n \rightarrow \infty} \mathbb{P}\left(\frac{c(\text{MDV}RP^{\text{MDITP}})}{c(\text{MDV}RP)} \leq 1 + \frac{2\mu}{w\beta}\right) = 1.$$

4. If $\lim_{n \rightarrow \infty} \frac{Q}{\sqrt{n}} = \infty$, then

$$\lim_{n \rightarrow \infty} \frac{c(\text{MDV}RP^{\text{MDITP}})}{c(\text{MDV}RP)} = 1 \text{ (a.s.)}$$

where β is the constant from [13], μ is the expected distance of a customer to its closest depot and \bar{c} is the constant of Lemma 3.6.7.

Proof. The proofs of these results go along the same lines as the proofs of Theorems 3.3.8 and 3.3.9. The proofs of equation 3.17 and 2 follow from Lemmas 3.5.1, 3.6.6, 3.6.7 and 3.7.1. The proofs of 1, 3 and 4 follow from Lemma 3.5.1 and Theorem 3.7.2. \square

3.8 Conclusions

We present a lower bound for the metric VRP and show that is at least the radial cost plus a term of the same order of magnitude as the cost of TSP when the location of customers are i.i.d. uniform random points in the plane and the distance is the Euclidean distance. This lower bound improves on the previous work by [35]. We show that for the Euclidean VRP there exists a constant $\hat{c} > 0$ such that this lower bound is within a factor of $2 - \hat{c}$ of the optimal cost for any Q with probability arbitrarily close to 1 as the number of customers goes to infinity. This is an improvement over the result proved in [35], which is a 2 factor approximation. As a result, the ITP heuristic is asymptotically a $2 - \hat{c}$ approximation algorithm for any Q . In the second part of this chapter we analyze the multi-depot vehicle routing problem. We give a natural generalization of the ITP heuristic for this problem. The lower bounds presented on the first part are extended to this problem. The asymptotic results of the single-depot case from the first part are generalized to the multi-depot case.

Chapter 4

Using Grammars to Generate Very Large Scale Neighborhoods for Sequencing Problems

4.1 Introduction

Very large scale neighborhood search has been used in a variety of contexts and employs a variety of techniques. For survey papers, see Ahuja et al. [3] and Deĭneko and Woeginger [47]. Deĭneko and Woeginger [47] described a variety of techniques used to search exponential neighborhoods for the TSP in polynomial time. Other closely related papers include Gutin et al. [34], Ergun and Orlin [26], and Burkard et al. [17].

As a type of local search algorithm, the time to find a local optimum using VLSN search may not be polynomial. For example, Krentel [40] showed that the TSP under the k -Opt neighborhood is PLS-complete for some constant k , giving relative evidence that finding a local optimum is a hard task. For definition and properties of the class of problems PLS, see the book by Aarts and Lenstra [1]. On the bright side, Orlin et al. [48] shows that, as long as the neighborhood can be searched in polynomial time (e.g., VLSN), it is possible to find an ϵ -local optimum in polynomial time.

Many of the efficient search techniques for VLSN rely on dynamic programming recursions. In this chapter, we unify these disparate results into a unifying framework based on context free grammars. We also provide a generic dynamic programming algorithm for finding the best tour in a grammar-induced neighborhood. When specialized to the neighborhoods given in Deĭneko and Woeginger [47], our generic algorithm achieves the same running time as the special purpose dynamic programs, except for the twisted sequence neighborhood. In that case, our generic DP improves upon the previous best bound by a factor of n . The framework developed for generating neighborhoods for the TSP and the dynamic programming solver applies to other sequencing problems including the linear ordering problem.

For successful development of VLSN search, it will become increasingly important to find effective ways of describing the exponentially large neighborhood efficiently via a computer language. Here we have made substantial progress towards that goal in the case of sequencing problems.

1. We develop the first language for compactly generating exponentially large neighborhoods for sequencing problems. In fact, it is the first language for compactly generating exponentially large neighborhoods.
2. We develop the mathematical foundation for using regular grammars and context free grammars to describe very large neighborhoods for sequencing problems.
3. We develop a dynamic programming solver for the TSP that determines an optimum neighbor in time polynomial in the size of the problem and the number of rules. The solver uses the rules of the grammar as input as well as the TSP instance and current solution.
4. We develop dynamic programming solvers for a list of other sequencing problems, such as the Linear Ordering Problem, scheduling problems that determine an optimum neighbor in time polynomial in the size of the problem and the number of rules. The solver uses the rules of the grammar as part of its input.

5. We provide efficient algorithms for enumerating the size of neighborhoods and for deciding whether two neighborhoods are distinct in the case that the generating grammars are context-free and unambiguous.

The rest of the chapter is organized as follows. In Section 4.2, we describe a list of problems that can be expressed as sequencing problems. We establish the notation and terminology for sequencing problems, local search algorithms and grammars that we use throughout the chapter. In Section 4.3, we present neighborhoods defined in the literature and their description using grammars. In Section 4.4, we state the generic dynamic programming solvers for finding the best tour in a grammar-induced neighborhood, for each of the sequencing problems described in Section 4.2. We also present algorithms to solve other problems for grammar-based neighborhoods. We study more theoretical questions in Section 4.5. Papadimitriou and Steiglitz [50] proved that, unless $P = NP$, if we can search a neighborhood for the TSP in polynomial time, then this neighborhood cannot be *exact*, namely it must have local optima which are not global optima. We show a stronger result for a restricted case in our context: a sequence grammar that generates the complete neighborhood must have an exponential number of production rules.

We also present algorithms for counting the size of a neighborhood, and for deciding whether two sequence grammars generate the same language. In Section 4.6 we study problems related to the inverse of sequence grammars. We present an algorithm for optimizing over an inverse neighborhood. Finally, we present our conclusions in Section 4.7.

4.2 Notation and terminology

4.2.1 Sequencing Problems

We start by giving the general definitions associated to sequencing problems. In the next subsections, we give a list of particular sequencing problems, all of them being NP -hard problems.

Let $\{1, \dots, n\}$ be a set of *objects*. A *sequence* or *permutation* $\pi = (\pi(1), \dots, \pi(n))$ is

a bijective function $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$. A *subsequence* a_1, \dots, a_k is an injective function of $\{1, \dots, k\}$ onto $\{1, \dots, n\}$. The set of all permutations is denoted by S_n . Let $P_n \subseteq S_n$ be the set of *feasible solutions*. Let c be a cost function, where $c : P_n \rightarrow \mathbb{Z}$. The sequencing problem is to find a sequence $\pi \in P_n$ of minimum cost $c(\pi)$.

Traveling Salesman Problem

The Asymmetric Traveling Salesman Problem (ATSP) is to find the minimum distance tour on n cities $1, \dots, n$. The distance from city i to city j is c_{ij} , and we let C denote the matrix of distances. We represent a tour using a permutation $\pi \in S_n$, where S_n is the set of permutations of $\{1, \dots, n\}$. The permutation $\pi = (\pi(1), \dots, \pi(n))$ refers to the tour in which the first city visited is $\pi(1)$, the next city visited is $\pi(2)$, and so on. We will also refer to π as a *tour*. The cost of the tour π is denoted as $c(\pi) = \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}$. We will refer to two consecutive cities of π (that is, $\pi(i), \pi(i+1)$ for $1 \leq i \leq n-1$ and $\pi(n), \pi(1)$) as *edges* of the tour π . We note that n distinct permutations correspond to the same tour. Both symmetric and asymmetric versions of the Traveling Salesman Problem are *NP*-hard; a book devoted to different aspects of the traveling salesman problem is [42].

Linear Ordering Problem

We associate with each pair (i, j) a cost c_{ij} . The cost of a permutation $\pi = (\pi(1), \dots, \pi(n)) \in S_n$ is $c(\pi) = \sum_{i < j} c_{\pi(i)\pi(j)}$. The Linear Ordering Problem (LOP) is to find the permutation $\pi \in S_n$ with minimum cost. The Linear Ordering Problem is *NP*-hard (see Karp [38]).

Minimum Latency Problem

We associate with each pair (i, j) a cost c_{ij} . A permutation $\pi \in S_n$ represents a tour visiting all the customers. Each customer $\pi(i)$ experience a cost $c(\pi(i)) = \sum_{k=1}^{i-1} c_{\pi(k), \pi(k+1)}$. The cost of a permutation $\pi \in S_n$ of customers is the sum of costs that each customer experiences: $c(\pi) = \sum_{k=1}^{n-1} c(\pi(k)) = \sum_{k=1}^{n-1} (n-k)c_{\pi(k)\pi(k+1)}$. The Minimum Latency Problem (MLP) is to find a permutation $\pi \in S_n$ with minimum cost (see Blum et al. [15]).

Single Machine Scheduling Problem

Let $\{1, \dots, n\}$ be a set of jobs to be processed by a single machine. Each job i has a processing time p_i . Each job may also have a *due date* d_i and/or a *ready time* r_i . There may be precedence constraints among jobs, i.e., job i has to be processed before job j . Each schedule (sequence) of jobs has a cost. Given a sequence of jobs $\pi \in S_n$, the *completion time* of job $\pi(i)$ is $C_{\pi(i)} = \sum_{k=1}^i p_{\pi(k)}$. The *tardiness* of job $\pi(i)$ is $T_{\pi(i)} = \max\{C_{\pi(i)} - d_{\pi(i)}, 0\}$; the *lateness* of job $\pi(i)$ is $L_{\pi(i)} = C_{\pi(i)} - d_{\pi(i)}$. A single machine scheduling problem is to find a sequence of jobs $\pi \in S_n$ of minimum cost, where the cost function can be the sum of completion times $\sum C_{\pi(i)}$, the weighted sum of completion times $\sum w_{\pi(i)} C_{\pi(i)}$, the maximum tardiness $\max\{T_{\pi(i)}\}$, the maximum lateness $\max\{L_{\pi(i)}\}$, the makespan $C_{\pi(n)}$, or the sum of weighted tardinesses $\sum w_{\pi(i)} T_{\pi(i)}$.

All these single machine scheduling problems are *NP*–hard. See the survey chapter by Lawler et al. [43].

Weighted Bipartite Matching Problem with side constraints

Let $L = \{1, \dots, n\}$ be a set of lessons. For each $l \in L$, let b_l be the number of consecutive time slots lesson l needs. Let $T = \{t_1, \dots, t_m\}$ be the set of time slots, let $E \subseteq L \times T$ give the possible time slots for each lesson. That is, if (l, t) belongs to E then it is valid to teach l on the block of time slots that goes from t to $t + b_l$. A time slot can be used by at most one lesson. In the optimization version of the Bipartite Matching Problem with Relations, each pair $(l, t) \in E$ has a cost c_{lt} . We assume that invalid pairs $(l, t) \notin E$ have large cost. The Weighted Bipartite Matching Problem with side constraints (WBMPSC) is to find a feasible matching with minimum cost. If $b_l = 1$ for all lesson l then the problem is a (regular) matching and thus is solvable in polynomial time. However, even if $b_l \leq 2$ the problem is *NP*–hard (see ten Eikelder and Willems [56]).

Shortest Hyperpath Problem

We give some definitions first. These definitions are taken from Gallo et al. [31]. A *directed hyperpath* $\mathcal{H} = (\mathcal{V}, \mathcal{A})$ is a generalization of a directed graph in the following sense. \mathcal{V} is the set of nodes of \mathcal{H} , and \mathcal{A} is the set of *hyperarcs*. A hyperarc is a pair $e = (T(e), h(e))$, where $T(e) \subset \mathcal{V}$ is the *tail* of e , and $h(e) \in \mathcal{V} \setminus T(e)$ is the *head* of e . When $|T(e)| = 1$, the hyperarc is an arc in the normal sense of the word. Each hyperarc e has a cost c_e . A *path* from node s to node t is a sequence of alternating nodes and hyperarcs $(s = v_1, e_1, v_2, e_2, \dots, v_q = t)$ that starts at node s , ends at node t , such that each node v_i in the sequence is the head $h(e_i)$ of the previous hyperarc, and belongs to the tail $T(e_{i+1})$ of the next hyperarc. A *hyperpath* from node s to node t is a sub-hypergraph that contains a path from node s to node t , and it contains all the heads and tails of its hyperarcs, and it is minimal with respect to deletion of nodes and hyperarcs. The cost of a hyperpath is the sum of the costs of its hyperarcs.

Given a directed hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{A})$, nodes s, t and a cost function $c : \mathcal{A} \rightarrow \mathbb{R}$, the task is to find a hyperpath of minimum cost that contains a directed path from node s to node t .

4.2.2 Neighborhoods

We describe neighborhoods for sequencing problems with n objects. For sequence $\pi = (\pi(1), \dots, \pi(n))$, and permutation $\sigma = (\sigma(1), \dots, \sigma(n))$, we associate another sequence $\pi\sigma = (\pi(\sigma(1)), \dots, \pi(\sigma(n)))$, which is formed by composing permutations π and σ . For example, if $\pi = (3, 2, 4, 1, 5)$ and if $\sigma = (2, 1, 3, 5, 4)$ then $\pi\sigma = (2, 3, 4, 5, 1)$. We may view σ as an operator in this case since it transforms one sequence π into another sequence $\pi\sigma$.

We associate a set of operators N^π for each sequence π that induces the neighborhood $N^\pi(\pi) = \{\pi\sigma : \sigma \in N^\pi\}$. We refer to N^π as the *neighborhood set* for π .

In the case that $N^\pi = N^\gamma$ for all sequences π and γ , we say that the neighborhood is *sequence-invariant*. In the case that the neighborhood is sequence-invariant, the neighborhood of all sequences is entirely determined by the neighborhood set N . In this case, the neighborhood of π is denoted as $N(\pi)$. In this thesis, every neighborhood that we consider

will be sequence-invariant. This is a common assumption for neighborhoods for the traveling salesman problem, and was used implicitly by Deĭneko and Woeginger [47].

In the case that we are treating neighborhoods for different sized sequencing problems, let N_n denote the neighborhood set for problems with n objects. In the case that the number of objects is obvious from context, we drop the index, and denote the neighborhood set as N .

For example, in the case that $n = 5$, the 2-exchange neighborhood consists of all permutations that can be obtained from the sequence $(1, 2, 3, 4, 5)$ by flipping the order of one set of consecutively labeled objects. So, the following some elements of the 2-exchange neighborhood N_5 : $(1, 2, 3, 4, 5)$, $(1, 2, 5, 4, 3)$, $(1, 4, 3, 2, 5)$, $(1, 2, 4, 3, 5)$ and $(5, 4, 3, 2, 1)$.

In general, we assume that the identity permutation is in N_n , that is $(1, 2, \dots, n) \in N_n$.

We define the inverse of a neighborhood N , as the neighborhood $inv(N) = \{\sigma^{-1} : \sigma \in N\}$.

4.2.3 Very Large Scale Neighborhood Search

A neighborhood search algorithm starts with a feasible solution of the optimization problem and successively improves it by replacing it by an improved neighbor until it obtains a locally optimal solution. For many neighborhoods, an improving neighbor is determined by exhaustively enumerating all of the neighbors. We refer to a neighborhood as very large scale if the neighborhood is too large to be searched exhaustively and is searched using some more efficient search procedure.

In this thesis, we are primarily (but not entirely) focused on neighborhoods with the following two properties, also given in Deĭneko and Woeginger [47].

1. *Exponential Size Property.* The neighborhood set N_n is exponentially large in n . That is, there is **no** polynomial function $f(n)$ such that $|N_n| = O(f(n))$.
2. *Polynomially Searchable Property.* The neighborhood N_n may be searched in polynomial time. That is, there is Algorithm A and a polynomial $f()$ such that for every

sequence $\pi \in S_n$, Algorithm A can find the neighbor of minimum cost in $N_n(\pi)$ with respect to any cost function in time $O(f(n))$.

Any neighborhood with the first property is referred to as an *exponential neighborhood*. Any neighborhood with the second property is said to be *searchable in polynomial time*.

Very large scale neighborhood search has been used for a wide type of problems, and uses different search techniques. For survey papers, see Ahuja et al. [3] and Deĭneko and Woeginger [47]. Deĭneko and Woeginger [47] described a variety of techniques used to search exponential neighborhoods for the TSP in polynomial time. Many of these relied on dynamic programming recursions. In this chapter, we unify the disparate results of those solvable by dynamic programming into a unifying framework based on context-free grammars. Moreover, we provide a generic dynamic programming algorithm for finding the best tour in a grammar-induced neighborhood that has the same running time for these special cases as those described by Deĭneko and Woeginger [47]. It is worth mentioning that it solves the twisted neighborhood faster than the algorithm given in [47].

4.2.4 Grammar terminology

Roughly speaking, a grammar is a set of rules for how to compose strings in a language. In this subsection we define the concepts from the theory of languages that are relevant in our work. We refer to the books by Hopcroft and Ullman [37] and Sipser [54] for more information on languages and grammars. The primitive symbols are either *terminals* or *non-terminals*. V_{NT} is the set of variables or non-terminals, and V_T is the set of terminals. We assume that V_{NT} and V_T are disjoint. We let $V = V_{NT} \cup V_T$. $S \in V_{NT}$ is the *start* symbol. A *string* $\alpha \in V^*$ is a finite sequence of terminals and non-terminals. In what follows, lower-case letters a, b, c denote terminals. Capital letters A, B, C denote non-terminals. Capital letters R, X, Z denote either terminals or non-terminals. Greek letters α, β denote strings of terminals and non-terminals.

P is the set of production rules. We first consider production rules $p \in P$ of the following

form:

$$A \rightarrow a_1 \dots a_k \text{ for some } k \geq 1, \quad (4.1)$$

$$A \rightarrow a_1 \dots a_k B_1 \text{ for some } k \geq 1, \text{ or} \quad (4.2)$$

$$A \rightarrow B_1 a_1 \dots a_k \text{ for some } k \geq 1, \text{ or} \quad (4.3)$$

with $A, B_1 \in V_{NT}$, $a_1, \dots, a_k \in V_T$. We denote by $G = (V_{NT}, V_T, P, S)$ a *grammar*. A grammar with production rules of the form 4.1 and 4.2 (4.1 and 4.3) is a *right (left) regular grammar*.

Example 4.2.1 ([37]). Let $G = (V_{NT}, V_T, P, S)$ be the grammar where $V_{NT} = \{S, B\}$, $V_T = \{a, b\}$ and $P = \{S \rightarrow aB, S \rightarrow ab, B \rightarrow bS\}$. Then G is a right regular grammar and the language it generates is $L(G) = \{ab, abab, ababab, \dots\} = \{(ab)^n : n \geq 1\}$.

A more general type of grammars are the *context-free* grammars. In a context-free grammar we allow production rules of the form (4.1) to (4.3) and

$$A \rightarrow B_1 B_2 \dots B_k \text{ for some } k \geq 1, \quad (4.4)$$

with $A, B_1, \dots, B_k \in V_{NT}$. Sometimes we write a production rule like $A \rightarrow B_1 B_2 \dots B_k$ as $A \rightarrow B_1, B_2, \dots, B_k$. The commas are used to separate symbols more clearly and they do not have any additional interpretation.

The following example from [37] shows a context-free grammar and the language it generates. It is worth mentioning that the so-called pumping lemma, a theoretical result from grammar theory, implies that there is no left-regular grammar that generates the same language.

Example 4.2.2 ([37]). Let $G = (V_{NT}, V_T, P, S)$ be the grammar where $V_{NT} = \{S, B\}$, $V_T = \{a, b\}$ and $P = \{S \rightarrow aB, S \rightarrow ab, B \rightarrow Sb\}$. Then G is a context-free grammar and the language it generates is $L(G) = \{ab, aabb, aaabbb, \dots\} = \{a^n b^n : n \geq 1\}$.

A context-free grammar with at most two nonterminals on the right hand side is in

normal form. A grammar with production rules of the form 1, 2, 3 is in *extended normal form*.

Finally, in a *context-sensitive* grammar we allow rules of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ for $\alpha_1, \alpha_2, \beta \in V^*$. A regular grammar is also a context-free grammar, and a context-free grammar is also a context-sensitive grammar. All these inclusions are strict (see Hopcroft and Ullman [37]). That is, there are languages generated by a context-sensitive grammar that cannot be generated by a context-free grammar. And there are languages generated by a context-free grammar that cannot be generated by a regular grammar. In this chapter, we will define *sequencing* grammars, which are a subset of regular grammars. In most of this thesis we will not use context-sensitive grammars since they are too powerful (see Section 4.3.5).

For a production rule $A \rightarrow \beta \in P$, we say that the string $\alpha A \gamma$ *directly derives* $\alpha \beta \gamma$ in grammar G for all strings $\alpha, \gamma \in V^*$. String $\alpha \in V^*$ *derives* string $\beta \in V^*$ in G if there exist strings $\alpha_0, \dots, \alpha_k \in V^*$ for some $k \geq 1$ such that $\alpha_0 = \alpha, \alpha_k = \beta$ and α_i directly derives α_{i+1} in G for $0 \leq i \leq k - 1$. The *language generated* by G , denoted $L(G)$, is the set of strings, consisting solely of terminals, that can be derived from S .

For each non-terminal A we denote by $L(G_A)$ the language generated by the grammar $G_A = (V_{NT}, V_T, P, A)$, that is, using non-terminal A as the start symbol.

A tree is a *derivation* (or *parse*) *tree* for G if

1. every vertex of the tree has a label which is a terminal or a non-terminal,
2. the label of the root of the tree is the start non-terminal S ,
3. all the interior vertices of the trees have non-terminals as labels,
4. if vertex i has label A and its sons are vertices i_1, \dots, i_r from left to right with labels R_1, \dots, R_r , then $A \rightarrow R_1, \dots, R_r$ must be a production rule in P .

The set of leaves of a derivation tree, ordered from left to right, form a string in V^* . We say that the derivation tree *generates* this string. A grammar is *ambiguous* if there exist two parse trees that generate the same string. It is unambiguous otherwise.

Given a production rule $p : A \rightarrow R_1 \dots R_r$ we denote by $L(G_p)$ the set of strings α generated by the grammar $G_A = (V_{NT}, V_T, P, A)$ such that p belongs to the parse tree of α .

4.2.5 Sequence Grammar

Let $G = (V_{NT}, V_T, P, S)$ be a regular or context-free grammar that generates a finite language. We refer to G as a *sequence grammar on n objects* if (i) the set of terminals contains n symbols (i.e., $V_T = \{a_1, \dots, a_n\}$), (ii) each nonterminal on the right-hand side of a production rule appears on the left-hand side of some production rule, and (iii) there exists a function $Objects : V_{NT} \cup V_T \rightarrow 2^{\{1, \dots, n\}}$ that assigns a subset of objects to each element of V with the following properties:

1. $Objects(S) = \{1, \dots, n\}$,
2. $Objects(A) = \cup_{j=1}^k Objects(R_j)$ for each production rule $A \rightarrow R_1 R_2 \dots R_k$,
3. $Objects(R_i) \cap Objects(R_j) = \emptyset$ for each production rule $A \rightarrow R_1 R_2 \dots R_k$, for $1 \leq i < j \leq k$,
4. $Objects(a_j) = \{j\}$ for every terminal $a_j \in V_T$.

We extend the definition of $Objects$ to strings generated by G . We observe that $Objects(\alpha)$ is uniquely defined for any string α . A unique subsequence (i_1, i_2, \dots, i_r) is associated to each string of terminals $\alpha = a_1 \dots a_r$ generated by G . The next proposition shows that a Sequence Grammar of n objects generates a subset of permutations of the n objects.

Proposition 4.2.3. *A Sequence Grammar generates a non-empty language. Every element of the language is a permutation on n objects.*

Proof. Since every nonterminal that appears on a RHS of a production rule also appears on the LHS of another production rule, a derivation cannot end on a string with non-terminals. Thus, a sequence grammar generates a non-empty language.

Property 2 of function $Objects$ implies that the sets of objects associated to the RHS and to the LHS of a production rule are the same. That is, all the objects on the RHS are also on

the LHS and viceversa. Property 3 implies that an object does not belong to more than one R in the RHS of a production rule. Therefore any string α generated by the grammar must be a subsequence through the objects of S , the start symbol. Since $Objects(S) = \{1, \dots, n\}$, a string α generated by the grammar is a sequence on n objects. \square

The neighborhood set N generated by G is the language $L(G)$. Following Subsection 4.2.2, the neighborhood set of sequence π generated by G , is the set $N(\pi) = \{\pi\alpha : \alpha \in L(G)\}$. Alternatively, $N(\pi)$ is the language generated by the sequence grammar $(G, Objects_\pi)$ where $Objects_\pi(R) = \pi(Objects(R))$ for all $R \in V$.

4.3 Neighborhoods Generated by Sequence Grammars

In this section, we describe neighborhoods for sequencing problems that can be defined using sequence grammars. Most of these neighborhoods were initially defined for the TSP. However, they are also valid neighborhoods for the list of sequencing problems given in Subsection 4.2.1. The running time to optimize over a neighborhood depends on the specific sequencing problem.

4.3.1 Polynomial Neighborhoods

Each neighborhood with K neighbors can be generated by a regular grammar with K production rules.

The following sequence grammar defines the neighborhood in which adjacent objects may be swapped. The set of terminals and non-terminals are $V_{NT} = \{S\}$, $V_T = \{1, \dots, n\}$. Its production rules are

The Adjacent Interchange Neighborhood Grammar

$$S \rightarrow 1, \dots, n,$$

$$S \rightarrow 2, 1, 3, \dots, n,$$

$$S \rightarrow 1, \dots, i, i+2, i+1, i+3, \dots, n \text{ for } 1 \leq i \leq n-3,$$

$$S \rightarrow 1, \dots, n-2, n, n-1,$$

$$S \rightarrow 2, \dots, n-1, 1, n.$$

The value of the function *Objects* on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

The following traveling salesman sequence grammar is the 2-exchange neighborhood. The set of terminals and non-terminals are $V_{NT} = \{S\}$, $V_T = \{1, \dots, n\}$. Its production rules are

The 2-exchange Neighborhood Grammar

$$S \rightarrow 1, \dots, n,$$

$$S \rightarrow 1, \dots, i, j, j-1, \dots, i+2, i+1, j+1, \dots, n \text{ for } 1 \leq i \leq j-2 \leq n-3,$$

$$S \rightarrow 1, \dots, i, n, n-1, \dots, i+2, i+1 \text{ for } 1 \leq i \leq n-3,$$

$$S \rightarrow i, i-1, \dots, 1, i+1, \dots, n \text{ for } 2 \leq i \leq n-1.$$

The value of the function *Objects* on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

In both simple grammars, the only non-terminal corresponded to the set $\{1, \dots, n\}$. Such would normally be the case for polynomially sized neighborhoods that are used in local search algorithms for sequencing problems.

4.3.2 Exponential Neighborhoods

We describe a number of Exponential Neighborhoods for sequencing problems. Most of them were originally described for the TSP and are covered in the survey paper by Deĭneko and

Woeginger [47]. Their common feature is that they can be optimized efficiently using a Dynamic Programming approach. These neighborhoods are valid for any sequencing problem described in Section 4.2.1. However, the running time for optimizing over a particular sequence grammar neighborhood is problem-specific.

Pyramidal neighborhood

The pyramidal neighborhood consists of *pyramidal permutations*, where the object labels increase monotonically to n and then decrease. That is,

$$N_P = \{\sigma : \sigma(1) < \sigma(2) < \dots < \sigma(k) = n, \quad \sigma(k+1) > \sigma(k+2) > \dots > \sigma(n) \text{ for some } k \geq 1\}.$$

For example, $(1, 3, 4, 7, 6, 5, 2)$ is a pyramidal permutation whereas $(7, 1, 2, 3, 4, 5, 6)$ is not.

Klyaus [39] gave an $O(n^2)$ dynamic programming algorithm for finding the best pyramidal tour for the TSP. See also Gilmore, Lawler, and Shmoys [32]. Sarvanov and Doroshko [52] employed the concept of pyramidal tours for use in very large scale neighborhood search. Carlier and Villon [19] combined the pyramidal tour neighborhood with a cyclic shift neighborhood in very large scale neighborhood search. They determined empirically that this composite neighborhood performed far better than 2-opt.

Ergun and Orlin [26] defined the pyramidal neighborhood recursively, by means of *pyramidal subsequences*. A subsequence σ is pyramidal if it is a sequence of objects $j, j+1, \dots, n$ for some $1 \leq j \leq n$, and the object labels increase monotonically to n and then decrease. Then, the pyramidal neighborhood N_P consists of all the pyramidal subsequences of objects $1, \dots, n$. A pyramidal subsequence σ of objects $j, j+1, \dots, n$ satisfies the recursive equation

$$\sigma = j, \sigma' \text{ or } \sigma = \sigma', j, \tag{4.5}$$

for some pyramidal subsequence σ' of objects $j+1, j+2, \dots, n$.

Permutation Tree neighborhood

We give the description by Deineko and Woeginger [47]. A permutation tree T over $\{1, \dots, n\}$ is a rooted, ordered tree that satisfies the following.

1. There is a bijective mapping between leaves of T and objects.
2. An interior node has at least two sons.
3. An interior node i with d sons has associated a set of permutations $\Phi_i \subseteq S_d$

A permutation tree T defines a set of permutations N_T as follows. For each node $i \in T$, let T_i be the subtree of T rooted at i . We associate the set N_{T_i} of sequences of objects to each subtree T_i . If i is a leaf of T then $N_{T_i} = \{i\}$. If i is an interior node of T with d sons i_1, \dots, i_d then $N_{T_i} = \cup_{\phi \in \Phi_i} N_{T_{i_{\phi(1)}}}, \dots, N_{T_{i_{\phi(d)}}}$. The set of permutations N_T is the set N_{T_r} defined by the root node $r \in T$.

Booth and Lueker [16] analyzed a special case of permutation trees that they called PQ-trees. Burkard, Deineko, and Woeginger [18] investigated finding the best tour in the PQ-tree neighborhood, which was later generalized by Deineko and Woeginger [47] to include all permutation trees. They also pointed out that pyramidal tours are a special case of permutation trees.

Twisted Sequences

Aurenhammer [10] described permutations in terms of possible derivations from permutation trees. A permutation tree T is a *twisted tree* if it satisfies the following conditions.

1. The identity sequence $\text{id}_n = (1, \dots, n) \in N_T$.
2. Every interior node i with d sons has associated a set of permutations $\Phi_i = \{\text{id}_d, \text{id}_d^-\}$, where $\text{id}_d^- = (n, n-1, \dots, 1)$.

A permutation σ is a *twisted sequence* if it is generated by a twisted tree. That is,

$$N_{\text{Twisted}} = \{\sigma : \sigma \in N_T \text{ for some twisted tree } T\}.$$

For example, $(3, 2, 4, 1)$ is a twisted sequence whereas $(1, 3, 5, 2, 4, 6)$ is not. An alternative way of describing twisted sequences is provided by Deĭneko and Woeginger [47]. They write on page 528:

“Another, equivalent way of defining twisted sequences is as follows: Start with the identity permutation $(1, 2, \dots, n)$ and choose a set of intervals (of cities) over $\{1, 2, \dots, n\}$ such that for every pair of intervals either one of them contains the other one, or the two intervals are disjoint. Then reverse (= twist) for every interval the order of its elements. A permutation is a twisted sequence if and only if it can be derived from the identity permutation via such a reversal process.” We observe that the reverse operation is defined over intervals of objects and not over places. In this sense, the twisted sequence obtained does not depend on the order on which we perform the reverse operations over the intervals.

Congram [21] showed that the number of twisted sequences on n objects is $\Theta((3+2\sqrt{2})^n)$.

Dynasearch neighborhood

Potts and van de Velde [51] defined this class of neighborhoods (see also Congram et al. [22]). A *Dynasearch Neighborhood* is obtained by combining a set of independent simple moves such as 2-exchanges, swaps, or insertions. Two moves for the TSP are said to be independent if the subpaths they are modifying contain no common edges. For example, consider two 2-exchange moves affecting sub-paths $i, i+1, \dots, j$ and $k, k+1, \dots, l$ by breaking edges $(i, i+1), (j-1, j)$ and $(k, k+1), (l-1, l)$ and adding edges $(i, j-1), (i+1, j)$ and $(k, l-1), (k+1, l)$. These two moves are independent if and only if either $j < k$ or $l < i$. For scheduling problems, two moves are independent if the subsequences they modify do not share any job.

The size of the compounded independent moves neighborhood is $\Omega(1.7548^n)$, see Con-

gram [21] and Ergun [26] for two derivations. For the TSP, these neighborhoods can be searched in $O(n^2)$ by dynamic programs as in Potts and van de Velde [51], Congram [21], Ergun and Orlin [27] and Congram et al. [22], and by network flows techniques as in Agarwal et al. [2], Ergun [26], and Ergun et al. [28].

Balas-Simonetti neighborhood

Balas and Simonetti [12] investigated a very large scale neighborhood for the TSP which we denote by N_{BS}^k . In their paper, they define a solution of the TSP as a permutation of places instead of a permutation of cities (objects). We give the definition of N_{BS}^k in terms of a permutation of cities (objects). Let k be a parameter. They consider all tours such that object 1 is visited first and that object i precedes object j whenever $i + k \leq j$ for some fixed parameter k . That is,

$$N_{BS}^k = \{\sigma : \sigma(1) = 1 \text{ and } \sigma^{-1}(i) < \sigma^{-1}(j) \text{ for } i + k \leq j\}.$$

For example, when $n = 6$ and $k = 3$, the sequence $(1, 4, 2, 5, 3, 6)$ belongs to N_{BS}^3 whereas $(1, 4, 5, 2, 3, 6)$ does not.

4.3.3 Exponential Neighborhoods as Sequencing Neighborhood Grammars

In this section, we show how to represent the neighborhoods in the previous section using grammars.

Pyramidal sequences

The following sequence grammar generates the Pyramidal Neighborhood.

$$V_{NT} = \{S\} \cup \{A_{j,n} : 1 \leq j \leq n\}, V_T = \{1, \dots, n\}. \text{ Its production rules are}$$

The Pyramidal Neighborhood Grammar

$$S \rightarrow A_{1,n},$$

$$A_{j,n} \rightarrow j, A_{j+1,n}, \text{ for } 1 \leq j \leq n-1$$

$$A_{j,n} \rightarrow A_{j+1,n}, j, \text{ for } 1 \leq j \leq n-1$$

$$A_{n,n} \rightarrow n.$$

The value of the function $Objects$ on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(A_{j,n}) = \{j, \dots, n\}$ for $1 \leq j \leq n$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

The Pyramidal neighborhood is not symmetric, that is, $N_{\text{Pyramid}} \neq \text{inv}(N_{\text{Pyramid}})$. In Subsections 4.4.3 and 4.6.2 we show that, in the context of the TSP, we can optimize over neighbors N_{Pyramid} and $\text{inv}(N_{\text{Pyramid}})$ in $O(n^2)$ time. The next proposition shows that N_{Pyramid} is indeed the pyramidal neighborhood.

Proposition 4.3.1. *The pyramidal neighborhood grammar generates the pyramidal neighborhood.*

Proof. We will prove that, for any $1 \leq j \leq n$, the language $L(G_{A_{j,n}})$ generated by the nonterminal $A_{j,n}$ as the starting symbol, is the set of all pyramidal subsequences of objects j, \dots, n . When $j = 1$, this claim is equivalent to the proposition. We prove the claim by induction in $|Objects(A_{j,n})| = n - j + 1$. When $|Objects(A_{j,n})| = 1$, the nonterminal $A_{j,n}$ is in fact $A_{n,n}$, and the language $L(G_{A_{j,n}})$ is equal to the set of all pyramidal subsequences of object n , namely the singleton set $\{n\}$. Given $A_{j,n}$, and assume the claim is true for $A_{j+1,n}$. There are two possible production rules that can be applied to $A_{j,n}$, namely $A_{j,n} \rightarrow j, A_{j+1,n}$ and $A_{j,n} \rightarrow A_{j+1,n}, j$. Therefore, the language $L(G_{A_{j,n}})$ are the subsequences σ of objects j, \dots, n that can be written as j, σ' or σ', j where $\sigma' \in L(G_{A_{j+1,n}})$. By inductive hypothesis, $L(G_{A_{j+1,n}})$ is the set of all the pyramidal subsequences of objects $j+1, \dots, n$. Therefore, $L(G_{A_{j,n}})$ satisfies the recursion (4.5), and thus it is the set of all the pyramidal subsequences of objects j, \dots, n . \square

Permutation Trees and their Grammars

Definition 4.3.2. *A sequencing neighborhood grammar G is tree-based if for any non-terminals A, B then either*

- $Objects(A) \cap Objects(B) = \emptyset$, or
- $Objects(A) \subset Objects(B)$, or
- $Objects(B) \subset Objects(A)$.

We can represent any tree-based structure as a tree, where each non-terminal is an internal node of the tree; each terminal is a leaf of the tree. Also, there is an arc from R to R' if $Objects(R) \subset Objects(R')$ and there is no other non-terminal R'' with $Objects(R) \subset Objects(R'') \subset Objects(R')$. Such a tree is called a *permutation tree*, as per Deineko and Woeginger [47].

It is easy to see that there is a correspondence between the neighborhoods generated by a permutation tree and neighborhoods defined by a tree-based grammar. A neighborhood generated by a permutation tree T is also generated by the tree-based grammar with non-terminals being the interior nodes of T , and with production rules that correspond to the permutations associated to each interior node of T . The tree associated with this tree-based grammar is precisely T . The converse is also true, and thus the following proposition holds.

Proposition 4.3.3. *A neighborhood generated by a permutation tree T can be generated by a tree-based grammar G , and vice versa. There is a one-to-one correspondence between the non-terminals of G and the interior nodes of T , and the number of production rules associated to each non-terminal of G is the number of permutations associated to the corresponding interior node of T*

Twisted Sequence Neighborhood

We describe a simply stated grammar that generates all twisted sequences. Let the set of non-terminals and the set of terminals be $V_{NT} = \{S\} \cup \{A_{i,j} : 1 \leq i \leq j \leq n\}$, $V_T = \{1, \dots, n\}$ respectively.

Its production rules are

Twisted Sequence Neighborhood Grammar

$$S \rightarrow A_{1,n},$$

$$A_{i,j} \rightarrow i, \dots, j \text{ for } 1 \leq i \leq j \leq n,$$

$$A_{i,j} \rightarrow A_{i,k}, A_{k+1,j} \text{ for } 1 \leq i \leq k < j \leq n,$$

$$A_{i,j} \rightarrow A_{k+1,j}, A_{i,k} \text{ for } 1 \leq i \leq k < j \leq n,$$

The value of the function *Objects* on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(A_{i,j}) = \{i, \dots, j\}$ for $1 \leq i \leq j \leq n$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

The following proposition holds.

Proposition 4.3.4. *The twisted neighborhood grammar generates the twisted neighborhood.*

Proof. By definition, the twisted neighborhood is the set of all sequences generated by all twisted trees. Fixed a twisted tree T , its correspondent tree-based grammar has non-terminals R with consecutive labeled objects, namely $Objects(R) = \{i, \dots, j\}$ for some $1 \leq i \leq j \leq n$, and production rules as the ones of the twisted neighborhood grammar. Therefore, the twisted neighborhood grammar generates a neighborhood that includes the twisted neighborhood.

The converse is also true. Given a sequence σ generated by the twisted neighborhood, it is easy to construct a twisted tree T from its parse tree that generates σ . \square

Congram [21] shows that the number of twisted sequences on n objects is $\Theta((3 + 2\sqrt{2})^n)$. The number of production rules of this grammar is $K = O(n^3)$. The generic DP algorithm runs in time $O(Kn^3)$ on a TSP grammar with K production rules and n cities (see Section 4.4.1). Therefore, it runs in time $O(n^6)$ when specialized to the Twisted Sequence Neighborhood, which is a factor of n smaller than the time obtained by Deĭneko and Woeginger [47].

The Twisted Sequence Neighborhood is symmetric as the next proposition shows.

Proposition 4.3.5. *The Twisted Sequence Neighborhood is symmetric. That is, σ^{-1} is a twisted sequence if σ is a twisted sequence.*

Proof. Let $\sigma = (\sigma(1), \dots, \sigma(n))$ be a twisted sequence and let T be its parse tree. The label R of a node of T is either a terminal j or a non-terminal S or $A_{i,j}$. To simplify notation, we write S as $A_{1,n}$ and j as $A_{j,j}$. If $A_{i,j}$ is a label of T , the objects $\{i, \dots, j\}$ are placed consecutively by σ (although they may not follow this order). For each label $A_{i,j}$ of T , let $t - 1$ be the number of objects placed by σ before placing any of the objects $\{i, \dots, j\}$. We claim that if we replace each label $A_{i,j}$ of the parse tree T by $A_{t,t+j-i}$ we obtain a new parse tree T' which is the parse tree of σ^{-1} in the Twisted Sequence Neighborhood Grammar.

First, we have to prove that T' is a valid parse tree of the Twisted Sequence Neighborhood Grammar. We observe that a production rule of the parse tree T , say $A_{i,j} \rightarrow A_{k+1,j}A_{i,k}$ for some $i \leq k \leq j$, becomes $A_{t,t+j-i} \rightarrow A_{t+j-k,t+j-i}A_{t,t+j-k-1}$ in T' for some t . The latter is a valid production rule of the Twisted Sequence Neighborhood Grammar. Since S remains the same and each terminal goes to a terminal after the transformation, T' is a valid parse tree of the Twisted Sequence Neighborhood Grammar.

It remains to prove that ω , the sequence generated by T' , is σ^{-1} . We will prove that, given a transformed symbol $A_{t,t+j-i}$ in T' that corresponds to the symbol $A_{i,j}$ in T , the number of objects placed before placing any of the objects $\{t, \dots, t + j - i\}$ by the permutation ω generated by T' is $i - 1$. We will prove this by induction on the level of $A_{t,t+j-i}$ in T' . For the start symbol S , it is clear that there are no objects placed by ω before the objects $\{1, \dots, n\}$ in T' . Therefore the inductive hypothesis holds for S . Let p be a production rule of T , say $A_{i,j} \rightarrow A_{k+1,j}A_{i,k}$. This production rule becomes $A_{t,t+j-i} \rightarrow A_{t+j-k,t+j-i}A_{t,t+j-k-1}$ for some t in T' . Assume by inductive hypothesis that the number of objects placed by ω before $\{t, \dots, t + j - i\}$ is $i - 1$. Then, the number of objects placed before $\{t + j - k, \dots, t + j - i\}$ is also $i - 1$ and the number of objects placed before $\{t + j - k, \dots, t + j - i\}$ is $(i - 1) + (t + j - i - t - j + k) = k - 1$. Therefore the inductive hypothesis holds.

Let j be the label of a leaf node in T . This label says that $\sigma(t) = j$ for some t . The label j in T is replaced by t in T' , and we proved that the number of objects placed before object

t by ω is $j - 1$. Therefore, $\omega(t) = j$, which means that ω is the inverse of σ . □

Semi-Twisted Sequence Neighborhood

In this subsection we present a smaller neighborhood, which is very similar to the twisted neighborhood. It has fewer rules and is written in extended normal form. Therefore, we can optimize faster than in the Twisted Sequence Neighborhood.

Semi-Twisted Sequence Neighborhood Grammar

$$\begin{aligned}
 S &\rightarrow A_{1,n}, \\
 A_{i,j} &\rightarrow i, A_{i+1,j} \text{ for } 1 \leq i < j \leq n, \\
 A_{i,j} &\rightarrow A_{i+1,j}, i \text{ for } 1 \leq i < j \leq n, \\
 A_{i,j} &\rightarrow A_{i,j-1}, j \text{ for } 1 \leq i < j \leq n, \\
 A_{i,j} &\rightarrow j, A_{i,j-1} \text{ for } 1 \leq i < j \leq n.
 \end{aligned}$$

The number of production rules of this grammar is $K = O(n^2)$. For the TSP, we can optimize in $O(n^3)$ time as Proposition 4.4.10 in Subsection 4.4.3 shows. It includes the Pyramidal neighborhood and has $\Theta((2 + \sqrt{2})^n)$ sequences as the next proposition shows.

Proposition 4.3.6. *The Semi-Twisted Sequence Neighborhood has $\Theta((2 + \sqrt{2})^n)$ sequences.*

Proof. Let $f(n)$ be the number of sequences of the Semi-Twisted Sequence Neighborhood with n objects. It is easy to check that $f(1) = 1$ and $f(2) = 2$. The following recursive relation holds.

$$f(n) = 4f(n - 1) - 2f(n - 2) \text{ for } n \geq 3. \tag{4.6}$$

To see why, it is easy to see from the production rules that a sequence in this neighborhood either starts or ends at object 1 or object n .

The number of sequences in this neighborhood that starts at object 1 (object n) is $f(n - 1)$. Similarly, the number of sequences that ends at object 1 (object n) is $f(n - 1)$. We have to subtract the sequences that are counted twice. That is, the sequences that start at object 1 and end at object n or vice versa. The number of such sequences is $2f(n - 2)$.

Thus equation (4.6) holds. This equation implies that the size of the Semi-Twisted Sequence Neighborhood is asymptotically $c\xi^n$, where $\xi = 2 + \sqrt{2}$ is the largest root of the polynomial $X^2 - 4X + 2$, and $c > 0$ is a constant (see the book by Graham, Knuth and Patashnik [33] for results about recursion formulas). \square

The Semi-Twisted Sequence Neighborhood is symmetric.

Proposition 4.3.7. *The Semi-Twisted Sequence Neighborhood is symmetric.*

Proof. It has a similar proof to Proposition 4.3.5. \square

Restricted Dynamic Programs

We first describe a sequencing neighborhood grammar for generating all the permutations whose initial object is object 1, which we refer to as the “complete sequencing neighborhood grammar”. This particular grammar has $n2^{n-1}$ different production rules, and generates $(n - 1)!$ permutations. We then give related grammars in which the number of production rules is polynomial, and the neighborhood is exponential.

Complete sequencing Neighborhood Grammar

$$S \rightarrow A_{\{1, \dots, n\}},$$

$$A_R \rightarrow A_{R \setminus \{j\}}, j \text{ for each } j \in R \setminus \{1\}, \text{ and for each subset } R \subseteq \{1, \dots, n\} \text{ with } 1 \in R,$$

$$A_{\{1\}} \rightarrow 1.$$

The value of the function *Objects* on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(A_R) = R$ for each subset $R \subseteq \{1, \dots, n\}$ with $1 \in R$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

In the following, let \mathcal{S} be any collection of subsets of $\{1, \dots, n\}$ with the property that for each $R \in \mathcal{S}$, $1 \in R$. We now create a subset of the complete sequence grammar, by restricting production rules to involve only non-terminals corresponding to sets in \mathcal{S} .

Restricted sequencing Neighborhood Grammar

$$S \rightarrow A_{\{1, \dots, n\}},$$

$$A_R \rightarrow A_{R-j}, j \text{ for each } R \in \mathcal{S}, \text{ and for each } j \in R \text{ such that } R - j \in \mathcal{S},$$

$$A_{\{1\}} \rightarrow 1.$$

Balas and Simonetti [12] investigated a very large scale neighborhood for the TSP which we denote by N_{BS} . They consider all tours such that city 1 is visited first and that city i precedes city j whenever $i + k \leq j$ for some fixed parameter k . That is, the neighborhood N_{BS} contains all permutations σ such that $\sigma(1) = 1$ and that object $\sigma^{-1}(i) < \sigma^{-1}(j)$ whenever $i + k \leq j$. We note that in their paper, a permutation represents a permutation of places instead of a permutation of objects. Their definition is different from, but equivalent to, the one we give. The Balas-Simonetti neighborhood can be searched in time $O(k^2 2^k n)$ (see [11] or [12]). Its running time is linear in n for fixed k . We will show that this neighborhood can be represented as a restricted sequence grammar. In Section 4.4.3 we show that, when used to find the best neighbor in N_{BS} , the running time of the generic DP algorithm is also $O(k^2 2^k n)$.

Let $\mathcal{S}_{BS} := \{R \subseteq \{1, \dots, n\} \text{ such that for } i \in R, j \notin R : i - k < j\}$. Then the Balas-Simonetti neighborhood corresponds to the restricted sequencing Neighborhood grammar with \mathcal{S} replaced by \mathcal{S}_{BS} .

If we consider $R \in \mathcal{S}_{BS}$ such that the highest index object in $Objects(R)$ is object j^* , then we note that $i \in R$ for $i < j^* - k$, and $i \notin R$ for $i > j^*$. We conclude that there are $O(2^k)$ elements of \mathcal{S}_{BS} with a highest index object of j^* , and thus $|\mathcal{S}_{BS}| = O(2^k n)$. Moreover, we will show in Proposition 4.3.9 implies that the number of production rules that can be applied to a particular non-terminal R is $O(k)$. Therefore the total number of production rules is $K_{BS} = O(k 2^k n)$.

The inverse neighborhood of N_{BS} is the set of permutations $\sigma \in S_n$ such that $\sigma^{-1} \in N_{BS}$. A priori, there is no reason to prefer one over the other since both neighborhoods have the

same number of elements. The inverse neighborhood of N_{BS} can be expressed as a restricted sequencing Neighborhood grammar as well as a context-sensitive sequencing neighborhood grammar.

As the next proposition shows, the inverse neighborhood of N_{BS} is the set of all permutations σ such that $\sigma(1) = 1$ and that for two positions i, j such that $i + k \leq j$, then $\sigma(i) < \sigma(j)$.

Proposition 4.3.8. *The neighborhood*

$$N_{IBS} := \{\sigma \in S_n : \sigma(1) = 1 \text{ and } i + k \leq j \text{ implies } \sigma(i) < \sigma(j)\}$$

is the inverse neighborhood of N_{BS} . That is, $N_{IBS} = \{\sigma^{-1} : \sigma \in N_{BS}\}$.

Proof. A permutation $\sigma \in S_n$ belongs to N_{BS} if and only if

1. $\sigma(1) = 1$.
2. If $i + k \leq j$ then $\sigma^{-1}(i) < \sigma^{-1}(j)$.

If $\omega = \sigma^{-1}$, then

1. $\omega(1) = 1$.
2. If $i + k \leq j$ then $\omega(i) < \omega(j)$.

By definition of N_{IBS} , this is equivalent to say that $\omega \in N_{IBS}$. □

The Balas-Simonetti neighborhood and its inverse neighborhood are included in a neighborhood which we call the *Enlarged Balas-Simonetti Neighborhood* as the following proposition shows.

Proposition 4.3.9. *The Balas-Simonetti neighborhood N_{BS} and its inverse neighborhood N_{IBS} are included in the Enlarged Balas-Simonetti neighborhood*

$$N_{EBS} := \{\sigma \in S_n : \sigma(1) = 1 \text{ and } |\sigma(i) - i| \leq k \text{ for all } i\}.$$

Proof. We will first prove that N_{IBS} is symmetric. That is, we will prove that if $\sigma \in N_{EBS}$ then $\sigma^{-1} \in N_{EBS}$. If a permutation $\sigma \in S_n$ belongs to N_{EBS} then

1. $\sigma(1) = 1$.
2. $|\sigma(i) - i| \leq k$ for all i .

It is clear that $\sigma(1) = 1$ implies $\sigma^{-1}(1) = 1$. To see why the second condition holds for σ^{-1} , let us fix i and let $j = \sigma(i)$. Then, $\sigma(i) - i = j - \sigma^{-1}(j)$. Therefore, $|\sigma(i) - i| \leq k$ is equivalent to $|j - \sigma^{-1}(j)| \leq k$ for $j = \sigma(i)$. Since the restriction $|\sigma(i) - i| \leq k$ holds for all i , the restriction $|j - \sigma^{-1}(j)| \leq k$ also holds for j such that $j = \sigma(i)$ for some i . The permutation σ is a bijective function and therefore the last restriction is equivalent to the restriction that $|j - \sigma^{-1}(j)| \leq k$ holds for all j . Therefore, σ^{-1} satisfies the second restriction and thus N_{EBS} is symmetric. Thus, in order to prove that Balas-Simonetti neighborhood and its inverse neighborhood are included in N_{EBS} , it is enough to prove that N_{IBS} is included in N_{EBS} .

Given a permutation $\sigma \in N_{IBS}$, we prove that $|\sigma(i) - i| \leq k$ for all $1 \leq i \leq n$ by contradiction. Assume that there exist $\sigma \in N_{IBS}$, and a position i such that $|\sigma(i) - i| > k$. Then either $\sigma(i) < i - k$ or $\sigma(i) > i + k$. Assume that $\sigma(i) < i - k$ (the case where $\sigma(i) > i + k$ is similar). By definition of N_{IBS} , any permutation σ in N_{IBS} satisfies that if $i + k \leq j$ then $\sigma(i) < \sigma(j)$. The set $L_i = \{j : j + k \leq i\}$ has $i - k$ elements. Its image $\{\sigma(j) : j + k \leq i\}$ also has $i - k$ elements. Any $j \in L_i$ satisfies that

$$\sigma(j) < \sigma(i) < i - k$$

and therefore the set $\{\sigma(j) : j + k \leq i\} \subseteq \{l : l + k < i\}$ has at most $i - k - 1$ elements, which contradicts the fact that it has $i - k$ elements. \square

The Enlarged Balas-Simonetti neighborhood can be expressed as a restricted sequencing neighborhood grammar as follows. Let \mathcal{S}_{EBS} be a collection of subsets R of $\{1, \dots, n\}$ such that $1 \in R$, any i such that $i < |\text{Objects}(R)| - k$ belongs to R , and that any i such that $i > |\text{Objects}(R)| + k$ does not belong to R . Then the Enlarged Balas-Simonetti neighborhood

corresponds to the restricted sequencing Neighborhood grammar with \mathcal{S} replaced by \mathcal{S}_{EBS} . If we consider $R \in \mathcal{S}_{EBS}$ such that $Objects(R) = m$, then we note that $i \in R$ for $i < m - k$, and $i \notin R$ for $i > m + k$. We conclude that there are $\binom{2k}{k} = O\left(\frac{4^k}{\sqrt{k}}\right)$ elements of \mathcal{S}_{EBS} with m objects. The last bound holds because of Stirling's bound of factorials: $\sqrt{2\pi n}n^{n+1/2}e^{-n+1/(12n+1)} < n! < \sqrt{2\pi n}n^{n+1/2}e^{-n+1/(12n)}$ (see e.g. Feller [29]). Thus, $|\mathcal{S}_{EBS}| = O\left(\frac{4^k}{\sqrt{k}}n\right)$. Moreover, Proposition 4.3.9 implies that the number of production rules that can be applied to a particular non-terminal R is $O(k)$. Therefore the total number of production rules is $K_{EBS} = O(k^{0.5}4^k n)$. In Section 4.4.3 we show that, when used to find the best neighbor in the TSP neighborhood N_{KBS} , the running time of the generic DP algorithm is $O(k^{1.5}4^k n)$, which is linear for fixed values of k . This running time matches the running time of the algorithm developed by Balas in [11].

Dynasearch Neighborhoods

We now describe several Dynasearch neighborhood grammars. The following Dynasearch neighborhood is based on combining independent swap moves. Given a sequence $\pi = (\pi(1), \dots, \pi(n))$, the swap neighborhood generates solutions by interchanging the positions of objects $\pi(i)$ and $\pi(j)$ for $1 \leq i < j \leq n$. For example let $\pi = (1, 2, 3, 4, 5, 6)$, let $i = 2$ and $j = 5$. Then $\pi' = (1, 5, 3, 4, 2, 6)$ is obtained from π after we swap i, j . The set of terminals and non-terminals are $V_{NT} = \{S\} \cup \{A_{i,j} : 1 \leq i \leq j \leq n\}$, $V_T = \{1, \dots, n\}$. Its production rules are

The Dynasearch swap Neighborhood Grammar

$$S \rightarrow A_{1,n},$$

$$A_{1,j} \rightarrow A_{1,i-1}, A_{i,j} \text{ for } 2 \leq i \leq j \leq n,$$

$$A_{i,j} \rightarrow i, \dots, j \text{ for } 1 \leq i \leq j - 1 \leq n - 1,$$

$$A_{i,j} \rightarrow i, j - 1, i + 2, \dots, j - 2, i + 1, j \text{ for } 1 \leq i \leq j - 3 \leq n - 3.$$

The value of the function $Objects$ on each terminal and non-terminal is $Objects(S) =$

$\{1, \dots, n\}$, $Objects(A_{i,j}) = \{i, \dots, j\}$ for $1 \leq i \leq j \leq n$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

The following Dynasearch neighborhood is based on 2-opt.

The Dynasearch 2-exchange Neighborhood Grammar

$$S \rightarrow A_{1,n},$$

$$A_{1,j} \rightarrow A_{1,i-1}, A_{i,j} \text{ for } 2 \leq i \leq j \leq n,$$

$$A_{i,j} \rightarrow i, \dots, j \text{ for } 1 \leq i \leq j - 1 \leq n - 1,$$

$$A_{i,j} \rightarrow i, j - 1, j - 2, \dots, i + 2, i + 1, j \text{ for } 1 \leq i \leq j - 3 \leq n - 3.$$

The following Dynasearch neighborhood is based on combining independent insertion moves. Without compounding independent moves, we are permitted to take object i and insert it immediately before object j for $i < j - 1$. If we are permitted to compound independent moves, we obtain the following grammar.

The Dynasearch Insertion Neighborhood Grammar

$$S \rightarrow A_{1,n},$$

$$A_{1,j} \rightarrow A_{1,i-1}, A_{i,j} \text{ for } 2 \leq i \leq j \leq n,$$

$$A_{i,j} \rightarrow i, \dots, j \text{ for } 1 \leq i \leq j - 1 \leq n - 1,$$

$$A_{i,j} \rightarrow i, i + 2, \dots, j - 1, i + 1, j \text{ for } 1 \leq i \leq j - 3 \leq n - 3.$$

In addition, one can obtain larger neighborhoods by making composite neighborhoods. For example, if one permits exchanges, swaps or insertions, one obtains the following Dynasearch neighborhood.

The Dynasearch Exchange/Swap/Insertion Neighborhood Grammar

$$\begin{aligned}
 S &\rightarrow A_{1,n}, \\
 A_{1,j} &\rightarrow A_{1,i-1}, A_{i,j} \text{ for } 2 \leq i \leq j \leq n, \\
 A_{i,j} &\rightarrow j, i+1, \dots, j-1, i \text{ for } 1 \leq i \leq j \leq n, \\
 A_{i,j} &\rightarrow i, \dots, j \text{ for } 1 \leq i \leq j-1 \leq n-1, \\
 A_{i,j} &\rightarrow i, i+2, \dots, j-1, i+1, j \text{ for } 1 \leq i \leq j \leq n,
 \end{aligned}$$

We will show in Section 4.4.3 that the generic algorithm for searching the TSP grammar neighborhood runs in $O(n^2)$ time on these Dynasearch neighborhoods.

It is also possible to weaken the notion of independence as in Ergun and Orlin [27]. For example, one can define that $i, i+1, \dots, j$ and $k, k+1, \dots, l$ are *weakly independent* if $i < j$ or $l < i$. In the case of weak independence, we obtain a larger neighborhood that is referred to as the weak Dynasearch neighborhood. At the same time, it takes longer for the generic dynamic programming algorithm to search the neighborhood, as pointed out in Section 4.4.3.

The Weak Dynasearch 2-exchange Neighborhood Grammar

$$\begin{aligned}
 S &\rightarrow A_{1,n}, \\
 A_{1,j} &\rightarrow A_{1,i-1}, A_{i,j} \text{ for } 2 \leq i \leq j \leq n, \\
 A_{i,j} &\rightarrow i, \dots, j \text{ for } 1 \leq i \leq j-1 \leq n-1, \\
 A_{i,j} &\rightarrow j, j-1, \dots, i+1, i \text{ for } 1 \leq i \leq j-1 \leq n-1,
 \end{aligned}$$

Similarly, one can create weak Dynasearch neighborhoods for a variety of other neighborhoods based on compounding weakly independent moves.

4.3.4 Compound Neighborhoods

Two grammars $G = (V_{NT}, V_T, P, S)$ and $G' = (V'_{NT}, V'_T, P', S)$ can be combined to generate a *compound* grammar $G \cup G' := (V_{NT} \cup V'_{NT}, V_T \cup V'_T, P \cup P', S)$. The number of production

rules of the compound grammar is at most the sum of the number of production rules of G, G' . The neighborhood generated by the compound grammar includes the neighborhood generated by G and G' , and can be searched efficiently, as Theorem 4.4.3 shows.

Pyramidal and Dynasearch Compound Neighborhoods

The following neighborhood is obtained by compounding the pyramidal and the Dynasearch 2-exchange grammars. The value of the function *Objects* on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(A_{i,j}) = \{i, \dots, j\}$ for $1 \leq i \leq j-1 \leq n-1$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$.

**The Pyramidal and Dynasearch 2-exchange
Neighborhood Compound Grammar**

$$S \rightarrow A_{1,n},$$

$$A_{j,n} \rightarrow j, A_{j+1,n}, \text{ for } 1 \leq j \leq n-1$$

$$A_{j,n} \rightarrow A_{j+1,n}, j, \text{ for } 1 \leq j \leq n-1$$

$$A_{1,j} \rightarrow A_{1,i-1}, A_{i,j} \text{ for } 2 \leq i \leq j \leq n,$$

$$A_{i,j} \rightarrow i, \dots, j \text{ for } 1 \leq i \leq j-1 \leq n-1,$$

$$A_{i,j} \rightarrow i, j-1, j-2, \dots, i+2, i+1, j \text{ for } 1 \leq i \leq j-3 \leq n-3.$$

4.3.5 Context-Sensitive sequence grammars

In this section we work with context-sensitive sequence grammars. We show that the complete neighborhood can be generated by a Context-Sensitive sequence grammar of polynomial size. This shows that the Context-Sensitive sequence grammars are very powerful. As a tradeoff, we cannot expect to find the best neighbor in a Context-Sensitive sequence grammar in polynomial time unless $P = NP$.

The Complete Neighborhood Context-Sensitive Grammar

- (1) $S \rightarrow A_1, \dots, A_n,$
- (2) $A_i A_j \rightarrow A_i B_{ij}$ for all $1 \leq i < j \leq n,$
- (3) $A_i B_{ij} \rightarrow C_{ij} B_{ij}$ for all $1 \leq i < j \leq n,$
- (4) $C_{ij} B_{ij} \rightarrow C_{ij} A_i$ for all $1 \leq i < j \leq n,$
- (5) $C_{ij} A_i \rightarrow A_j A_i$ for all $1 \leq i < j \leq n,$
- (6) $A_i \rightarrow i$ for $1 \leq i \leq n.$

Applying Rules (2*ij*) to (5*ij*) swaps non-terminals A_i and A_j , and so every permutation can be obtained. Also, there are no other strings that can be derived from the grammar. To see this, note the following. If Rule (2*ij*) is performed, the only way of get rid of B_{ij} is to have Rule (4*ij*) performed. In order to perform Rule (4*ij*), we have to had performed Rule (3*ij*) so as to generate the non-terminal C_{ij} . Then, the only way of get rid of C_{ij} is to have rule (5*ij*) performed. Moreover, non of these rules can be applied twice.

The number of production rules is $O(n^2)$. Therefore, this context sensitive sequence grammar generates all permutations of S_n (and no other string of terminals) with a polynomial number of production rules.

A more restrictive class of context-sensitive grammar define permutation neighborhoods which can be optimized efficiently. This class will be discussed in Section 4.4.10.

4.4 Algorithms

4.4.1 A generic solution procedure for TSP grammars using dynamic programming

The primary result of this subsection is a dynamic programming algorithm for finding the best tour in a neighborhood that is generated by a sequence grammar G . The initial tour is $\pi = (\pi(1), \dots, \pi(n))$. We denote by $f(i, j)$ the arc cost $c_{\pi(i), \pi(j)}$. In what follows, we associate

to a terminal, non-terminal or a production rule a number of sets which are going to be used later in the DP optimizer. For a given terminal or non-terminal R , let $States(R)$ be the set of pairs of cities (i, j) such that there is a path generated by the grammar G_R from city i to city j that passes through all the cities of $Objects(R)$. We also define the sets $InitObjects(R) = \{i : (i, j) \in States(R) \text{ for some } j\}$, $EndObjects(R) = \{j : (i, j) \in States(R) \text{ for some } i\}$. We define $States(p)$, $InitObjects(p)$ and $EndObjects(p)$ for a production rule p in the same manner. For each terminal R , the set $States(R)$ has one element. These sets satisfy the following recursive relations.

1. If a is a terminal with $Objects(a) = \{k\}$, then

$$States(a) = \{(k, k)\} InitObjects(a) = \{k\}, \text{ and } EndObjects(a) = \{k\}.$$
2. Suppose p is the production rule $A \rightarrow R_1, R_2, \dots, R_r$. Then,

$$States(p) = \{(i, j) : i \in InitObjects(R_1), j \in EndObjects(R_r)\};$$

$$InitObjects(p) = InitObjects(R_1); EndObjects(p) = EndObjects(R_r).$$
3. If R is a non-terminal, then

$$States(R) = \cup_{\{p:p \text{ applies to } R\}} States(p);$$

$$InitObjects(R) = \cup_{\{p:p \text{ applies to } R\}} InitObjects(p);$$

$$EndObjects(R) = \cup_{\{p:p \text{ applies to } R\}} EndObjects(p).$$

These sets do not change from iteration to iteration of the local search algorithm. Therefore, they are computed once. We can assume that the time to compute them is amortized by the time spent by the local search algorithm. The complexity of their computation is as follows.

Proposition 4.4.1. *Given a sequence grammar with K production rules for a problem with n cities, the sets $States$, $InitCities$, $EndCities$, corresponding to all terminals and non-terminals R and all production rules p , can be computed in $O(Kn^2)$ time using the recursion formulas given above.*

Proof. It is easy to see that, for each terminal a , the sets $States(a)$, $InitObjects(a)$, and $EndObjects(a)$ are computed in $O(1)$ time, for a total of $O(n)$ time. The computation of

$States(p)$, $InitObjects(p)$, $EndObjects(p)$ for each production rule $p : A \rightarrow R_1, R_2, \dots, R_r$ from the values $InitObjects(R_1)$ and $EndObjects(R_r)$ take $O(n^2)$ time, for a total $O(n^2)$ time. The computation of $States(R)$, $InitObjects(R)$, $EndObjects(R)$ for all non-terminal R take $O(Kn^2)$ time. \square

Given a terminal or non-terminal R and cities $i, j \in Objects(R)$, we denote by $V(i, R, j)$ the minimum cost path from $\pi(i)$ to $\pi(j)$ passing through all the cities of $\pi(Objects(R)) = \{\pi(i) : i \in Objects(R)\}$ as generated by the grammar. Given a terminal or non-terminal R , city $i \in Objects(R)$ and city $j \notin Objects(R)$, we denote by $\bar{V}(i, R, j)$ the minimum cost path from $\pi(i)$ to $\pi(j)$, passing through all the cities of $\pi(Objects(R))$ as generated by the grammar. We note that the difference between $V(i, R, j)$ and $\bar{V}(i, R, j)$ is that, in the latter, the city j does not belong to $Objects(R)$.

Similarly, given a production rule $p : A \rightarrow R_1 R_2 \dots R_r$ and cities $i, j \in Objects(A)$, we denote by $V(i, p, j)$ the minimum cost path from $\pi(i)$ to $\pi(j)$ passing through all the cities of $\pi(Objects(A))$ as generated by p .

We refer to a triple $i - R - j$ as a state of the dynamic programming recursion. The best tour generated by the grammar has value $\min_{i,j} \{V(i, S, j) + f(j, i)\}$. The following is a dynamic programming recursion for computing the best tour in a TSP Neighborhood grammar.

1. If a is a terminal with $Objects(a) = \{k\}$, then

$$V(k, a, k) = 0.$$

2. Suppose p is the production rule $A \rightarrow a_1 a_2 \dots a_r B$. Let $Objects(a_k) = \{i_k\}$ for $1 \leq k \leq r$. Then,

For $(i_1, j) \in States(p)$:

$$V(i_1, p, j) = \min\{\sum_{k=1}^{r-1} f(i_k, i_{k+1}) + f(i_r, t) + V(t, B, j) : t \in InitObjects(B)\}.$$

3. Suppose p is the production rule $A \rightarrow R_1, R_2, \dots, R_r$.

For $k = 1$ to $r - 1$ and for $i \in InitObjects(R_1), j \in IniObjects(R_{k+1})$:

$$\bar{V}(i, R_1, R_2, \dots, R_k, j) = \min\{V(i, R_1, R_2, \dots, R_k, s) + f(s, j) : s \in EndObjects(R_k)\}.$$

For $k = 2$ to r and for $i \in \text{InitObjects}(R_1), j \in \text{EndObjects}(R_k)$:

$$V(i, R_1, R_2, \dots, R_k, j) = \min\{\bar{V}(i, R_1, \dots, R_{k-1}, t) + V(t, R_k, j) : t \in \text{InitObjects}(R_k)\}.$$

For $(i, j) \in \text{States}(p)$:

$$V(i, p, j) = V(i, R_1, \dots, R_r, j)$$

4. If R is a non-terminal, then

For $(i, j) \in \text{States}(R)$:

$$V(i, R, j) = \min\{V(i, p, j) : p \text{ is a production rule and } p \text{ applies to } R\}.$$

Proposition 4.4.2. *Suppose that p is the production rule $p : A \rightarrow R_1 R_2 \dots R_r$. Then the time to compute $V(i, p, j)$ for all $(i, j) \in \text{States}(p)$ from the values for R_1, R_2, \dots, R_r is $O(n^3)$.*

Proof. For each $i \in \text{InitObjects}(R_1)$ and each $j \in \text{EndObjects}(R_k)$, the value $V(i, R_1, R_2, \dots, R_k, j)$ is equal to the cost of the shortest Hamiltonian path from city $\pi(i)$ to city $\pi(j)$ and passing through all cities in $\pi(\text{Objects}(R_1 \dots R_k))$, as generated by the grammar. Similarly, for each $i \in \text{InitObjects}(R_1)$ and each $j \in \text{InitObjects}(R_{k+1})$, the value $\bar{V}(i, R_1, R_2, \dots, R_k, j)$ is equal to the cost of the shortest Hamiltonian path from city $\pi(i)$ to city $\pi(j)$ and passing through all cities in $\pi(\text{Objects}(R_1 \dots R_k)) \cup \{\pi(j)\}$. The recursion formulas given in (3) compute the shortest path from any city $i \in \text{InitObjects}(R_1)$ to any city $j \in \text{EndObjects}(R_r)$. The running time to compute them is $O(n^3)$. For each $2 \leq k \leq r$ and for each $i \in \text{InitObjects}(R_1), j \in \text{EndObjects}(R_k)$, it takes $O(|\text{InitObjects}(R_k)|)$ time to compute $V(i, R_1, \dots, R_k, j)$ from the values $\bar{V}(\cdot)$ and $V(\cdot, R_k, \cdot)$. For each $2 \leq k \leq r$ and for each $i \in \text{InitObjects}(R_1)$, it takes $O(|\text{InitObjects}(R_k)| \times |\text{EndObjects}(R_k)|) = O(|\text{InitObjects}(R_k)| \times n)$ time to compute $V(i, R_1, \dots, R_k, j)$ for all $j \in \text{EndObjects}(R_k)$. Finally, for every $i \in \text{InitObjects}(R_1)$ and every $2 \leq k \leq r$, it takes

$$\sum_{i \in \text{InitObjects}(R_1), 2 \leq k \leq r} O(|\text{InitObjects}(R_k)| \times n) = \sum_{2 \leq k \leq r} O(|\text{InitObjects}(R_k)| \times n^2) = O(n^3) \quad (4.7)$$

time. Similar bound holds for the computation of $\bar{V}(i, R_1, \dots, R_k, j)$ for all $1 \leq k \leq r - 1$ and all $i \in \text{InitObjects}(R_1), j \in \text{InitObjects}(R_{k+1})$. \square

In order to use the recursion formulas, we first construct the auxiliary graph $G' = (V', A')$ where $V' = V_T \cup V_{NT} \cup P$. For any production rule $p : A \rightarrow R_1, \dots, R_r$, the arcs $(A, p), (p, R_1), \dots, (p, R_r)$ belong to A' . This graph has $O(Kn)$ nodes and $O(Kn)$ arcs. This graph is acyclic; otherwise the language $L(G)$ is infinite. This graph defines an order among its nodes which is obtained by breadth first search. This order is compatible with the recursion formulas in the sense that $V(i, p, j), V(i, R, j)$ are constructed using higher order terms. We then compute the values $V(i, p, j), V(i, R, j)$ following this order.

Theorem 4.4.3. *Let K be the number of production rules of a TSP neighborhood grammar for a problem with n cities. Then the time to compute the best neighbor generated by the grammar is $O(Kn^3)$. If the grammar is in extended normal form, the time is $O(Kn^2)$. If the grammar is a left (right) regular grammar such that $\sigma(n) = n$ ($\sigma(1) = 1$), then the time to compute the best neighbor is $O(Kn)$.*

Proof. Compute the order in G' can be done in $O(Kn)$ steps. Each terminal value $V(k, a, k)$ is computed in $O(1)$ steps. Now consider a production rule p and consider the result in Proposition 4.4.2. The time to compute $V(i, p, j)$ for all $i - p - j$ is then $O(Kn^3)$. In order to bound the time to compute $V(i, R, j)$ for all $R \in V_{NT}, (i, j) \in States(R)$, we observe that $States(R) = O(n^2)$, and that each production rule p applies to exactly one non-terminal. Thus, the time to compute $V(i, R, j)$ for all R and all $(i, j) \in States(R)$ is $O(Kn^2)$. Therefore, the time to compute the best neighbor generated by the grammar is $O(Kn^3)$.

When the TSP grammar is in extended normal form, all the production rules have at most one non-terminal on the right hand side. That is, p is either of the form $p : A \rightarrow \alpha B$ or $p : A \rightarrow B\alpha$ or $p : A \rightarrow \alpha$, where α is a string of terminals and B is a non-terminal. We analyze $p : A \rightarrow \alpha B$. Since $|States(\alpha)| = 1$, the bound

$$Time(p) = O(|States(\alpha)| \times |States(B)|) = O(n^2)$$

holds. Therefore, the time to compute the best neighbor generated by a grammar in extended normal form is $O(Kn^2)$.

When the TSP grammar is a left regular grammar such that $\sigma(n) = n$, then for any non-terminal R , $|States(R)| = O(n)$, and for any production rule p , $|States(p)| = O(n)$. The same bounds hold when G is a right regular grammar such that $\sigma(1) = 1$. We will prove the bound $O(Kn)$ for left regular grammars such that $\sigma(n) = n$. The proof is the same for right regular grammars such that $\sigma(1) = 1$. Given a production rule $p : A \rightarrow a_1 a_2 \dots a_r B$, the time to compute $V(i, p, n)$ is $O(|States(B)|) = O(n)$. Therefore, the time to compute $V(i, p, n)$ for all p and all $(i, n) \in States(p)$ (from previous values $V(j, B, n)$) is $O(Kn)$.

For all non-terminals R and for all $(i, n) \in States(R)$, the time to compute $V(i, R, n)$ is $O(Kn)$.

Therefore, the DP algorithm runs in $O(Kn)$ time when G is a left (right) regular grammar such that $\sigma(n) = n$ ($\sigma(1) = 1$). □

The following corollary matches the running time of the algorithm of Deĭneko and Woeginger ([47], Theorem 4) for permutation trees neighborhoods.

Corollary 4.4.4. *The generic dynamic programming algorithm of Section 4.4.1 searches a Tree-based grammar with at most F production rules per non-terminal in time $O(Fn^4)$.*

Proof. Let T be the permutation tree defined by the grammar. This tree has n leaves, one per each nonterminal, and it has at most $n-1$ internal nodes. Therefore, the number of nodes of T (that is, the number of non-terminals and terminals of G) is at most $2n-1 = O(n)$. Then the number of production rules is $K = O(Fn)$. By Theorem 4.4.3, the running time of the generic DP in a Tree-Based Grammar is $O(Kn^3) = O(Fn^4)$. □

The following corollary improves the running time of the algorithm of Deĭneko and Woeginger ([47], Theorem 6) for twisted neighborhoods by a factor of n .

Corollary 4.4.5. *The time to compute the best permutation in the Twisted Sequence Neighborhood is $O(n^6)$.*

Proof. The number of production rules of the Twisted Sequence Neighborhood Grammar is $K = O(n^3)$. By Theorem 4.4.3, the running time of the generic DP in this grammar is $O(Kn^3) = O(n^6)$. □

4.4.2 Incrementality

In neighborhoods for the symmetric TSP such as 2-opt, the neighbors are very similar to the current solution. In these neighborhoods, the cost of a neighbor can usually be computed from the cost of the current solution in constant time. Implementations of local search heuristics take this into account in order to save running time. Michel and Van Hentenryck [45] implement this idea in the context of a modeling language for local search heuristics. In this subsection we focus on how to compute efficiently the cost of production rules of the form $A \rightarrow a_1 \dots a_r$ or $A \rightarrow R_1 R_2$, where either R_1 or R_2 are strings of terminals. We restrict our analysis to the symmetric TSP.

Let π be the current solution. For each $2 \leq i \leq n$, we define

$$c_i = \sum_{k=1}^{i-1} c_{\pi(k)\pi(k+1)}.$$

We define $c_0 = c_1 = 0$. These numbers can be computed in $O(n)$ time at each iteration.

Two cities a, b are *consecutive* if $b = a + 1$ or $b = a - 1$. Similarly, a sequence a_1, \dots, a_r of cities is *consecutive* if $a_k = a_1 + k - 1$ for all $2 \leq k \leq r$. If the distance matrix is symmetric, we also consider a sequence a_1, \dots, a_r consecutive if $a_k = a_1 - k + 1$ for all $2 \leq k \leq r$. Let p be a production rule of the form $A \rightarrow a_1 \dots a_r$ or $A \rightarrow a_1 \dots a_r B$ or $A \rightarrow B a_1 \dots a_r$ where a_1, \dots, a_r are terminals and B is a non-terminal. To simplify notation, we assume that p is of the form $A \rightarrow a_1 \dots a_r$. We associate to this production rule a sequence of integers $0 = i_0^p < i_1^p < \dots < i_W^p = r$ so that

1. For any $0 \leq t < W$, the sequence $a_{i_t^p+1}, \dots, a_{i_{t+1}^p}$ is consecutive.
2. For any $0 \leq t < W$, the cities $a_{i_t^p}, a_{i_{t+1}^p}$ are not consecutive.

For example, the sequence associated with the production rule $A_{1,n} \rightarrow 1, n-1, n-2, \dots, 3, 2, n$ of 2-opt is $0, 1, n-1, n$. The sequence associated to each production rule can be computed only once, since it does not change at any iteration. The following lemma holds.

Lemma 4.4.6. *Given a production rule p of the form $A \rightarrow a_1 \dots a_r$, the value $V(a_1, p, a_r)$ as defined in the generic DP algorithm of Section 4.4.1 can be computed from c_1, \dots, c_n in $O(W)$ time, where $W + 1$ is the length of the sequence $0 = i_0^p < \dots < i_W^p = r$ associated to p as defined above.*

Proof. The cost $V(a_1, p, a_r) = \sum_{k=1}^{r-1} c_{\pi(k)\pi(k+1)}$ associated to the path $\pi(a_1), \dots, \pi(a_r)$ is equal to $\sum_{k=1}^W (c_{i_k^p} - c_{i_{k-1}^p+1}) + \sum_{k=1}^{W-1} c_{\pi(i_k^p)\pi(i_{k+1}^p)}$. This last expression can be computed in $O(W)$ time. \square

For example, the length of a sequence associated to any production rule of 2-opt is 4 and therefore the value $V(a_1, p, a_r)$ corresponding to a particular production rule p can be computed from $c(\pi)$ in constant time. The next corollary follows.

Corollary 4.4.7. *The value $V(a_1, p, a_r)$ of a production rule p of the Adjacent, 2-opt, Twisted or Dynasearch TSP Grammar of the form $A \rightarrow a_1 \dots a_r$ can be computed in $O(1)$ time.*

4.4.3 Running times for the Generic Algorithm on TSP grammars

In this subsection, we establish the running times for the generic dynamic programming algorithm when applied to the neighborhoods generated by the grammars in Sections 4.3.1 and 4.3.2. In most cases, the time bound is better than the naïve time bound of $O(Kn^3)$ as proved in the previous subsection.

Data structures for grammars in normal form

All the grammars defined in Sections 4.3.1 and 4.3.2 are in normal form; that is, all their production rules replace a non-terminal by at most two non-terminals. In this subsection, we present some data structures to store information related to the sets *States*. The recursion formulas of the DP solver can be computed more efficiently using these data structures.

The following data structures are written for grammars in normal form. For each non-terminal R , we store the set $States(R)$ as a list $L(R)$. Each $(i, j) \in L(R)$ has associated a list $L(i, R, j)$ with the states $\{i - p - j : p \text{ applies to } R \text{ and } (i, j) \in States(p)\}$. Similarly, for each

TSP Neighborhood Grammar	Running time
Adjacent Interchange Neighborhood	$O(n)$
2-opt Neighborhood	$O(n^2)$
Pyramidal Tour Neighborhood	$O(n^2)$
Tree-based grammars with at most F production rules per non-terminal	$O(Fn^4)$
Twisted Sequence Neighborhood	$O(n^6)$
Semi-Twisted Sequence Neighborhood	$O(n^3)$
Complete Neighborhood	$O(n^2 2^n)$
Balas-Simonetti Neighborhood	$O(k^2 2^k n)$
Enlarged Balas-Simonetti Neighborhood	$O(k^{1.5} 4^k n)$
Dynasearch Neighborhoods	$O(n^2)$
Extended Dynasearch Neighborhoods	$O(n^3)$

Table 4.1: Running times for the generic dynamic programming algorithm for TSP neighborhoods.

production rule p , we store $States(p)$ as a list $L(p)$. For each production rule $p : A \rightarrow R_1 R_2$ with two non-terminals on the right hand side and for each $(i, j) \in L(p)$ we associate a list $L(i, p, j)$ with the elements of $\{(i, s, t, j) : (i, s) \in States(R_1), (t, j) \in States(R_2)\}$.

These data structures don't change from iteration to iteration of the local search algorithm. Therefore they are computed once. They can be computed in $O(Kn^2)$, and the proof of this bound is similar to the proof of Proposition 4.4.1.

Proposition 4.4.8. *The running times of the generic dynamic programming algorithm of Section 4.4.1 in the Adjacent Interchange Neighborhood grammar and in the 2-exchange Neighborhood grammar defined in Section 4.3.2 is $O(n)$ and $O(n^2)$ respectively.*

Proof. For both sequence grammars, Corollary 4.4.7 says that $V(i, p, j)$ can be computed in constant time when p is of the form $A_{1,n} \rightarrow \alpha$ for a string α of terminals. In order to prove the proposition, it remains to bound the sets $States$.

We analyze the Adjacent Interchange Neighborhood grammar first. Since $States(R) \subseteq \{(k, l) : k \in \{1, 2\}, l \in \{n - 1, n\}\}$ for the Adjacent Interchange Neighborhood grammar, it follows that $|States(R)| \leq 4 = O(1)$ in this case. Therefore, the time to compute $V(i, R, j)$ for all $(i, j) \in L(R)$ is $O(1)$. The states of the production rule $p_0 : S \rightarrow A_{1,n}$ are the states of $A_{1,n}$ and therefore $|States(p_0)| \leq 4 = O(1)$. Except for $p_0 : S \rightarrow A_{1,n}$, all the production

rules of the Adjacent Interchange Neighborhood are of the form $A_{1,n} \rightarrow \alpha$ for some string α of terminals. It follows that $|States(p)| = O(1)$ for all the production rules of this TSP grammar. Therefore, the time to compute $V(i, p, j)$ for all $(i, j) \in L(p)$ is $O(1)$.

The total number of production rules is $O(n)$ and therefore the total time to compute $V(i, p, j)$ for all p , all $(i, j) \in L(p)$ is $O(n)$. There are two nonterminals, S and $A_{1,n}$. Each one has associated a list $L(R)$ of length $O(1)$. Each state $i - R - j$ has associated a list $L(i, R, j)$ of length $O(n)$. Then, the total time to compute $V(i, R, j)$ for all nonterminal R , all $(i, j) \in L(R)$ is $O(n)$.

We analyze the 2-exchange Neighborhood grammar.

Since $States(R) \subseteq \{(k, l) : \text{either } k = 1 \text{ or } l = n\}$, it follows that $|States(R)| = O(n)$ in this case. The time to compute $V(1, A_{1,n}, n)$ is $O(n^2)$. The time to compute $V(i, A_{1,n}, n)$ for all $i > 1$ is $O(n)$. The time to compute $V(1, A_{1,n}, j)$ for all $j < n$ is $O(n)$. The states of the production rule $p_0 : S \rightarrow A_{1,n}$ are the states of $A_{1,n}$ and therefore $|States(p_0)| = O(n)$. The time to compute $V(i, p_0, j)$ for all $(i, j) \in L(p)$ is $O(n^2)$. Except for $p_0 : S \rightarrow A_{1,n}$, all the production rules of the Adjacent Interchange Neighborhood are of the form $A_{1,n} \rightarrow \alpha$ for some string α . It follows that $|States(p)| = O(1)$ for all the production rules but p_0 of this TSP grammar. Therefore, the time to compute $V(i, p, j)$ for all $(i, j) \in L(p)$ is $O(1)$.

The total number of production rules is $O(n^2)$ and therefore the total time to compute $V(i, p, j)$ for all p , all $(i, j) \in L(p)$ is $O(n^2)$. Then, the total time to compute $V(i, R, j)$ for all nonterminal R , all $(i, j) \in L(R)$ is $O(n^2)$. Therefore the generic DP algorithm runs in $O(n^2)$ time in the 2-exchange Neighborhood grammar. \square

Proposition 4.4.9. *The running times of the generic dynamic programming algorithm of section 4.4.1 in the Dynasearch Neighborhood grammars defined in Section 4.3.2 is $O(n^2)$.*

Proof. Given a nonterminal R of the Dynasearch Neighborhood grammars, there is only one pair of states $(i, j) \in States(R)$. For any nonterminal of the form $A_{1,j}$ the length of the list $L(1, A_{1,j}, j)$ is $O(n)$. There are $O(n)$ nonterminals of this form and therefore the total time to compute $V(1, A_{1,j}, j)$ for all j is $O(n^2)$. For any nonterminal of the form $A_{i,j} : i \neq 1$, the length of the list $L(i, A_{i,j}, j)$ is $O(1)$. There are $O(n^2)$ nonterminals of this form and

therefore the total time to compute $V(i, A_{i,j}, j)$ for all $1 < i < j$ is $O(n^2)$.

All the production rules of the form $A_{i,j} \rightarrow \alpha$ for some string α satisfies that $|States(p)| = O(1)$. There are $O(n^2)$ production rules of this form and therefore the total time to compute $V(i, p, j)$ for all p of this form, for all $(i, j) \in L(p)$ is $O(n^2)$.

All the production rules of the form $A_{1,j} \rightarrow A_{1,i-1}, A_{i,j}$ satisfies that $|States(p)| = O(1)$ and that $L(1, p, j) = \{(1, i-1, i, j)\} = O(1)$. There are $O(n^2)$ production rules of this form and therefore the total time to compute $V(i, p, j)$ for all p of this form, for all $(i, j) \in L(p)$ is $O(n)$. \square

Proposition 4.4.10. *The running time of the generic dynamic programming algorithm of Section 4.4.1 in the Pyramidal Tour Neighborhood or the Semi-Twisted Sequence Neighborhood given as in Section 4.3.2 is $O(n^2)$ and $O(n^3)$ respectively.*

Proof. We prove the proposition for the Pyramidal Tour Neighborhood case since the proof for the Semi-Twisted Sequence Neighborhood is similar. The difference between the running times of these two grammars follows from the fact that the number of production rules of the Semi-Twisted Sequence Neighborhood is $O(n^2)$ while the number of production rules of the Pyramidal Tour Neighborhood is $O(n)$.

Given a non-terminal $A_{j,n}$, we observe that any path generated by the subgrammar $G_{A_{j,n}}$ satisfies that either it starts or it ends in city j . Therefore, $States(A_{j,n}) \subseteq \{(k, l) : \text{either } k = j \text{ or } l = j\}$, that is, $States(A_{j,n}) = O(n)$. The number of production rules that can be applied to a nonterminal is at most two. Therefore, the time to compute $V(i, R, j)$ for all $(i, j) \in L(R)$ is $O(n)$. Given a production rule p , say $p : A_{j,n} \rightarrow A_{j+1,n}, j$, the lists $L(k, p, l)$ which are nonempty are of the form $L(k, p, j)$. When $k \neq j+1$, there is only one element in $L(k, p, j)$, namely $(i, j+1, j, j)$. Therefore, we can compute $V(i, p, j)$ in $O(1)$ steps. The total time to compute $L(k, p, j)$ for all $k \neq j+1$ is then $O(n)$. When $k = j+1$, the number of elements of $L(j+1, p, j)$ is $O(n)$ since $L(j+1, p, j) = \{(j+1, s, j, j) : j+2 \leq s \leq n\}$. In this case we compute $V(j+1, p, j)$ in $O(n)$ steps. The total time to compute $V(i, p, j)$ for all $(i, j) \in L(p)$ is $O(n)$ then. The total number of production rules and nonterminals is $O(n)$ and therefore the total time to run the generic dynamic programming algorithm for

the Pyramidal Tour Neighborhood grammar of Section 4.3.2 is $O(n^2)$. □

The following proposition shows that the generic DP solver for TSP grammar neighborhoods has the same running times as the solver proposed for the complete neighborhood, on this neighborhood, and as the solvers proposed for Balas-Simonetti Neighborhood and the Enlarged Balas-Simonetti Neighborhood, on these neighborhoods. We also compute the running time over the inverse Balas-Simonetti neighborhood, as defined in Section 4.6.1.

Proposition 4.4.11. *The generic dynamic programming algorithm of Section 4.4.1 searches the Complete Neighborhood as defined in Section 4.3.2 in time $O(2^n n^2)$. Its running times in the Balas-Simonetti Neighborhood grammar, in the Enlarged Balas-Simonetti Neighborhood grammar and in the Inverse Balas-Simonetti Neighborhood grammar as given in Section 4.3.2 is $O(k^2 2^k n)$, $O(k^{1.5} 4^k n)$ and $O(k^2 4^k k! n)$ respectively.*

Proof. All these grammars are right regular grammars and all the tours in the neighborhood they generate satisfy that $\sigma(1) = 1$. Therefore, the DP solver runs in $O(Kn)$ in all these neighborhoods, as proven in Theorem 4.4.3. This bound gives the $O(2^n n^2)$ time for the Complete Neighborhood grammar, since the number of production rules of this grammar is $K = O(2^n n)$.

For the Balas-Simonetti (BS) Neighborhood grammar, the Enlarged Balas-Simonetti (EBS) Neighborhood grammar, and the Inverse Balas-Simonetti (IBS) Neighborhood grammar, we apply the following variation of this bound. Let G be a right regular grammar such that all the tours in the neighborhood its generates satisfy that $\sigma(1) = 1$. Let $s = \max\{|States(p)|, |States(R)|\}$ be the maximum number of states of a production rule or a non-terminal of G . Then, the DP solver runs in $O(Ks)$ time.

Given a non-terminal R of the Balas-Simonetti Neighborhood grammar, the Enlarged Balas-Simonetti Neighborhood grammar, or the Inverse Balas-Simonetti Neighborhood grammar, we claim that $|States(R)| = O(k)$. This claim easily follows from two properties shared by these neighborhoods: any sequence σ (in any of these neighborhoods) satisfies that $\sigma(1) = 1$, and any object placed in the m th place belongs to the set $\{m - k, \dots, m + k\}$.

A sharper bound holds for the number of states associated with a production rule of these grammars: $|States(p)| = 1$.

Let K_{BS} , K_{EBS} be the number of production rules of the Balas-Simonetti Neighborhood grammar and the Enlarged Balas-Simonetti Neighborhood grammar respectively. As we proved in Section 4.3.2, $K_{BS} = O(k2^k n)$ and $K_{EBS} = O(k^{0.5}4^k n)$. These bounds on the number of production rules and the $O(Ks)$ time bound (where $s = O(k)$) on the running time of the DP solver prove the claim. \square

4.4.4 A generic solution procedure for Linear Ordering Grammars

A sequence grammar defines valid permutations for the Linear Ordering Problem. In this subsection we present a Dynamic Programming algorithm that finds the best permutation in a neighborhood generated by a linear ordering grammar.

The initial permutation is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. We introduce some additional notation. We denote by $f(i, j)$ the cost $c_{\pi(i), \pi(j)}$. Given a terminal or non-terminal R , we denote by $V(R)$ the value $\min\{\sum_{i < j} f(\sigma(i), \sigma(j)) : \sigma \in L(G_R)\}$ that corresponds to the best ordering of elements in R as generated by the grammar. The cost of the best sequence in the neighborhood is $V(S)$. Similarly, given a production rule $p : A \rightarrow R_1 R_2 \dots R_r$ we denote by $V(p)$ the cost of the best ordering of elements in R as generated by the grammar $G_{p(R)}$. The following is a dynamic programming recursion for computing the best tour in a Neighborhood grammar.

1. If a is a terminal then $V(a) = 0$.
2. Suppose p is the production rule $A \rightarrow R_1, R_2, \dots, R_r$.
Then, $V(p) = \sum_{1 \leq t_1 < t_2 \leq r} \sum_{i \in R_{t_1}, j \in R_{t_2}} f(i, j) + \sum_{k=1}^r V(R_k)$

3. If R is a non-terminal, then
 $V(R) = \min\{V(p) : p \text{ is a production rule and } p \text{ applies to } R\}$.

Theorem 4.4.12. *Let K be the number of production rules of a neighborhood grammar for a linear ordering problem with n elements. Then the time to compute the best neighbor generated by the grammar is $O(Kn^2)$.*

Proof. The time needed to compute $c(R)$ after we compute $c(p)$ for all production rules p that apply to R is proportional to the number of rules that apply to R . Since each production rule applies to at most one non-terminal, the total time to compute $c(R)$ for all R is $O(K)$.

For each production rule $p : A \rightarrow R_1, R_2, \dots, R_r$, the total time needed to compute $V(p)$ after we compute $V(R_1), \dots, V(R_r)$ is at most $O(|A|^2) = O(n^2)$. Therefore the total time to compute $V(p)$ for all p is $O(Kn^2)$. \square

4.4.5 A generic solution procedure for Minimum Latency Grammars

We present a Dynamic Programming algorithm for finding the best solution in a grammar neighborhood of the Minimum Latency Problem.

The initial tour is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. We denote by $f(i, j)$ the arc cost $c_{\pi(i), \pi(j)}$. We introduce some additional notation. For a given terminal or non-terminal R , let $States(R)$ be the set of pairs of customers (i, j) such that there is a path generated by the grammar G_R from customer i to customer j that passes through all the customers of $Customers(R)$. We also define the set $InitCustomers(R) = \{i : (i, j) \in States(R) \text{ for some } j\}$, and the set $EndCustomers(R) = \{j : (i, j) \in States(R) \text{ for some } i\}$.

We define the sets $States(p), InitCustomers(p), EndCustomers(p)$ for a production rule p in the same manner. These sets are defined in the same way as the corresponding sets of the DP algorithm for the TSP. Therefore, they satisfy the same recursive relations, and the time to compute them is $O(Kn^2)$, as Proposition 4.4.1 shows.

Given a subsequence of customers a_b, \dots, a_e located in positions $b, b+1, \dots, e$, we denote by $V(b, a_b, \dots, a_e) = \sum_{k=b}^e (n-k+1)c_{a_k, a_{k+1}}$ the latency cost associated to it. Given a terminal or non-terminal R , customers $i, j \in Customers(R)$, and an initial position $1 \leq b \leq n$, we denote by $V(b, i, R, j)$ the minimum latency cost of a path that starts from $\pi(i)$ in

position b , to city $\pi(j)$ passing through all the customers of $\pi(\text{Customers}(R))$ as generated by the grammar. Similarly, given a production rule $p : A \rightarrow R_1 R_2 \dots R_r$, customers $i, j \in \text{Customers}(A)$, and an initial position $1 \leq b \leq n$, we denote by $V(b, i, p, j)$ the minimum cost path, starting from city $\pi(i)$ in position b to city $\pi(j)$, passing through all the customers of $\pi(\text{Customers}(A))$ as generated by p .

We refer to a 4-tuple $b - i - R - j$ as a state of the dynamic programming recursion. The best tour generated by the grammar has value $\min\{V(1, i, S, j) : i, j \in \text{States}(S)\}$. The following is a dynamic programming recursion for computing the best tour in a Minimum Latency Problem Neighborhood grammar.

1. If a is a terminal with $\text{Customers}(a) = k$, then for $1 \leq b \leq n : V(b, k, a, k) = 0$.

2. Suppose p is the production rule $A \rightarrow a_1 a_2 \dots a_r B$. Then,

For $(a_1, j) \in \text{States}(p)$, $1 \leq b \leq n :$

$$V(b, a_1, p, j) = \min\{\sum_{k=1}^{r-1} (n+1-b-k)f(a_k, a_{k+1}) + (n+1-b-r)f(a_r, t) + V(b+r, t, B, j) : t \in \text{InitCustomers}(B)\}.$$

3. Suppose p is the production rule $A \rightarrow R_1, R_2, \dots, R_r$.

Then, for $2 \leq k \leq r$, for $i \in \text{InitCustomers}(R_1), j \in \text{EndCustomers}(R_k)$, and for $1 \leq b \leq n$:

$$V(b, i, R_1, R_2, \dots, R_k, j) = \min\{\bar{V}(b, i, R_1, R_2, \dots, R_{k-1}, t) + V(b + \sum_{l=1}^{k-1} |R_l|, t, R_k, j) : t \in \text{InitCustomers}(R_k)\}.$$

For $1 \leq k \leq r-1$, for $i \in \text{InitCustomers}(R_1), j \in \text{IniCustomers}(R_{k+1})$, and for $1 \leq b \leq n$:

$$\bar{V}(b, i, R_1, R_2, \dots, R_k, j) = \min\{V(b, i, R_1, R_2, \dots, R_k, s) + (n-b - \sum_{l=1}^k |R_l| + 1)f(s, j) : s \in \text{EndCustomers}(R_k)\}.$$

For $(i, j) \in \text{States}(p)$ and for $1 \leq b \leq n :$

$$V(b, i, p, j) = V(b, i, R_1, R_2, \dots, R_r, j)$$

4. If R is a non-terminal, then

For $(i, j) \in \text{States}(R)$ and for $1 \leq b \leq n$:

$$V(b, i, R, j) = \min\{V(b, i, p, j) : p \text{ is a production rule and } p \text{ applies to } R\}.$$

Theorem 4.4.13. *Let K be the number of production rules of a Latency Problem neighborhood grammar for a problem with n customers. Then the time to compute the best neighbor generated by the grammar is $O(Kn^4)$. The running time on Left Regular Grammars is $O(Kn^2)$.*

Proof. The DP procedure is the same as the one derived for TSP grammars, except for the extra parameter b in the DP recursion. The running time of the DP procedure for a context-free MLP grammar is n times the corresponding running time of the DP procedure for a sequence context-free grammar because of this extra parameter.

When G is a Left Regular MLP Grammar we do not need this parameter since b , the initial position of a subsequence generated by a production rule p or from a non-terminal R , is always the same. Therefore the running time of the DP procedure for Left Regular MLP Grammars is the same as the running time of the DP procedure for a Left Regular Grammars for TSP, which is $O(Kn^2)$. \square

4.4.6 A generic solution procedure for Scheduling Grammars

We analyze the single machine scheduling problem with cost the sum of weighted tardinesses, although the results also apply for different versions of Single Machine Problems. For this problem, the DP algorithm we present is pseudopolynomial for sequence context-free grammars, and is polynomial for left regular grammars.

The initial permutation is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. Let $P = \sum_{j=1}^n p_j$ be the time to completion. We introduce some additional notation. Given a terminal or non terminal R , let $t(R) = \sum_{j \in Jobs(R)} p_{\pi(j)}$. Given a production rule p that applies to R , let $t(p) = t(R)$. For each terminal or non terminal R , and each time $0 \leq t \leq P$, let $V(t, R)$ denote the sum of tardinesses of jobs in $Jobs(R)$, assuming that the first job of R is scheduled to start at time t , that corresponds to the best ordering of jobs in R as generated by the grammar G_R . That

is,

$$V(t, R) = \min\left\{ \sum_{k=1}^{|Jobs(R)|} \max\{t + p_{\pi(\sigma(k))} - d_{\pi(\sigma(k))}; 0\} : \sigma \in L(G_R) \right\}. \quad (4.8)$$

The optimal sequence $\pi\sigma \in N(\pi)$ has cost $V(0, S)$. Similarly, given a production rule p , and a time $0 \leq t \leq P$, we denote by $V(t, p)$ the cost of the best ordering of elements in R as generated by the grammar G_p , assuming that the first job of R is scheduled to start at time t . The following is a dynamic programming recursion for computing the best tour in the neighborhood of π generated by the context free sequence grammar G .

1. If a is a terminal with $Jobs(a) = l$, then for each $0 \leq t \leq P$,

$$V(t, a) = \max\{t + p_{\pi(l)} - d_{\pi(l)}; 0\}.$$
2. Suppose p is a production rule of the form $A \rightarrow R_1, R_2, \dots, R_r$. Then, for each $0 \leq t \leq P$,

$$V(t, p) = \sum_{k=1}^r V(t + \sum_{m=1}^{k-1} t(R_m), R_k)$$
3. If R is a non-terminal, then for each $0 \leq t \leq P$,

$$V(t, R) = \min\{V(t, p) : p \text{ is a production rule and } p \text{ applies to } R\}.$$

The following theorem holds.

Theorem 4.4.14. *Let K be the number of production rules of a context-free neighborhood grammar for a single machine problem with n jobs. Let $P = \sum_{i=1}^n p_i$ be the total processing time. Then the time to compute the best neighbor generated by the context-free grammar is $O(KPn)$.*

Proof. There are $O(KP)$ states of type $t - p$, and it takes $O(n)$ time to compute each $V(t, p)$. Therefore, the total time to compute $V(t, p)$ is $O(KPn)$.

There are $O(KP)$ states of type $t - R$, and it takes $O(KP)$ time to compute them all.

Therefore, the total time is $O(KPn)$. □

This running time is pseudopolynomial. When G is a left regular grammar, the DP algorithm described before can be adapted to find the optimal solution in polynomial time.

The states $t - R$ (or $t - p$) of the previous DP algorithm have a parameter $0 \leq t \leq P$, which corresponds to the earliest time available for schedule a job of $Jobs(R)$. When G is a left regular grammar, we can restrict the DP algorithm to compute $V(P - t(R), R)$ and $V(P - t(p), p)$ for all R and all production rules p . This is so since any sequence $\sigma \in L(G)$ that is derived using a production rule p that applies to R satisfies that the earliest time for scheduling the jobs of R is $\sum_{j \in Jobs(R)} p_{\pi(j)} = P - t(R)$. The following theorem holds

Theorem 4.4.15. *Let K be the number of production rules of a left regular neighborhood grammar for a single machine problem with n jobs. Then the time to compute the best neighbor generated by the left regular grammar is $O(Kn)$.*

Proof. There are $O(K)$ states of type p , and it takes $O(n)$ time to compute each $V(p)$. Therefore, the total time to compute $V(p)$ is $O(Kn)$.

There are $O(K)$ states of type R , and it takes $O(K)$ time to compute them all.

Therefore, the total time is $O(Kn)$. □

Lawler's neighborhood for minimizing the total tardiness

Lawler [41] proved that the single machine scheduling problem with cost the sum of weighted tardinesses can be solved in pseudopolynomial time when the weighting of jobs is *agreeable*, in the sense that $p_i < p_j$ implies $w_i \geq w_j$. Lawler's algorithm runs in $O(n^4 P)$ time, where $P = \sum_{i=1}^n p_i$. In this section we restate his algorithm as finding the best solution in a grammar neighborhood of a particular initial sequence.

Assume the jobs are ordered by increasing order of due dates, that is, $d_i < d_j$ whenever $i < j$. For $1 \leq i \leq k \leq j \leq n$, we define the following subset of jobs $S(i, j, k) = \{q : i \leq q \leq j \text{ and } p_q < p_k\}$. There are $O(n^3)$ subsets $S(i, j, k)$. We define the set $\mathcal{S}_L = \{S(i, j, k) : 1 \leq i \leq k \leq j \leq n\}$. Then, $|\mathcal{S}_L| = O(n^3)$. The Lawler's neighborhood grammar is as follows.

Lawler's Neighborhood Grammar

$$S \rightarrow A_{S(1,k+s,k)}, k, A_{S(k+s+1,j,k)}$$

for $1 \leq k \leq n$ such that $p_k = \max\{p_q : 1 \leq q \leq n\}$

and for each $0 \leq s \leq j - k$.

$$A_{S(i,j,k)} \rightarrow A_{S(i,k'+s,k')}, k', A_{S(k'+s+1,j,k')}$$

for $k' \in S(i,j,k)$ such that $p_{k'} = \max\{p_q : q \in S(i,j,k)\}$

and for each $0 \leq s \leq j - k'$.

$$A_{S(i,i,k)} \rightarrow i \text{ for all nonempty } S(i,i,k) \in \mathcal{S}_L.$$

The value of the function $Jobs$ on each terminal and non-terminal is straightforward: $Jobs(S) = \{1, \dots, n\}$, $Jobs(A_{S(i,j,k)}) = S(i,j,k)$ for each subset $S(i,j,k) \in \mathcal{S}_L$.

Lawler [41] proved that there exists an optimal sequence that belongs to Lawler's neighborhood of $\pi = (1, \dots, n)$, the sequence of jobs ordered by their due dates. That is, the local optimum in $N_{\mathcal{L}}(\pi)$ is a global optimum.

Let us bound the total number of production rules of Lawler's neighborhood. For each $S(i,j,k) \in \mathcal{S}_L$ there are $O(n)$ production rules that apply to the nonterminal $A_{S(i,j,k)}$. Since $|\mathcal{S}_L| = O(n^3)$, the total number of production rules of Lawler's neighborhood is $O(n^4)$. The generic DP algorithm for Scheduling Neighborhoods, when applied to Lawler's Neighborhood Grammar, has the same running time as Lawler's DP algorithm as the next proposition shows.

Proposition 4.4.16. *The generic DP algorithm for optimizing over a Scheduling Neighborhood with n jobs and $P = \sum_{i=1}^n p_i$ total processing time runs in $O(n^4P)$ time when the neighborhood is Lawler's neighborhood.*

Proof. The bound obtained by applying Theorem 4.4.14 is $O(n^5P)$ time. However, by going over it carefully, it can be improved by a factor of n .

The number of production rules is $K = O(n^4)$. There are $O(KP) = O(n^4P)$ states of type $t-p$, and it takes $O(1)$ time to compute each $V(t,p)$ (this is the main difference with the bound derived in Theorem 4.4.14). Therefore, the total time to compute $V(t,p)$ is $O(n^4P)$.

There are $O(n^4P)$ states of type $t - R$, and it takes $O(n^4P)$ time to compute them all. Therefore, the total time is $O(n^4P)$. \square

4.4.7 A generic solution procedure for Weighted Bipartite Matching Problem with Side Constraint Grammars

Greedy Solution for the Weighted Bipartite Matching Problem with Side Constraints

Let $\pi = (\pi(1), \dots, \pi(n)) \in S_n$ be a permutation of the lessons. We assign to this permutation the optimal matching M_π for WBMPSC such that lesson $\pi(i)$ is assigned to earlier time spot(s) than lesson $\pi(j)$ whenever $i < j$. We use a simple DP algorithm to find matching M_π .

1. For any $1 \leq i \leq n$, $1 \leq b < e \leq m$:

$$V(b, \pi(i), e) = \min\{c_{\pi(i),t} : 1 \leq t \leq e - b_{\pi(i)}\}$$

2. For $1 \leq e \leq m$ and for $k = 2$ to n :

$$V(1, \pi(1), \dots, \pi(k), e) = \min\{V(1, \pi(1), \dots, \pi(k-1), t) + V(t+1, \pi(k), e) : 1 \leq t < e\}$$

$V(1, \pi(1), \dots, \pi(n), m)$ is the cost of the matching assigned to π .

Proposition 4.4.17. *Given a weighted matching problem with relations with n lessons and m time slots, the running time of the DP algorithm is $O(nm^3)$.*

Proof. The number of states $b - \pi(i) - e$ is $O(nm^2)$. For each of these states, the time to compute $V(b, \pi(i), e)$ is $O(m)$. Therefore, the total time to compute $V(b, \pi(i), e)$ for all states $b - \pi(i) - e$ is $O(nm^3)$.

The number of states $1 - \pi(1), \dots, \pi(k) - e$ is $O(nm)$. For each of these states, the time to compute $V(1, \pi(1), \dots, \pi(k), e)$ is $O(m)$. Therefore, the total time to compute $V(1, \pi(1), \dots, \pi(k), e)$ for all states $1 - \pi(1), \dots, \pi(k) - e$ is $O(nm^2)$.

Therefore, the total time of the DP is $O(nm^3)$. \square

Every solution associated to a sequence of lessons is a feasible solution for the WBMPSC. The following proposition shows that the sequencing problem for the WBMPSC is equivalent to the WBMPSC.

Proposition 4.4.18. *There exists a permutation π such that its associated matching M_π is an optimal solution of the WBMPSC.*

Proof. Given an optimal solution OPT of the WBMPSC, it defines a sequence of lessons π_{OPT} as follows. The first lesson, $\pi_{\text{OPT}}(1)$, is the lesson assigned to the earliest slot time(s) (among all lessons) according to OPT, $\pi_{\text{OPT}}(2)$ is the lesson assigned to the second earliest slot time(s) according to OPT, and so on. It is easy to see that the solution associated to π_{OPT} via the greedy DP algorithm is OPT (or a solution with the same cost). Therefore, the proposition holds. \square

Dynamic Programming Procedure for Grammars for WBMPSC

The initial permutation of lessons is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. We introduce some additional notation. Let $States = \{(b, e) : 1 \leq b < e \leq m\}$ be the set of ordered pairs of time slots. Given a terminal or non terminal R , and a pair $(b, e) \in States(R)$, we denote by $V(b, R, e)$ the minimum cost that corresponds to the best sub-matching of lessons of $\pi(R)$ on the time period $[b, e]$ as generated by the grammar. Similarly, given a production rule $p : A \rightarrow R_1 R_2 \dots R_r$ and a pair $(b, e) \in States(R)$, we denote by $V(b, p, e)$ the minimum cost that corresponds to the best sub-matching of lessons of $\pi(R)$ on the time period $[b, e]$ as generated by the grammar.

We refer to a triple $b - R - e$ or $b - p - e$ as a state of the dynamic programming recursion. The best matching generated by the grammar neighborhood of π has value $\{V(1, S, m)\}$. The following is a dynamic programming recursion for computing the best tour in a Matching Neighborhood of π .

1. If a is a terminal with $Lessons(a) = l$, then

For any $1 \leq b < e \leq m$:

$$V(b, a, e) = \min\{c_{\pi(l), t} : 1 \leq t \leq e - b_{\pi(l)}\}$$

2. Suppose p is the production rule $A \rightarrow R_1, R_2, \dots, R_r$.

Then, for $k = 2$ to r and for $(b, e) \in \text{States}$:

$$V(b, R_1, R_2, \dots, R_k, e) = \min\{V(b, R_1, R_2, \dots, R_{k-1}, t) + V(t + 1, R_k, e) : b \leq t \leq e\}$$

Then, $V(b, p, e) = V(b, R_1 R_2 \dots R_r, e)$.

3. If R is a non-terminal, then

$$V(b, R, e) = \min\{V(b, p, e) : p \text{ is a production rule and } p \text{ applies to } R\}.$$

Theorem 4.4.19. *Let K be the number of production rules of a Matching neighborhood grammar for a weighted matching problem with side constraints, with n lessons and m time slots. Then the time to compute the best neighbor generated by the grammar is $O(Knm^3)$.*

Proof. The number of states $b - a - e$ is $O(nm^2)$. For each of these states, the time to compute $V(b, a, e)$ is $O(m)$. Therefore, the total time to compute $V(b, a, e)$ for all states $b - a - e$ is $O(nm^3)$.

For each production rule $p : A \rightarrow R_1, R_2, \dots, R_r$, the The number of states $b - p - e$ is $O(Km^2)$. For each of these states, the time to compute $V(b, R_1, \dots, R_k, e)$ for all $1 \leq k \leq r$ from the values $V(b', R_i, e')$ is $O(nm)$. Therefore, the total time to compute $V(b, p, e)$ for all states $b - p - e$ is $O(Knm^3)$.

For all non-terminals, the total time to compute $b - R - e$ is $O(Km^2)$. Therefore, the total time of the DP is $O(Knm^3)$. \square

4.4.8 The Shortest Hyperpath Problem

We show that the Shortest Hyperpath Problem can be restated as a sequencing problem. In the next subsection, we assign a hyperpath to a sequence of nodes.

Greedy Solution for the Shortest Hyperpath Problem

Let $\pi = (\pi(1), \dots, \pi(n)) \in S_n$ be a permutation of nodes. We use a simple greedy procedure to assign a hyperpath H_π to π . Let $\pi^{-1}(t)$ be the placement of node t in the sequence π . In some sense, we only care about the first $\pi^{-1}(t)$ nodes of the sequence. For any

$1 < i \leq \pi^{-1}(t)$, let $e_{\pi(i)} = (T(e_{\pi(i)}), h(e_{\pi(i)}))$ be an hyperarc with minimum cost among the hyperarcs $e = (T(e), h(e))$ such that $T(e) \subseteq \{\pi(1), \dots, \pi(i-1)\}$ and $h(e) = \pi(i)$. Let $A_{H_\pi} = \cup_{i=2}^{\pi^{-1}(t)} \{e_{\pi(i)}\}$. Let $V_\pi = \{\pi(1), \dots, t\}$. The hyperpath H_π is then (V_π, A_{H_π}) .

Proposition 4.4.20. *Given a hypergraph with n nodes and m hyperarcs, and a sequence $\pi \in S_n$, the greedy solution H_π can be computed in $O(\sum_{e \in A} |T(e)|)$ time.*

Proof. We associate to each hyperarc e the number $i(e) = \max\{i : \pi(i) \in T(e)\}$. The computation of these numbers takes $O(\sum_{e \in A} |T(e)|) = O(m\bar{T})$ time, where $\bar{T} = \frac{1}{m}(\sum_{e \in A} |T(e)|)$. We sort the m hyperarcs e according to its number $i(e)$. This operation takes $O(m \log m)$ time. For each node i , we construct a set of hyperarcs $A(i) = \{e : i(e) = i\}$. This takes $O(m)$ time. Finally, $e_{\pi(i)}$ is the hyperarc in $A(i)$ with minimum cost. This computation also takes $O(m)$ time. Therefore, the greedy solution can be computed in $O(m\bar{T})$ time. \square

The following proposition has a similar proof as Proposition 4.4.18.

Proposition 4.4.21. *There exists a permutation π such that its associated hyperpath H_π is an optimal solution of the minimum hyperpath problem.*

DP algorithm for Left Regular Grammars for SHP

We present an efficient DP algorithm for neighborhoods for the SHP defined by left regular grammars. We observe that context-free-grammar-based neighborhoods may not have an efficient algorithm. The initial permutation is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. Given a terminal or non terminal R , we denote by $V(R)$ the minimum cost that corresponds to the best greedy solution associated to a subsequence $\pi(\sigma)$, where σ is in the language $L(R)$. Similarly, given a production rule $p : A \rightarrow a_1 \dots a_r B$ we denote by $V(p)$ the minimum cost that corresponds to the best sub-matching of lessons of $\pi(R)$ as generated by the grammar using p .

The best hyperpath in the grammar neighborhood of π has value $V(S)$. The following is a dynamic programming recursion for computing the best tour in a Neighborhood of π .

1. If a is a terminal with $Nodes(a) = l$, then

$$V(a) = \min\{c_e : T(e) \subseteq N - \{l\}, h(e) = l\}.$$

2. Suppose p is the production rule $A \rightarrow a_1 a_2 \dots a_r B$.

Then, $V(p) = \sum_{k=1}^r \min\{c_e : T(e) \subseteq N - (\{a_{k+1}, \dots, a_r\} \cup \text{Nodes}(B)), h(e) = a_k\} + V(B)$.

3. If R is a non-terminal, then

$V(R) = \min\{V(p) : p \text{ is a production rule and } p \text{ applies to } R\}$.

Theorem 4.4.22. *Let K be the number of production rules of a Left Regular Grammar for a Shortest Hyperpath Problem with n nodes and m hyperarcs. Then the time to compute the best neighbor generated by the grammar is $O(Knm)$.*

Proof. Step 1 takes $O(Km)$ for all terminals. Step 3 takes $O(K)$ for all non-terminals. Step 2 takes $O(Knm)$ for all production rules. \square

The following table depicts the running times of the DP algorithms for optimizing over grammar neighborhood for different sequencing problems.

Problem	n	m	Greedy	DP for CFG	DP for LRG
TSP	cities	-	$O(n)$	$O(Kn^3)$	$O(Kn^2)$
LO	objects	-	$O(n^2)$	$O(Kn^2)$	$O(Kn^2)$
MLP	cities	-	$O(n)$	$O(Kn^4)$	$O(Kn^2)$
Single Machine Scheduling	jobs	-	$O(n)$	$O(KPn)$	$O(Kn)$
WBMPSC	Lessons	Time slots	$O(nm^3)$	$O(Knm^3)$	$O(Knm^2)$
SHP	nodes	hyperarcs	$O(nm)$	-	$O(Knm)$

4.4.9 Determining whether a permutation σ is in a sequence grammar

In this subsection, we address the question of whether a permutation is generated by a sequence grammar. The membership question of a string of length n to a language generated by a context-free grammar in normal form is solved by the Cocke-Younger-Kasami (CYK) algorithm (see Hopcroft and Ullman [37]). In their analysis, the size of the string can be arbitrarily large whereas the size of G is treated as a constant. The running time of the

CYK algorithm as a function of n and K is $O(Kn^3)$. We give a faster algorithm for the membership problem for sequence grammars in this subsection.

Given a terminal or non terminal R and a number $0 \leq q \leq n - 1$, we define $V(R, q)$ as 0 if there exists a sequence $\sigma_R \in L(G_R)$ such that $\sigma_R(t) = \sigma(t+q)$ for all $1 \leq t \leq |Objects(R)|$, and 1 otherwise. If $V(R, q) = 0$ for some q , we define $Position(R) = q$. If $V(R, q) = 1$ for all q , then we define $Position(R) = 0$. We define $V(p, q)$ and $Position(p)$ for a production rule p in a similar way.

The sequence σ belongs to $L(G)$ if and only if $V(S, 0) = 0$. The following is a dynamic programming recursion for computing $V(S, 0)$.

1. If a is a terminal with $Objects(a) = \{k\}$, then

$$Position(a) = \sigma^{-1}(k) \text{ and } V(a, Position(a)) = 0.$$

2. Suppose p is the production rule $A \rightarrow R_1, R_2, \dots, R_r$. Then,

$$Position(p) = Position(R_1) \text{ and}$$

$$V(p, Position(p)) = \min\{1; \sum_{1 \leq t \leq r} V(R_t, Position(R_1) + \sum_{1 \leq u \leq t-1} |Objects(R_u)|)\}$$

3. If R is a non-terminal, then

If there exists a production rule p that applies to R and such that $V(p, Position(p)) = 0$ then $Position(R) = Position(p)$ and $V(R, Position(R)) = 0$, otherwise $Position(R) = 0$ and $V(R, Position(R)) = 1$.

The following theorem holds. It has a similar proof as Theorem 4.4.3.

Theorem 4.4.23. *This algorithm determines whether a permutation σ is generated by a sequence grammar $(G, Objects)$ with K rules and n objects in $O(Kn)$ time.*

4.4.10 Context-Sensitive Sequence Grammars for the TSP

A generic solution procedure using dynamic programming

In this subsection we generalize the generic algorithm for finding the optimal solution on a grammar-based neighborhood for the TSP developed in Section 4.4.1 to deal with a class of

context-sensitive sequence grammars, namely when rules of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ for $\alpha_1, \alpha_2, \beta \in V^*$ are restricted to $\alpha_1, \alpha_2 \in V_T^*$.

The initial tour is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. We denote by $f(i, j)$ the arc cost $c_{\pi(i), \pi(j)}$. The generic algorithm for context-free sequence grammars for the TSP has states of the form (i, R, j) or (i, p, j) for terminals or non-terminals R , production rules p and cities i, j . The algorithm for context-sensitive grammars needs bigger states since a production rule p can be applied to a non-terminal R depending on the strings that surround R . Let C be the set of “contexts” of production rules of G , that is, $C = \{(\alpha, \beta) \in V_T^* \times V_T^* : \exists p \in P \text{ of the form } p : \alpha A \beta \rightarrow \alpha R_1 \dots R_r \beta\}$. The cardinality of C is at most K , the number of production rules. If G is a context-free grammar, then $C = \emptyset$. Given a string $\alpha \in V_T^*$, we define $InitObject(\alpha)$ ($EndObject(\alpha)$) as the first (last) object in α .

Let W be the set of substrings of either α or β for any $(\alpha, \beta) \in C$. We remark that the empty substring belongs to W . Given a terminal or non terminal R , strings $\alpha_E, \beta_E \in W, \alpha_I, \beta_I \in W \cup V_T$, (the subscripts E and I stand for *exterior* and *interior* respectively), we denote by $V(\alpha_E, \alpha_I, R, \beta_I, \beta_E)$ the minimum cost of any subsequence that starts at α_I , ends at β_I as generated by the context-sensitive sequence grammar $G_{\alpha_E R \beta_E}$ (that is, the grammar with R as the starting symbol and with the strings α_E, β_E as the surrounding “context”). In short, α_E, β_E give restrictions on the strings of terminals that appear on the left and on the right of R , and α_I, β_I , give restrictions on how the string generated by R (when escorted by α_E, β_E) starts and ends. Any of the strings α_E, β_E can be the empty set, which means that we place no restriction on the subsequence that precedes or follows the subsequence generated by R . The total number of 4-tuples $(\alpha_E, \alpha_I, \beta_I, \beta_E)$ is $O((Kn^2)^4) = O(K^4 n^8)$.

Given a nonterminal R , we say that the 4-tuple $(\alpha_E, \alpha_I, \beta_I, \beta_E)$ is *compatible with R* if the last $\min\{|Objects(R)|; |\alpha_I|\}$ objects of α_I belong to $Objects(R)$; and the first $\min\{|Objects(R)|; |\beta_I|\}$ objects of β_I belong to $Objects(R)$. For each nonterminal R , we define the set $ExtStates(R) = \{(\alpha_E, \alpha_I, \beta_I, \beta_E) \text{ compatible with } R\}$. We also define the sets $InitExtStates(R) = \{(\alpha_E, \alpha_I) : (\alpha_E, \alpha_I, \beta_I, \beta_E) \in ExtStates(R) \text{ for some } \beta_I, \beta_E\}$, $EndCities(R) = \{(\beta_I, \beta_E) : (\alpha_E, \alpha_I, \beta_I, \beta_E) \in ExtStates(R) \text{ for some } \beta_I, \beta_E\}$.

Given a production rule $p : \gamma_1 A \gamma_2 \rightarrow \gamma_1 R_1, R_2, \dots, R_r \gamma_2$, we define $ExtStates(p)$ and $V(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ in a similar way. The set $ExtStates(p)$ contains the 4-tuples $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in ExtStates(R)$ such that $\alpha_E = \alpha'_E \gamma_1$, and $\beta_E = \gamma_2 \beta'_E$ for some strings $\alpha'_E, \beta'_E \in V_T^*$.

We refer to a 5-tuple of the form $(\alpha_E, \alpha_I, R, \beta_I, \beta_E)$ or $(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ as a *state* of the dynamic programming recursion. The best tour generated by the grammar has value $\min_{i,j} \{V(\emptyset, i, S, j, \emptyset) + f(j, i)\}$.

We remark that, when G is a context free sequence grammar, the 4-tuples of $ExtStates(R)$ or $ExtStates(p)$ collapse to 4-tuples of the form $(\emptyset, i, j, \emptyset)$ for objects i, j , and we thus recover the generic DP solver of Section 4.4.1.

The following is a dynamic programming recursion for computing the best tour in a TSP Neighborhood grammar.

1. If a is a terminal with $Objects(a) = k$, then

$$V(\alpha_E, \alpha_I, a, \beta_I, \beta_E) = \begin{cases} 0 & \text{if } Objects(\alpha_I) = \{k\} \text{ and } Objects(\beta_I) = \{k\}, \\ \infty & \text{otherwise.} \end{cases}$$

2. Suppose p is the production rule $\gamma_1 A \gamma_2 \rightarrow \gamma_1 R_1, R_2, \dots, R_r \gamma_2$. Then,

for $k = 2$ to r and for $(\alpha_E, \alpha_I) \in InitExtStates(R_1), (\beta_I^k, \beta_E^k) \in EndExtStates(R_k)$:

$$V(\alpha_E, \alpha_I, R_1, R_2, \dots, R_k, \beta_I^k, \beta_E^k) =$$

$$\min\{V(\alpha_E, \alpha_I, R_1, R_2, \dots, R_{k-1}, \beta_I^{k-1}, \beta_E^{k-1}) + f(EndObject(\beta_I^{k-1}), InitObject(\beta_E^{k-1})) +$$

$$V(\beta_I^{k-1}, \beta_E^{k-1}, R_k, \beta_I^k, \beta_E^k) :$$

$$(\beta_I^{k-1}, \beta_E^{k-1}) \in EndExtStates(R_{k-1}), (\beta_I^{k-1}, \beta_E^{k-1}, \beta_I^k, \beta_E^k) \in ExtStates(R_k)\},$$

For $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in ExtStates(p)$:

$$V(\alpha_E, \alpha_I, p, \beta_I, \beta_E) = V(\alpha_E, \alpha_I, R_1, R_2, \dots, R_r, \beta_I, \beta_E)$$

3. If R is a non-terminal, then for $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in ExtStates(R)$:

$$V(\alpha_E, \alpha_I, R, \beta_I, \beta_E) = \min\{$$

$$V(\alpha_E, \alpha_I, p, \beta_I, \beta_E) : p \text{ is a production rule that applies to } \alpha_E R \beta_E\}.$$

Proposition 4.4.24. *Suppose that p is the production rule $p : \gamma_1 A \gamma_2 \rightarrow \gamma_1 R_1 R_2 \dots R_r \gamma_2$. Then the time to compute $V(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ for all $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in \text{ExtStates}(p)$ from the values for $R_1 R_2 \dots R_r$ is*

$$\text{Time}(p) = O\left(\sum_{k=2}^r |\text{ExtStates}(R_1 R_2 \dots R_{k-1})| \times |\text{ExtStates}(R_k)|\right)$$

Proof. It follows directly from the dynamic programming recursion given in (2) of the generic algorithm. \square

Theorem 4.4.25. *Let K be the number of production rules of a restricted context-sensitive sequence grammar for a problem with n cities. Then the time to compute the best tour generated by the grammar is $O(K^9 n^{16})$. If the grammar is in extended normal form, the time is $O(K^4 n^{13})$.*

Proof. Consider a production rule p and consider the result in Proposition 4.4.24. For each k , we can bound $|\text{ExtStates}(R_1 R_2 \dots R_{k-1})|$ by $|\text{InitExtStates}(R_1)| \times |\text{ExtStates}(R_{k-1})|$. The bounds $|\text{InitExtStates}(R)| = O(K^2 n^4)$, $|\text{EndExtStates}(R)| = O(K^2 n^4)$, and $|\text{ExtStates}(R)| = O(K^4 n^8)$ hold for any terminal or non-terminal R . Since $\text{Objects}(R_i) \cap \text{Objects}(R_j) = \emptyset$ for all $1 \leq i < j \leq r$, we claim that each 2-tuple (β_I, β_E) appears in at most one of the sets $\text{EndExtStates}(R_k) \cap \text{InitExtStates}(R_{k+1})$ for $1 \leq k \leq n$. This is so since the string β_I imposes conditions on how the subsequence generated by R_k has to start, that at most one of the R_1, \dots, R_r satisfy, and similarly, β_E imposes conditions over R_{k+1} that at most one of the R_1, \dots, R_r satisfy. Thus, the bound $\sum_{k=1}^r |\text{ExtStates}(R_k)| \leq K^4 n^8$ holds. Therefore, we can bound $\text{Time}(p)$ as follows.

$$\begin{aligned} \text{Time}(p) &= O\left(\sum_{k=2}^r |\text{ExtStates}(R_1 R_2 \dots R_{k-1})| \times |\text{ExtStates}(R_k)|\right) = \\ &O\left(\sum_{k=2}^r |\text{InitExtStates}(R_1)| \times |\text{EndExtStates}(R_{k-1})| \times |\text{ExtStates}(R_k)|\right) = \\ &O\left(\sum_{k=2}^r K^2 n^4 \times |\text{EndExtStates}(R_{k-1})| \times K^4 n^8\right) = O(K^8 n^{16}). \end{aligned}$$

The time to compute $V(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ for all p and all $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in \text{ExtStates}(p)$ is then $O(K^9 n^{16})$. The time to compute $V(\alpha_E, \alpha_I, R, \beta_I, \beta_E)$ for all terminal or non-terminal R and any $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in \text{ExtStates}(R)$ is bounded by K times the maximum number of states in $\text{ExtStates}(R)$, that is, $O(K^5 n^8)$. Therefore, the time to compute the best neighbor generated by a context-sensitive grammar in extended normal form is $O(K^9 n^{16})$.

When the sequence grammar is in extended normal form, all the production rules have at most one non-terminal on the right hand side. That is, p is either of the form $p : \gamma_1 A \gamma_2 \rightarrow \gamma_1 \alpha B \gamma_2$, $p : \gamma_1 A \gamma_2 \rightarrow \gamma_1 B \alpha \gamma_2$ or $p : \gamma_1 A \gamma_2 \rightarrow \gamma_1 \alpha \gamma_2$, where γ_1, γ_2 and α are strings of terminals, and B is a non-terminal. We analyze $p : \gamma_1 A \gamma_2 \rightarrow \gamma_1 \alpha B \gamma_1$ (the other cases are similar). In this case, the time to compute $V(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ is $O(K^4 n^8)$. Since there are K production rules, the time to compute $V(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ for all $(\alpha_E, \alpha_I, p, \beta_I, \beta_E)$ is then $O(K^5 n^8)$.

This is the same time as the time to compute $V(\alpha_E, \alpha_I, R, \beta_I, \beta_E)$ for all terminal or non-terminal R and any $(\alpha_E, \alpha_I, \beta_I, \beta_E) \in \text{ExtStates}(R)$. Therefore, the time to compute the best neighbor generated by a context-sensitive grammar in extended normal form is $O(K^5 n^8)$. \square

4.5 Complexity questions

In this Section we study some theoretical aspects of sequence grammars. In the next subsection we show that, from the standpoint of computational complexity, the set of left regular sequence grammars is strictly contained in the set of context-free sequence grammars. That is, there are neighborhoods that can be generated by context-free grammars with a polynomial number of production rules, but they cannot be generated by left regular sequence grammars with a polynomial number of production rules. We also show that any context-free sequence grammar that generates the complete neighborhood must have an exponential number of production rules. Furthermore, we show a VLSN that cannot be defined by a context-free sequence grammar with a polynomial number of production rules.

In subsection 4.5.2, we present sufficient conditions for a sequence grammar to be unambiguous. In subsection 4.5.3, we show an algorithm that checks ambiguity and tour ambiguity

for sequence left regular grammars. We also present algorithms that check whether two left regular grammars generate a common permutation, and whether the neighborhood generated by one of them is included in the neighborhood generated by the other.

4.5.1 Hierarchy of Sequence Grammars

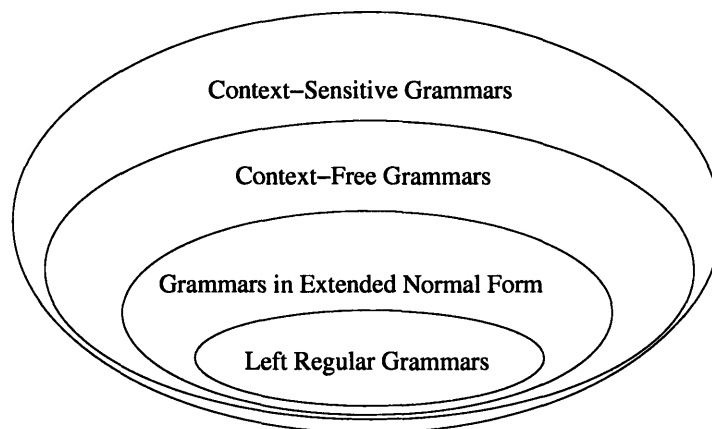


Figure 4-1: Hierarchy of sequence grammars

By definition, the set of sequence left regular grammars is included in the set of sequence extended grammars, which in turn is included in the set of sequence context-free grammars. Figure 4-1 describes the relations between the different grammars. We observe that any neighborhood N can be defined by a sequence left regular grammar G : simply define the production rules of G as $p_\pi : S \rightarrow \pi$, for each sequence $\pi \in N$. In particular, the set of neighborhoods generated by the sequence context-free grammars is the same as the set of neighborhoods generated by the sequence left regular grammars. However, this reduction is not polynomial. It turns out that there is no polynomial reduction from sequence extended grammars to sequence left regular grammars, nor from sequence context-free grammars to sequence extended grammars, nor from context-sensitive sequence grammars to context-free sequence grammars. We prove these results in this subsection. We also show that the ASSIGN neighborhood, a very large scale neighborhood, cannot be defined by a context-free sequence grammar with a polynomial number of production rules.

Proposition 4.5.1. *Any left regular sequence grammar that generates the pyramidal neighborhood has $\Omega(\frac{2^n}{n^{3/2}})$ production rules. In particular, there is no polynomial reduction from extended sequence grammars to left regular sequence grammars.*

Proof. The proof is based on a counting argument. Let G_n be a left regular grammar with K_{G_n} production rules, that generates the pyramidal neighborhood. By Lemma 4.5.11, there exists a left regular grammar G'_n such that

1. $L(G'_n) = L(G_n)$,
2. G'_n is a left regular grammar with $K_{G'_n} = O(nK_{G_n})$ production rules,
3. All its production rules are of the form $p : A \rightarrow aB$ or $p : A \rightarrow a$ for some non-terminals A, B and some terminal a .

We claim that $K_{G'_n}$ is $\Omega(\frac{2^n}{\sqrt{n}})$. This is so since for each subset $W \subset \{1, \dots, n\}$ with $\lfloor n/2 \rfloor$ elements, there exists a pyramidal sequence $\pi^W \in N_{\text{Pyramid}}$ with its first $\lfloor n/2 \rfloor$ objects being the objects of W in increasing order. Let p_1^W, \dots, p_n^W be a sequence of production rules in $K_{G'_n}$ that derivates π^W . We associate to W the $\lfloor n/2 \rfloor$ -th production rule in the sequence, namely $p_{\lfloor n/2 \rfloor}^W$. This production rule is of the form $A \rightarrow aB$, where A, B are nonterminals, a is a terminal, and $\text{Objects}(B) = \{1, \dots, n\} \setminus W$. In particular, two different subsets $W, W' \subset \{1, \dots, n\}$ with $\lfloor n/2 \rfloor$ elements have associated different production rules $p_{\lfloor n/2 \rfloor}^W, p_{\lfloor n/2 \rfloor}^{W'}$. Therefore, the number of production rules of G' is at least the number of different subsets of $\{1, \dots, n\}$ with $\lfloor n/2 \rfloor$ elements, which is $\binom{n}{\lfloor n/2 \rfloor} = \Omega(\frac{2^n}{\sqrt{n}})$. This in turn implies that the number of production rules of G is at least $\frac{K_{G'}}{n} = \Omega(\frac{2^n}{n^{3/2}})$.

Since in Section 4.3.2 we showed a sequence grammar in extended normal form that generates the pyramidal neighborhood using $\Theta(n)$ production rules, we conclude that there are neighborhoods that can be efficiently generated by a sequence grammar in extended normal but cannot be efficiently generated by a left regular sequence grammar. \square

The proof of the previous proposition implies something stronger. Any left regular grammar that generates a neighborhood that contains the pyramidal neighborhood must have

$\Omega(\frac{2^n}{n^{3/2}})$ production rules. In particular, we have the following corollary.

Corollary 4.5.2. *A left regular grammar that generates the complete neighborhood has $\Omega(\frac{2^n}{n^{3/2}})$ production rules.*

The next proposition shows a similar result regarding context-free sequence grammars and extended sequence grammars.

Proposition 4.5.3. *There is no polynomial reduction from context-free sequence grammars to extended sequence grammars.*

Proof. The proof is similar to the one of Proposition 4.5.1. For each n , let us define the *doubly pyramidal neighborhood* with tours of the form $\pi = (\pi_1, \pi_2)$ where π_1 is a pyramidal sequence on the objects $\{1, \dots, n/2\}$, and π_2 is a pyramidal sequence on the objects $\{n/2 + 1, \dots, n\}$. The context-free grammar G_n that generates such neighborhood is as follows.

The Doubly Pyramidal Neighborhood Grammar

$$\begin{aligned}
 S &\rightarrow A_{1, \lfloor n/2 \rfloor}, A_{\lfloor n/2 \rfloor + 1, n} \\
 A_{j, \lfloor n/2 \rfloor} &\rightarrow j, A_{j+1, \lfloor n/2 \rfloor}, \text{ for } 1 \leq j < \lfloor n/2 \rfloor \\
 A_{j, \lfloor n/2 \rfloor} &\rightarrow A_{j+1, \lfloor n/2 \rfloor}, j, \text{ for } 1 \leq j < \lfloor n/2 \rfloor \\
 A_{\lfloor n/2 \rfloor, \lfloor n/2 \rfloor} &\rightarrow \lfloor n/2 \rfloor \\
 A_{j, n} &\rightarrow j, A_{j+1, n}, \text{ for } n/2 < j \leq n - 1 \\
 A_{j, n} &\rightarrow A_{j+1, n}, j, \text{ for } 1 \leq j \leq n - 1 \\
 A_{n, n} &\rightarrow n.
 \end{aligned}$$

The value of the function *Objects* on each terminal and non-terminal is $Objects(S) = \{1, \dots, n\}$, $Objects(A_{j, n/2}) = \{j, \dots, n/2\}$ for $1 \leq j \leq n/2$, $Objects(A_{j, n}) = \{j, \dots, n\}$ for $n/2 < j \leq n$, $Objects(j) = \{j\}$ for $1 \leq j \leq n$. G_n is a context free grammar with $K_{G_n} = \Theta(n)$ production rules.

Let G'_n be a sequence grammar in extended normal form with $K_{G'_n}$ production rules, that generates the same language as G_n . That is, the production rules of G'_n are of the form

$p : A \rightarrow a_1 \dots a_r B$, or $p : A \rightarrow B a_1 \dots a_r$, or $p : A \rightarrow a_1 \dots a_r$. Lemma 4.5.12 shows that there exists an extended regular grammar G''_n such that

1. $L(G''_n) = L(G'_n)$,
2. G''_n is an extended regular grammar with $K_{G''_n} = O(nK_{G'_n})$ production rules,
3. All its production rules are of the form $p : A \rightarrow aB$ or $p : A \rightarrow Ba$ or $p : A \rightarrow a$ for some non-terminals A, B and some terminal a .

Since G'' is in extended normal form and since all its production rules have at most one nonterminal, and one terminal on their RHS, any derivation of a string is as follows. It starts from the starting symbol S , and it ends in a sequence of terminals. At any point of this derivation, we have a sequence of terminals and nonterminals of the form $\alpha_1 R \alpha_2$, where α_i are strings of terminals and R is a nonterminal.

Given a (sub)sequence $\pi_1 \in L(A_{1, \lfloor n/2 \rfloor})$, we say that G''_n *left weakly derives* π_1 if there exists a derivation of a sequence such that, in an intermediate step, it obtains a sequence $\pi_1 R_1 \alpha_2$, where R_1 is a nonterminal and α_2 is a (possibly empty) sequence of terminals. Similarly, we say that G''_n *right weakly derives* $\pi_2 \in L(A_{\lfloor n/2 \rfloor + 1, n})$ if there exists a derivation such that, in an intermediate step, it obtains a sequence α_1, R_1, π_2 , where R_1 is a nonterminal and α_1 is a (possibly empty) sequence of terminals. We claim that either G''_n left weakly derives all $\pi_1 \in L(A_{1, \lfloor n/2 \rfloor})$, or it right weakly derives all $\pi_2 \in L(A_{\lfloor n/2 \rfloor + 1, n})$. The proof is by contradiction. Assume there exist $\pi_1 \in L(A_{1, \lfloor n/2 \rfloor}), \pi_2 \in L(A_{\lfloor n/2 \rfloor + 1, n})$ such that G''_n does not left weakly derive π_1 and does not right weakly derive π_2 . Given any derivation of (π_1, π_2) , it looks like we add one object at a time, either to the right or to the left, until we add the last object. We start from the starting symbol S with no objects on its side. At any point of this derivation, we have something of the form $\alpha_1 R \alpha_2$, where α_i are strings of terminals and R is a nonterminal. At some point of this derivation, we have either $\lfloor n/2 \rfloor$ objects at the left of the nonterminal R , or $\lceil n/2 \rceil$ objects at the right of the nonterminal R . But the first case would imply that G''_n left weakly derive π_1 , which we assume is not possible, and the second case would imply that G''_n right weakly derive π_2 , which we also

assume is not possible. Therefore, we reached a contradiction.

WLOG, we can assume that G''_n left weakly derivates all $\pi_1 \in L(A_{1, \lfloor n/2 \rfloor})$. The rest of the proof is similar to the proof of Proposition 4.5.1. For each subset $W \subset \{1, \dots, \lfloor n/2 \rfloor\}$ with $\lfloor n/4 \rfloor$ objects, there exists a sequence $\pi^W \in L(G''_n)$ such that its first $\lfloor n/4 \rfloor$ objects are the objects of W in increasing order. Let p_1^W, \dots, p_n^W be a sequence of production rules in $K_{G''_n}$ that derivates π^W . We associate to W the $\lfloor n/4 \rfloor$ -th production rule of the form $A \rightarrow aB$, and we call it p^W , in the sequence p_1^W, \dots, p_n^W . This production rule satisfies that $Objects(B) \cap \{1, \dots, \lfloor n/2 \rfloor\} = \{1, \dots, \lfloor n/2 \rfloor\} \setminus W$. Thus, two different subsets $W, W' \subset \{1, \dots, \lfloor n/2 \rfloor\}$ with $\lfloor n/4 \rfloor$ elements have associated different production rules $p^W, p^{W'}$. Therefore, the number of production rules of G'' is at least the number of different subsets of $\{1, \dots, \lfloor n/2 \rfloor\}$ with $\lfloor n/4 \rfloor$ elements, which is $\binom{\lfloor n/2 \rfloor}{\lfloor n/4 \rfloor} = \Omega(\frac{2^{n/2}}{\sqrt{n}})$. This in turn implies that the number of production rules of G' is at least $\frac{K_{G''}}{n} = \Omega(\frac{2^{n/2}}{n^{3/2}})$. \square

The proof of the previous proposition says that any sequence grammar in extended normal form that generates a language that contains the doubly pyramidal neighborhood has $\Omega(\frac{2^{n/2}}{n^{3/2}})$ production rules. This in turn implies a lower bound on the number of production rules that a sequence grammar in extended normal form that generates the complete neighborhood has. However, the next theorem proves a stronger lower bound, which is valid for any context-free sequence grammar. In order to prove this theorem, we first introduce some notation and we prove a lemma.

Definition 4.5.4. *Let N be a neighborhood generated by a context-free sequence grammar G . Given $\pi \in N$ and a nonterminal $A \in G$, we say that A is compatible with π (and vice versa) if the objects of A appear consecutively in π , i.e., if there exists $1 \leq i \leq n$ such that $Objects(A) = \{\pi(i), \pi(i+1), \dots, \pi(i + |Objects(A)| - 1)\}$.*

The following lemma holds. It applies to context-free sequence grammar with at most two terminals/non-terminals on the left hand side of any production rule, not both of them being terminals.

Lemma 4.5.5. *Let G be a context-free sequence grammar for a problem with $n \geq 3$ objects, with production rules of the form $p : A \rightarrow a$, $p : A \rightarrow aB$, $p : A \rightarrow Ba$, $p : A \rightarrow BC$, where a is a terminal and A, B, C are nonterminals. Let N be the neighborhood generated by this grammar. For any $\pi \in N$, there exists a nonterminal $A \in G$, compatible with π , such that $\frac{n}{3} \leq |\text{Objects}(A)| \leq \frac{2n}{3}$.*

Proof. Let T be the set of nonterminals $A \in G$ such that $\frac{n}{3} \leq |\text{Objects}(A)| \leq \frac{2n}{3}$.

Since G generates $\pi \in N$, there exists a set of production rules p_1^π, \dots, p_t^π in G that derives π . It is easy to see that all the production rules p_1^π, \dots, p_t^π apply to nonterminals that are compatible with π . Assume that these production rules are ordered by the number of objects of the nonterminal that are applied to. That is, each p_i^π applies to nonterminal A_i^π , and $|\text{Objects}(A_i^\pi)| \geq |\text{Objects}(A_{i+1}^\pi)|$. We remind that all production rules of G are of the form $p : A \rightarrow a$, $p : A \rightarrow aB$, $p : A \rightarrow Ba$, and $p : A \rightarrow BC$. In particular, the first production rule of the sequence p_1^π, \dots, p_t^π applies to the starting symbol S , and the last applies to a nonterminal with one object. Therefore, for some $1 \leq j < t$, the first j production rules of this sequence applies to nonterminals with at least $\frac{2n}{3}$ objects, while the last $t - j$ production rules applies to nonterminals with at most $\frac{2n}{3} - 1$ objects. If $|\text{Objects}(A_j^\pi)| = \frac{2n}{3}$, then we prove the lemma since A_j^π is the desired nonterminal.

If $|\text{Objects}(A_j^\pi)| > \frac{2n}{3}$, we consider the following cases: $p_j^\pi : A_j^\pi \rightarrow a_j^\pi$, $p_j^\pi : A_j^\pi \rightarrow a_j^\pi B_j^\pi$, $p_j^\pi : A_j^\pi \rightarrow B_j^\pi a_j^\pi$, and $p_j^\pi : A_j^\pi \rightarrow B_j^\pi C_j^\pi$.

The first case is not possible since $n \geq 3$.

Since j is the index such that $|\text{Objects}(A_j^\pi)| \geq \frac{2n}{3}$ and $|\text{Objects}(A_{j+1}^\pi)| < \frac{2n}{3}$, the second and third cases imply that $|\text{Objects}(A_j^\pi)| - 1 = |\text{Objects}(B_j^\pi)| = \frac{2n}{3}$ and therefore we prove the lemma since B_j^π is the desired nonterminal.

Finally, in the fourth case, either B_j^π or C_j^π (say B_j^π) has at least half of the objects of nonterminal A_j^π . Then, $|\text{Objects}(B_{j+1}^\pi)| \geq \frac{n}{3}$. Since j is the index such that $|\text{Objects}(A_j^\pi)| \geq \frac{2n}{3}$ and $|\text{Objects}(A_{j+1}^\pi)| < \frac{2n}{3}$, the nonterminal B_j^π has at most $\frac{2n}{3}$ objects. Therefore, $|\text{Objects}(B_{j+1}^\pi)| < \frac{2n}{3}$ and thus this case also satisfies the claim. \square

The next theorem proves an exponential lower bound on the number of production rules

of any sequence context-free grammar that generates the complete neighborhood. It proves something stronger, which is that any context-free sequence grammar G that generates a neighborhood with a size which is a polynomial fraction of the size of the complete neighborhood must have an exponential number of production rules.

Theorem 4.5.6. *A sequence of context-free grammars G_n that generate neighborhoods N_n with size $|N_n| = \Omega(\frac{n!}{f(n)})$ for some polynomial f have $K_n = \Omega(\frac{1}{n^{5/2}f(n)}(\frac{3}{2^{2/3}})^n)$ production rules. In particular, a sequence of context-free grammars G_n that generate the complete neighborhood S_n have $K_n = \Omega(n^{-5/2}(\frac{3}{2^{2/3}})^n) = \Omega(n^{-5/2}1.8898^n)$ production rules.*

Proof. The proof follows by a counting argument. Lemma 4.5.12 shows that there exists G'_n that generates the same neighborhood as G_n , with $K'_n = O(nK_n)$ production rules, and all of them of the form $p : A \rightarrow a$, $p : A \rightarrow aB$, $p : A \rightarrow Ba$, $p : A \rightarrow BC$, where a is a terminal and A, B, C are nonterminals.

Let T_n be the set of nonterminals $A \in G'_n$ with $m = |\text{Objects}(A)|$ such that there exists a production rule that applies to A and $\frac{n}{3} \leq m \leq \frac{2n}{3}$. It is clear that $0 \leq |T_n| \leq K_n$.

Assuming $n \geq 3$, Lemma 4.5.5 says that for any $\pi \in N_n$, there exists a nonterminal $A \in T_n$ that is compatible with π . In other words, the number of sequences $\pi \in N_n$ that are compatible with some $A \in T_n$ is exactly $|N_n|$.

Given a fixed nonterminal $A \in T_n$, let $E_A = \{\pi \in N_n : \pi \text{ is compatible with } A\}$ be the set of sequences $\pi \in N_n$ that are compatible with A . The size of E_A is at most

$$n \times |\text{Objects}(A)|! \times (n - |\text{Objects}(A)|)! \leq \max\{n \times m!(n - m)! : \frac{n}{3} \leq m \leq \frac{2n}{3}\} \leq n \times \left(\frac{n}{3}\right)! \left(\frac{2n}{3}\right)! \quad (4.9)$$

The inequality $\max\{n \times m!(n - m)! : \frac{n}{3} \leq m \leq \frac{2n}{3}\} \leq n \times \left(\frac{n}{3}\right)! \left(\frac{2n}{3}\right)!$ holds since

$$n \times m!(n - m)! = n \times n! \times \binom{n}{m}^{-1},$$

and since $\min\left\{\binom{n}{m} : \frac{n}{3} \leq m \leq \frac{2n}{3}\right\} = \binom{n}{n/3}$.

Since every $\pi \in N_n$ is compatible with at least one nonterminal $A \in T_n$, and since the number of sequences that are compatible with a fixed nonterminal $A \in T_n$ is at most $n \times (\frac{n}{3})!(\frac{2n}{3})!$, the following inequalities hold.

$$\Omega\left(\frac{n!}{f(n)}\right) = |N_n| \leq \sum_{A \in T_n} |E_A| \leq K'_n \times n \times \left(\frac{n}{3}\right)!\left(\frac{2n}{3}\right)!. \quad (4.10)$$

This implies that

$$K'_n = \Omega\left(\frac{n!}{nf(n)\left(\frac{n}{3}\right)!\left(\frac{2n}{3}\right)!}\right) = \Omega\left(\frac{1}{n^{3/2}f(n)}\left(\frac{3}{2^{2/3}}\right)^n\right). \quad (4.11)$$

The last equality holds by Stirling's formula: $n! \approx \sqrt{2\pi n}n^{n+1/2}e^{-n}$. Since $K'_n = O(nK_n)$, the bound $K_n = \Omega\left(\frac{1}{n^{5/2}f(n)}\left(\frac{3}{2^{2/3}}\right)^n\right) = \Omega\left(\frac{1.8898^n}{n^{5/2}f(n)}\right)$ holds. \square

In Section 4.3.5 we showed that the complete neighborhood can be generated by a context-sensitive sequence grammar with a polynomial number of production rules. This result and the theorem we just proved imply the following corollary.

Corollary 4.5.7. *There is no polynomial reduction from context sensitive sequence grammars to context-free sequence grammars.*

The next theorem shows a VLSN for sequencing problems that cannot be *efficiently* defined by a sequence grammar. *Efficient* in this context means “with a polynomial number of production rules”. Its proof is similar to the one of Theorem 4.5.6.

Sarvanov and Doroshko [52] defined the neighborhood ASSIGN_n as

$$\text{ASSIGN}_n = \{\sigma \in S_n : \sigma(2i - 1) = 2i - 1 \text{ for all } i = 1, \dots, \lceil \frac{n}{2} \rceil\}.$$

This neighborhood has size $\lfloor \frac{n}{2} \rfloor!$. It can be searched in $O(n^3)$ since it is equivalent to an assignment problem with $\lfloor \frac{n}{2} \rfloor$ objects.

Theorem 4.5.8. *For each n , let N_n be the neighborhood ASSIGN_n . A sequence of context-free grammars G_n that generate the neighborhood N_n have $K_n = \Omega\left(\frac{1}{n^{5/2}}\left(\frac{3}{2^{2/3}}\right)^{n/2}\right) = \Omega\left(\frac{1.3747^n}{n^{5/2}}\right)$*

production rules.

Proof. By Lemma 4.5.12, there exists G'_n that generates the same neighborhood as G_n , with $K'_n = O(nK_n)$ production rules, and all of them of the form $p : A \rightarrow a$, $p : A \rightarrow aB$, $p : A \rightarrow Ba$, $p : A \rightarrow BC$, where a is a terminal and A, B, C are nonterminals.

Let T_n be the set of nonterminals $A \in G'_n$ with $m = |\text{Objects}(A)|$ such that there exists a production rule that applies to A and $\frac{n}{3} \leq m \leq \frac{2n}{3}$. It is clear that $0 \leq |T_n| \leq K_n$.

Lemma 4.5.5 says that for any $\pi \in N_n$, there exists a nonterminal $A \in T_n$ that is compatible with π . In other words, the number of sequences $\pi \in N_n$ that are compatible with some $A \in T_n$ is exactly $|N_n|$.

Given a fixed nonterminal $A \in T_n$, let $E_A = \{\pi \in N_n : \pi \text{ is compatible with } A\}$ be the set of sequences $\pi \in N_n$ that are compatible with A . Since all sequences of N_n have fixed their odd numbered objects, the size of E_A is at most

$$|E_A| \leq n \times \left\lceil \frac{|\text{Objects}(A)|}{2} \right\rceil! \times \left\lceil \frac{n - |\text{Objects}(A)|}{2} \right\rceil! \leq n \times \left(\frac{n}{6}\right)! \left(\frac{2n}{6}\right)! \quad (4.12)$$

Since every $\pi \in N_n$ is compatible with at least one nonterminal $A \in T_n$, and since the number of sequences that are compatible with a fixed nonterminal $A \in T_n$ is at most $n \times \left(\frac{n}{6}\right)! \left(\frac{2n}{6}\right)!$, the following inequalities hold.

$$\left(\frac{n}{2}\right)! = |N_n| \leq \sum_{A \in T_n} |E_A| \leq K'_n \times n \times \left(\frac{n}{6}\right)! \left(\frac{2n}{6}\right)!. \quad (4.13)$$

This implies that

$$K'_n = \Omega\left(\frac{\left(\frac{n}{2}\right)!}{n\left(\frac{n}{6}\right)! \left(\frac{2n}{6}\right)!}\right) = \Omega\left(\frac{1}{n^{3/2}} \left(\frac{3}{2^{2/3}}\right)^{n/2}\right). \quad (4.14)$$

The last equality holds by Stirling's formula: $n! \approx \sqrt{2\pi n} n^{n+1/2} e^{-n}$. Since $K'_n = O(nK_n)$, the bound $K_n = \Omega\left(\frac{1}{n^{5/2}} \left(\frac{3}{2^{2/3}}\right)^{n/2}\right) = \Omega\left(\frac{1.3747^n}{n^{5/2}}\right)$ holds. \square

4.5.2 Sufficient condition for a Context-Free grammar to be unambiguous

A context-free grammar G is unambiguous when every sequence in the neighborhood N is derived by a unique parse tree. Unambiguity is a desirable property for a sequence grammar from a computational point of view: the number of production rules of G has an impact on the time needed to optimize over the neighborhood generated by this grammar. In this subsection, we show a sufficient condition for a context-free sequence grammar to be unambiguous.

A production rule $p : A \rightarrow R_1 R_2 \dots R_k$ defines a partial order $<_p$ in $Objects(A)$: we say that $i <_p j$ if $i \in Objects(R_r)$, $j \in Objects(R_t)$ for some $r < t$. The following lemma gives a sufficient condition for a sequence grammar to be unambiguous.

Lemma 4.5.9. *Let $(G, Objects)$ be a sequence grammar such that for any non-terminal A , for any two production rules defined on A , $p : A \rightarrow R_1 R_2 \dots R_k$, $p' : A \rightarrow R'_1 R'_2 \dots R'_{k'}$, there exist objects $i, j \in Objects(A)$ such that $i <_p j$ and $j <_{p'} i$. Then $(G, Objects)$ is an unambiguous sequence grammar.*

Proof. Assume there exist two parse trees T and T' that derive the same permutation σ . Then, there exist two production rules $p \in T, p' \in T'$ such that $p \neq p'$ and $Objects(p) = Objects(p')$. The permutation σ defines a total order on the objects: $i <_\sigma j$ if i precedes j in σ . This total order is compatible with the partial order $<_p$ defined by p and also with $<_{p'}$, which contradicts the incompatibility of $<_p$ and $<_{p'}$. \square

This condition can be checked in time $O(K^2 n^3)$. Examples of sequence grammars that satisfy this condition are the 2-exchange Neighborhood Grammar, the Pyramidal Neighborhood Grammar, the complete sequencing Neighborhood, the restricted sequencing Neighborhood as defined in Subsections 4.3.1 and 4.3.2. On the other hand, the Dynasearch and twisted sequence grammars, as defined in Subsection 4.3.2, are ambiguous.

4.5.3 Ambiguity, Intersection and Inclusion for regular grammars

In this subsection we give an algorithm that checks whether a left regular sequence grammar $(G, Objects)$ (i.e., with production rules of the form $A \rightarrow a_1 \dots a_k$ or $A \rightarrow Ba_1 \dots a_k$) is ambiguous. We also present algorithms that check whether two left regular grammars generate a common permutation and whether the neighborhood generated by one of them is included in the neighborhood generated by the other.

We first give a necessary and sufficient condition for a sequence grammar $(G, Objects)$ to be ambiguous for a restricted case, namely when its production rules are of the form $p : A \rightarrow aB$ or $p : A \rightarrow a$.

Proposition 4.5.10. *A sequence grammar $(G, Objects)$ with production rules of the form $p : A \rightarrow aB$ or $p : A \rightarrow a$ is ambiguous if and only if there exists a stage R reachable from initial stage S and production rules $p_1 : R \rightarrow a_1R_1, p_2 : R \rightarrow a_2R_2$ such that $a_1 = a_2$ and $L(G_{R_1}) \cap L(G_{R_2}) \neq \emptyset$.*

Proof. It follows from the definition of ambiguity of Section 4.2.4, and from the form of the production rules of G . □

Given a sequence left regular grammar G we can assume that all its non-terminals are reachable from the initial state S since we can delete the non reachable ones otherwise. We construct an auxiliary sequence grammar G' as follows. G' has the same set of terminals as G . For each production rule $p : A \rightarrow a_1, \dots, a_r, B$ of G we construct r nonterminals B_1^p, \dots, B_r^p of G' and r production rules of G' : $p_1 : A \rightarrow a_1, B_1^p, p_2 : B_1^p \rightarrow a_2, B_2^p, \dots, p_r : B_{r-1}^p \rightarrow a_r, B_r^p$. We set $Objects'(B_i^p) = \{a_{i+1}, \dots, a_r\} \cup Objects'(B)$ for $1 \leq i \leq r$.

We do the same construction with production rules of G of the form $p : A \rightarrow a_1, \dots, a_r$.

The sequence grammar $(G', Objects')$ has the following properties.

Lemma 4.5.11. *Given a sequence left regular grammar $(G, Objects)$ with K rules and n objects, the auxiliary sequence grammar $(G', Objects')$ constructed above satisfies that*

1. G' is a sequence left regular grammar, with $O(Kn)$ rules and n objects,

2. All its production rules are of the form $p : A \rightarrow aB$ or $p : A \rightarrow a$ for some non-terminals A, B and some terminal a ,
3. $L(G) = L(G')$,
4. G is ambiguous if and only if G' is.

Proof. It is clear by construction that G' is a sequence left regular grammar. Since we replace each production rule of G with $O(n)$ production rules, G' has $O(Kn)$ production rules. By construction it is clear that all production rules of G' are of the form $p : A \rightarrow aB$ or $p : A \rightarrow a$. It is also clear by construction that there is a correspondence between parse trees of G and G' . The corresponding parse trees generate the same string. Therefore, the equality $L(G) = L(G')$ holds and G is ambiguous if and only if G' is. \square

A similar lemma holds for context-free or extended normal form grammars, which is used in Section 4.5.1. We state it without proof.

Lemma 4.5.12. *Given a sequence context-free (extended normal form) grammar $(G, \text{Objects})$ with K rules and n objects, there exists a sequence context-free (extended normal form) grammar $(G', \text{Objects}')$ such that*

1. G' has $O(Kn)$ production rules,
2. All its production rules are of the form $p : A \rightarrow aB$, or $p : A \rightarrow a$, or $p : A \rightarrow BC$ for some non-terminals A, B, C and some terminal a ,
3. $L(G) = L(G')$,
4. G is ambiguous if and only if G' is.

From now on we assume that G is a sequence grammar with production rules of the form $p : A \rightarrow aB$ or $p : A \rightarrow a$. We allow G to have two non-terminals that represent the same subset of objects.

We define the following auxiliary undirected graph $\bar{G} = (\bar{V}, \bar{E})$. The nodes of \bar{G} are the terminals and non-terminals of grammar G . For any terminals or non-terminals R, R' of G

there is an edge (R, R') in \bar{E} if and only if $L(G_R) \cap L(G_{R'}) \neq \emptyset$. The following recursive relation allows us to compute \bar{A} efficiently. (R, R') belongs to \bar{A} if and only if

1. $Objects(R) = Objects(R') = \{a\}$ for some object a , or
2. there exist $p : R \rightarrow aB, p' : R' \rightarrow a'B'$ such that $a = a'$, and (B, B') belongs to \bar{A} .

By Proposition 4.5.10, in order to check the unambiguity of G it is enough to run the following loop

```

for every production rule  $p_1 : A_1 \rightarrow a_1B_1, p_2 : A_2 \rightarrow a_2B_2$ 
  if  $A_1 = A_2$  and  $a_1 = a_2$  and  $(B_1, B_2) \in \bar{A}$  then  $G$  is ambiguous;
end for

```

The following theorem computes the running time of the algorithm.

Theorem 4.5.13. *Given a sequence left regular grammar $(G, Objects)$ with K production rules and n objects, the algorithm given above checks whether it is ambiguous in $O(K^2n^2)$ steps.*

Proof. We remove non-terminals that are not reachable from S and production rules associated with these non-terminals. This operation can be done in $O(K)$ steps. We compute the auxiliary sequence grammar G' as in Lemma 4.5.11. This computation can be done in $O(Kn)$ steps. This new grammar has $K' = O(Kn)$ production rules. We observe that all its non-terminals are reachable from the initial state S . Using the recursion formulas we can compute the auxiliary graph \bar{G} corresponding to grammar G' in $O((K')^2) = O((K^2n^2))$ steps. The last checking loop runs in $O((K')^2) = O(K^2n^2)$ steps too. \square

A similar algorithm allows us to check whether two sequence left regular grammars $(G^1 = (V_{NT}^1, V_T^1, P^1, S^1), Objects_1)$ and $(G^2 = (V_{NT}^2, V_T^2, P^2, S^2), Objects_2)$ with K_1 and K_2 production rules respectively and n objects generate a common permutation. We define the auxiliary undirected graph $\bar{G} = (\bar{V}, \bar{E})$. The nodes of \bar{G} are the terminals and non-terminals

of grammars G^1 and G^2 . For any terminals or non-terminals $R \in G^1, R' \in G^2$ there is an edge (R, R') in \bar{E} if and only if $L(G^1_R) \cap L(G^2_{R'}) \neq \emptyset$. The following recursive relation holds. (R, R') belongs to \bar{A} if and only if

1. $Objects_1(R) = Objects_2(R') = \{a\}$ for some object a , or
2. there exist $p : R \rightarrow aB \in P^1, p' : R' \rightarrow a'B' \in P^2$ such that $a = a'$, and (B, B') belongs to \bar{A} .

The sequence grammars G_1, G_2 generate a common permutation if and only if the edge (S^1, S^2) belongs to \bar{A} . The following result holds.

Corollary 4.5.14. *Given two sequence left regular grammars $(G^1, Objects_1)$ and $(G^2, Objects_2)$ with K_1 and K_2 production rules respectively and n objects, the algorithm given above checks whether they generate a common permutation in $O(K_1K_2n^2)$ steps.*

A similar algorithm allows us to check whether the language generated by a sequence left regular grammar $(G^1 = (V_{NT}^1, V_T^1, P^1, S^1), Objects_1)$ is included in the generated by another sequence left regular grammar $(G^2 = (V_{NT}^2, V_T^2, P^2, S^2), Objects_2)$ with K_1 and K_2 production rules respectively and n objects. We define the auxiliary undirected graph $\bar{G} = (\bar{V}, \bar{E})$. The nodes of \bar{G} are the terminals and non-terminals of grammars G^1 and G^2 . For any terminals or non-terminals $R \in G^1, R' \in G^2$ there is an edge (R, R') in \bar{E} if and only if $L(G^1_R) \cup L(G^2_{R'}) \neq \emptyset$. The following recursive relation holds. (R, R') belongs to \bar{A} if and only if

1. $Objects_1(R) = Objects_2(R') = \{a\}$ for some object a , or
2. For every $p : R \rightarrow aB \in P^1$ there exists $p' : R' \rightarrow a'B' \in P^2$ such that $a = a'$, and (B, B') belongs to \bar{A} .

The language $L(G_1)$ is included in $L(G_2)$ if and only if the edge (S^1, S^2) belongs to \bar{A} . The following result holds.

Corollary 4.5.15. *Given two sequence left regular grammars $(G^1, Objects_1)$ and $(G^2, Objects_2)$ with K_1 and K_2 production rules respectively and n objects, the algorithm given above checks whether $L(G_1) \subseteq L(G_2)$ in $O(K_1K_2n^2)$ steps.*

4.5.4 Testing properties of Sequence Grammars

In this Subsection we address the following questions:

1. Given an unambiguous G grammar and a function $Objects : V_{NT} \cup V_T \rightarrow 2^{\{1, \dots, n\}}$, is the 2-tuple $(G, Objects)$ a sequence grammar?
2. Can we count the number of sequences generated by an unambiguous sequence grammar?
3. Given two unambiguous sequence grammars, do they generate different sequences?
4. Is an unambiguous sequence grammar symmetric (i.e. $\pi' \in N(\pi)$ iff $\pi \in N(\pi')$)?

We observe that in Section 4.5 we give sufficient conditions for a sequence grammar to be unambiguous. Moreover, question 3 was addressed in that section for left regular sequence grammars. That is, Corollary 4.5.15 answers this question for left regular sequence grammars. In this subsection we do not restrict ourselves to left regular sequence grammars. As a tradeoff, the algorithms we present for questions 3 and 4 are Monte Carlo algorithms. That is, they run in polynomial time and answer correctly with probability $1/2$.

There is a polynomial time algorithm that checks whether G generates a language with finite number of strings (see [37]). We have to check whether $Objects$ is correctly defined. This can be done in time $O(Kn)$.

We remind that we count the sequences generated by the sequence grammar, even if they represent the same tour. Many of the sequence grammars of Section 4.3.2 avoid this redundancy by fixing one object to a particular place in the sequence, for example $\pi(1) = 1$. There is a DP algorithm that answers efficiently the second question. There is an efficient randomized DP algorithm that answers correctly the third question with probability greater than $1/2$. There is an efficient randomized DP algorithm that answers correctly the fourth question with probability greater than $1/2$.

In order to answer these questions we associate two polynomials to each sequence grammar G . We first give the following definitions. For each $1 \leq i, j \leq n$, a variable $Y_{i,j}$ is

associated to the placement of object j in the i -th position. The total number of variables $Y_{i,j}$ is n^2 . Let Y denote the vector of variables $Y_{i,j}$. Each permutation $\sigma = (\sigma(1), \dots, \sigma(n))$ has associated a polynomial $g_\sigma(Y) = \prod_{i=1}^n Y_{i,\sigma(i)}$. The inverse of σ , denoted by $\sigma^{-1} = (\sigma^{-1}(1), \dots, \sigma^{-1}(n))$, has associated the polynomial $g_{\sigma^{-1}}(Y) = \prod_{i=1}^n Y_{i,\sigma^{-1}(i)}$. We give an example. Let $\sigma = (1, 4, 2, 3)$. Its inverse is $\sigma^{-1} = (1, 3, 4, 2)$. Then $g_\sigma(Y) = Y_{1,1}Y_{2,4}Y_{3,2}Y_{4,3}$ and $g_{\sigma^{-1}}(Y) = Y_{1,1}Y_{2,3}Y_{3,4}Y_{4,2}$. Each sequence grammar G has associated a polynomial $g_G(Y) = \sum_{\sigma \in L(G)} g_\sigma(Y)$. We also define the *inverted* polynomial $h_G(Y) = \sum_{\sigma \in L(G)} g_{\sigma^{-1}}(Y)$.

A tour π' belongs to the neighborhood of tour π if there exists a permutation $\sigma \in N$ such that $\pi' = \pi\sigma$. Conversely, π belongs to the neighborhood of π' if there exists a permutation $\omega \in N$ such that $\pi = \pi'\omega$. It turns out that $\omega = \sigma^{-1}$. The polynomials $g_G(Y), h_G(Y)$ have the following properties.

Lemma 4.5.16. *Let G be an unambiguous sequence grammar. Then,*

1. $g_G(Y), h_G(Y)$ have at most n^2 variables and total degree n .
2. $g_G(1, \dots, 1) = |L(G)|$.
3. Let G' be another unambiguous sequence grammar. Then, $g_G(X) = g_{G'}(X)$ if and only if $L(G) = L(G')$.
4. For each $1 \leq i, j \leq n$, let $1 \leq y_{i,j} \leq M$ be an integer. Let y be the vector of integers $y_{i,j}$. The integers $g_G(y), h_G(y)$ are at most $n!M^n$. Their bit representation are $O(n \log n + n \log M)$.
5. G is symmetric if and only if $g_G(Y) = h_G(Y)$.

Proof. Property 1 is true since it is true for the polynomials g_π, h_π associated to each permutation π .

Since $g_\pi(1, \dots, 1) = 1$ for any π , it follows that $g_G(1, \dots, 1) = \sum_{\pi \in L(G)} 1 = |L(G)|$.

Two sequence grammars are different if and only if there exists a tour π that is generated by exactly one of the two grammars. Two tours π, π' are different if and only if their associated polynomials $g_\pi, g_{\pi'}$ are different. This implies property 3.

We will prove property 4 for the polynomial $g_G(y)$, since the proof for the polynomial $h_G(y)$ is similar. Property 4 follows from the fact that for every tour π , $g_\pi(y) \leq M^n$, and therefore $g_G(y) \leq |L(G)|M^n \leq n!M^n$. By Stirling's formula, $n! \approx \sqrt{2\pi n}n^{n+1/2}e^{-n}$, and therefore the number of bits needed to write $n!$ is $O(n \log n)$. This implies the second statement of property 4.

Assume $L(G)$ is an symmetric grammar. Let us fix a tour π . Then, each $\pi' \in N(\pi)$ satisfies that $\pi \in N(\pi')$. Written in terms of permutations, the symmetry property is equivalent to the property that each $\sigma \in N$ satisfies that $\sigma^{-1} \in N$. Let us consider the function $inv : N \rightarrow N$ defined by $inv(\sigma) = \sigma^{-1}$. This function is well defined since G is invertible. It is injective since different permutations have different inverses. Since it is injective, the number of elements of its range must be equal to the number of elements of its domain. The codomain is equal to the domain and therefore it is also surjective. Using that inv is bijective we derive the following equalities.

$$g_G(Y) = \sum_{\sigma \in L(G)} g_\sigma(Y) = \sum_{inv(\sigma) \in L(G)} g_\sigma(Y) = \sum_{\sigma \in L(G)} g_{inv(\sigma)}(Y) = h_G(Y).$$

The converse is also true. If $g_G(Y) = h_G(Y)$ then both polynomials have the same monomials. Therefore, each $\sigma \in L(G)$ has associated a permutation $\omega \in L(G)$ such that $g_\sigma(Y) = h_\omega(Y)$. But this equality implies that $\omega = \sigma^{-1}$. Therefore $L(G)$ is symmetric. \square

We present the recursion formulas that allow us to evaluate g_G, h_G in a vector $y \in [1, M]^{n^2}$ in time polynomial in $|G|, \log n, \log M$.

For any terminal or non-terminal R , for any $1 \leq q \leq n$, we let (q, R) to represent the set of tours π generated by the sequence grammar such that its parse tree contains R and such that the first object in the sequence π that belongs to $Objects(R)$ appears in the q -th position. For any production rule $p : A \rightarrow R_1, R_2, \dots, R_r$, for any $1 \leq q \leq n$, we let (q, p) to represent the set of tours π such that its parse tree contains p and such that the first object in the sequence π that belongs to $Objects(A)$ appears in the q -th position.

1. If α is a terminal with $Objects(\alpha) = \{i\}$, for any $1 \leq q \leq n$ then

$$g(q, \alpha)(y) = y_{q,i},$$

$$h(q, \alpha)(y) = y_{i,q}.$$

2. Suppose p is the production rule $p : A \rightarrow R_1, R_2, \dots, R_r$.

Then, for $1 \leq q \leq n$,

$$g(q, p)(y) = \prod_{k=1}^r g(q + \sum_{t=1}^{k-1} |\text{Objects}(R_t)|, R_k)(y),$$

$$h(q, p)(y) = \prod_{k=1}^r h(q + \sum_{t=1}^{k-1} |\text{Objects}(R_t)|, R_k)(y),$$

3. If R is a non-terminal, then for $1 \leq q \leq n$,

$$g(q, R)(y) = \sum_{\{p:p \text{ applies to } R\}} g(q, p)(y),$$

$$h(q, R)(y) = \sum_{\{p:p \text{ applies to } R\}} h(q, p)(y).$$

4. $g_G(y) = g(1, S)(y), h_G(y) = h(1, S)(y)$.

The randomized algorithm to check whether two unambiguous sequence grammars G, G' generate the same neighborhood is as follows. We pick a random vector y uniformly in $[1, 2n^2]^{n^2}$ and we compute $g_G(y), g_{G'}(y)$. If $g_G(y) = g_{G'}(y)$, then the algorithm answers that the neighborhoods are the same; otherwise the algorithm answers that the neighborhoods are different. The randomized algorithm to check whether $L(G)$ is symmetric is similar. We pick a random vector y uniformly in $[1, 2n^2]^{n^2}$ and we compute $g_G(y), h_G(y)$. If $g_G(y) = h_G(y)$ the algorithm answers that G is symmetric, otherwise the algorithm answers that it is not symmetric. In order to compute $g_G(y), h_G(y)$ efficiently from the recursion formulas, we first construct the auxiliary graph $\bar{G} = (\bar{V}, \bar{A})$ where $\bar{V} = \{p : \text{production rule of } G\} \cup V_T \cup V_{NT}$. For any production rule $p : A \rightarrow R_1, R_2, \dots, R_r$, the arcs $(A, p), (p, R_1), \dots, (p, R_r)$ belong to \bar{A} . This graph has $O(Kn)$ nodes and $O(Kn)$ arcs. This graph is acyclic, otherwise the language $L(G)$ is infinite. This graph defines an order among its nodes which is obtained by breadth first search. This order is compatible with the recursion formulas in the sense that $g(q, R)(y), g(q, p)(y), h(q, R)(y), h(q, p)(y)$ are constructed using higher order terms. We then compute $g(q, R)(y), g(q, p)(y), h(q, R)(y), h(q, p)(y)$ following this order.

We analyze the running time of this algorithm. We count each arithmetic operation as 1 unit of time. The construction of \bar{G} and the breadth first search on \bar{G} takes $O(Kn)$ time. For each $1 \leq q \leq n$ and each non-terminal R , the term $g(q, R)(y)$ can be computed in $O(K)$ time. The computation of $g(q, R)(y)$ for all (q, R) takes $O(n|\bar{A}|) = O(Kn^2)$ time. For each $1 \leq q \leq n$ and for each production rule p , the term $g(q, p)(y)$ can be computed in $O(n)$ time. The computation of $g(q, p)(y)$ for all (q, p) takes $O(n|\bar{A}|) = O(Kn^2)$ time.

The following lemma is from Schwartz [53].

Lemma 4.5.17. *Let p be a polynomial in $F[X_1, \dots, X_n]$. For each $1 \leq i \leq n$, let d_i be the degree of p with respect to variable X_i , let I_i be a set of elements of F . Then, the number of zeros of p in $I_1 \times I_2 \times \dots \times I_n$ is at most $|I_1 \times I_2 \times \dots \times I_n|(\sum \frac{d_i}{|I_i|})$.*

The following theorem analyzes the correctness and the running times of the algorithms outlined above for questions 2, 3 and 4.

Theorem 4.5.18. *Given an unambiguous sequence grammar G with n objects and K production rules, we can compute $|L(G)|$ in $O(Kn^2)$ time. Given two unambiguous sequence grammars G_1, G_2 with n objects and K_1, K_2 production rules respectively, the algorithm given above is a Monte Carlo algorithm that checks if $L(G_1) \neq L(G_2)$ in $O((K_1 + K_2)n^2)$ time. That is, if $L(G_1) = L(G_2)$ the algorithm answers correctly. If $L(G_1) \neq L(G_2)$ the algorithm answers correctly with probability greater than $1/2$.*

Proof. In order to compute $|L(G)|$ we compute $g_G(1, \dots, 1)$ using the DP outlined before. With respect to the second algorithm, we pick a random vector x in $[1, 2n^2]^{n^2}$ and we compute $g_{G_1}(x), g_{G_2}(x)$ using the DP outlined before. If $g_{G_1}(x) = g_{G_2}(x)$, the algorithm answers that G_1 and G_2 generate the same tours; otherwise the algorithm answers that G_1 and G_2 generate different tours. If G_1 and G_2 generate the same tours, the polynomial $g_{G_1} - g_{G_2}$ is identically zero and therefore the numbers $g_{G_1}(x), g_{G_2}(x)$ are the same. Thus the algorithm answers correctly in this case. If G_1 and G_2 generate different tours, the polynomial $g_{G_1} - g_{G_2}$ is not zero. It has at most n^2 variables, and its degree with respect to any variable $X_{i,j}$ is at most one. By Lemma 4.5.17, the number of zeros of $g_{G_1} - g_{G_2}$ in $[1, 2n^2]^{n^2}$ is at most

$(2n^2)^{n^2} \left(\sum \frac{1}{2n^2}\right) = (2n^2)^{n^2} \times n^2 \times \frac{1}{2n^2} \leq (2n^2)^{n^2} \times \frac{n^2}{2n^2} = (2n^2)^{n^2} \times \frac{1}{2}$. The probability of $x \in [1, 2n^2]^{n^2}$ being a zero of $g_{G_1} - g_{G_2}$ is at most $\frac{(2n^2)^{n^2} \times \frac{1}{2}}{(2n^2)^{n^2}} = \frac{1}{2}$, and therefore the algorithm answers correctly with probability at least $1/2$ in this case. \square

Theorem 4.5.19. *Given an unambiguous sequence grammar G with n objects and K production rules, the algorithm given above is a Monte Carlo algorithm that checks if $L(G)$ is symmetric. That is, if $L(G)$ is symmetric the algorithm answers correctly. If $L(G)$ is not symmetric the algorithm answers correctly with probability greater than $1/2$. It runs in $O(Kn^2)$ time.*

Proof. To analyze its correctness, if G is symmetric the polynomial $g_G(Y) - h_G(Y)$ is identically zero and therefore the equality $g_G(y) = h_G(y)$ holds. Thus the algorithm answers correctly in this case. If G is not symmetric the polynomial $g_G(Y) - h_G(Y)$ is not identically zero. It has at most n^2 variables and its degree with respect to any variable $Y_{i,j}$ is at most one. By Lemma 4.5.17, the number of zeros of $g_G - h_G$ in $[1, 2n^2]^{n^2}$ is at most $(2n^2)^{n^2} \left(\sum \frac{1}{2n^2}\right) = (2n^2)^{n^2} \times n^2 \times \frac{1}{2n^2} \leq (2n^2)^{n^2} \times \frac{n^2}{2n^2} = (2n^2)^{n^2} \times \frac{1}{2}$. The probability of $y \in [1, 2n^2]^{n^2}$ being a zero of $g_G - h_G$ is at most $\frac{(2n^2)^{n^2} \times \frac{1}{2}}{(2n^2)^{n^2}} = \frac{1}{2}$ and therefore the algorithm answers correctly with probability at least $1/2$ in this case. \square

4.6 Inverse Neighborhood

Symmetry, the property that $\pi \in N(\sigma)$ if and only if $\sigma \in N(\pi)$, is standard in most neighborhood search techniques. But symmetry is often not true in the VLSN generated by grammars. Moreover, the neighborhoods may have very different properties from their inverse neighborhood. Given a set of permutations N we denote by $inv(N)$ the set of the inverse of permutations of N . That is, $inv(N) = \{\sigma^{-1} : \sigma \in N\}$. Given a set of permutations N and a tour π the *inverse neighborhood* $inv(N)(\pi)$ is the set $\{\pi\sigma^{-1} : \sigma \in N\}$. We say that a neighborhood is *symmetric* if $N = inv(N)$. The symmetric property is a natural concept in local search and it is usually assumed in the literature. For example, it is one of the conditions required by the celebrated convergence theorems of simulated annealing to hold

(see the book by Aarts and Lenstra [1]). Since not all the sequence grammar neighbors are symmetric, some natural questions are whether we can express the inverse neighborhood of a sequence grammar as a sequence grammar and whether we can optimize over the inverse neighborhood.

4.6.1 Inverse Balas-Simonetti Neighborhood

The Balas-Simonetti neighborhood [12] is as follows. It contains all sequences such that object 1 is visited first, and that object i precedes object j whenever $i + k \leq j$ for some fixed parameter k . That is,

$$N_{BS}^k = \{\sigma : \sigma(1) = 1 \text{ and } \sigma^{-1}(i) < \sigma^{-1}(j) \text{ for } i + k \leq j\}.$$

Balas and Simonetti [12] showed that, for each value of k , this neighborhood has an exponential size and can be optimized in linear time. This neighborhood can be defined by a left regular grammar, as we showed in Subsection 4.3.2. N_{BS}^k is not a symmetric neighborhood. Since its inverse neighborhood has the same (exponential) size, one may ask if we can use it to define a VLSN search algorithm. That is, we ask whether we can optimize over it efficiently. In this subsection, we show that N_{IBS}^k , the inverse Balas-Simonetti neighborhood, is defined by a left regular sequence grammar too, and we can optimize over it in linear time too (for a fixed value of k). The inverse Balas-Simonetti neighborhood is

$$N_{IBS}^k = \{\sigma : \sigma(1) = 1 \text{ and } \sigma(i) < \sigma(j) \text{ for } i + k \leq j\}.$$

Every sequence $\sigma \in N_{IBS}^k$ satisfy that $|i - \sigma(i)| \leq k$ for every $1 \leq i \leq n$, that is, the i -th object is located within k places from the i -th place (see Proposition 4.3.9). In particular, any city i such that $i - k < m$ is placed in one of the first m places by $\sigma \in N_{IBS}^k$. It also implies that if city j is placed in one of the first m places of any sequence $\sigma \in N_{IBS}^k$, then $j < m + k$.

The inverse Balas-Simonetti grammar we present is a restricted sequence grammar (see

Subsection 4.3.3). Each nonterminal $A_{(R,\alpha)}$ contains two pieces of information: $R \subset \{1, \dots, n\}$ is the subset of objects to place next, and α is a string of at most k objects that are going to be placed in that order by the next production rules. We define \mathcal{R} , a set of subsets of $\{1, \dots, n\}$, as follows. It contains the sets $R = \{1, \dots, m - k\} \cup R_m$, where $1 \leq m \leq n$, $R_m \subset \{m - k, m + k\}$, and $|R_m| = k$. For each $R \in \mathcal{R}$, we associate a set \mathcal{A}_R of strings α of $t := \min\{m, k\}$ terminals, $\alpha = a_1, \dots, a_t$, where $Objects(\alpha) \subset R \cap \{m - 2k, \dots, m + k\}$, and such that the sequence $\alpha_0 \alpha_1$ with $Objects(\alpha_0 \alpha) = R$, and with the objects in the strings α_0 and α_1 in increasing order, belongs to N_{IBS} . We define V_{NT}^{IBS} , the set of nonterminals of the inverse Balas-Simonetti grammar, as $V_{NT}^{IBS} = \{A_{(R,\alpha)} : R \in \mathcal{R}, \alpha \in \mathcal{A}_R\}$.

Inverse Balas-Simonetti Neighborhood Grammar

$S \rightarrow A_{(\{1,\dots,n\},\alpha)}$ for each $A_{(\{1,\dots,n\},\alpha)} \in V_{NT}^{IBS}$,
 $A_{(R,\alpha)} \rightarrow A_{(R',\alpha')}, j$ for each $A_{(R,\alpha)}, A_{(R',\alpha')} \in V_{NT}^{IBS}$ such that
 $R' = R \setminus \{j\}$, and $i, \alpha = \alpha', j$ for some $i \in R \setminus Objects(\alpha)$,
 $A_{(\{1\},1)} \rightarrow 1$.

It is easy to see that $|\mathcal{R}| \leq n \binom{2k}{k}$, and that $|\mathcal{A}_R| \leq k! \binom{3k}{k}$ for any $R \in \mathcal{R}$. Therefore, the number of nonterminals in the inverse Balas-Simonetti TSP Neighborhood grammar is $O(n \binom{2k}{k} \binom{3k}{k} k!) = O(n 2^{5k} k!)$. The number of production rules that can be applied to a non-terminal is $O(k)$. Therefore, the total number of production rules is $K_{IBS} = O(nk 2^{5k} k!)$. For fixed k , the number of production rules is linear in n . Therefore, the inverse Balas-Simonetti neighborhood can be efficiently searched for a list of sequencing problems, including the TSP.

In particular, it can be shown that the generic DP algorithm for TSP neighborhoods of Section 4.4.1 runs in $O(k \times K_{IBS}) = O(k^2 2^{5k} k! n)$ time in the inverse Balas-Simonetti TSP Neighborhood. In Subsection 4.6.2 we show another way of optimizing in N_{IBS} when applied to the TSP, in $O(2^{3k} k^{2k+2} n)$ time. Although these two running times are linear for fixed k , these bounds are worse than the bound we prove for optimizing over the Enlarged

Balas-Simonetti Neighborhood, which is also a bigger neighborhood.

4.6.2 Optimizing in the inverse Neighborhood of a Sequence Grammar for the TSP

In Section 4.5.1 we presented an efficient Monte Carlo algorithm for checking whether an unambiguous grammar generates an invertible neighborhood. In this subsection, we are interested on finding the best tour in the inverse neighborhood generated by a sequence grammar $(G, Objects)$. That is, we want to find the best tour in $inv(N)(\pi) = \{\pi\sigma^{-1} : \sigma \in N\}$. In this section we present an optimizer for the inverse neighborhood defined by a sequence grammar in normal form.

We observe that some of the grammar-induced neighborhoods listed in Section 4.3 are symmetric, that is, $\pi' \in N(\pi)$ iff $\pi \in N(\pi')$. In those neighborhoods, the inverse neighborhood is the same as the original neighborhood, and therefore there is no need of an optimizer for the inverse neighborhood. However, some other neighborhoods such as the pyramidal or the Balas-Simonetti neighborhoods are not symmetric, and therefore their inverse neighborhoods define a different topology on S_n . When applied to the inverse pyramidal or the inverse Balas-Simonetti neighborhood, the optimizer we present is an efficient algorithm, namely it runs in polynomial time.

We start by observing that the sequence grammar $(G, Objects)$ also generates the inverse neighborhood of $L(G)$, if we interpret a permutation $\gamma = (\gamma(1), \gamma(2), \dots, \gamma(n))$ generated by $(G, Objects)$ as a sequence of **places** instead of **cities**. That is, $\gamma(i)$ is the place where city i goes. The syntaxis of a permutation is the same as before, but the semantic differs. The cost of a permutation of places $\gamma = (\gamma(1), \gamma(2), \dots, \gamma(n))$ is $c(\gamma) = \sum_{i=1}^{n-1} c_{\gamma^{-1}(i)\gamma^{-1}(i+1)} + c_{\gamma^{-1}(n)\gamma^{-1}(1)}$.

To avoid confusion, we use the Greek letters π, σ to denote permutations of cities and the Greek letters γ, δ to denote permutations of places. We rename the function *Objects* as *Places* when we talk about sequences of places instead of sequences of cities. We then say that the sequence grammar $(G, Places)$ generates $inv(N)$. Although the inverse neighborhood

grammar $(G, Places)$ is easily defined in terms of $(G, Cities)$, computing the best tour in $inv(N)(\pi)$ is not as easy as in $N(\pi)$. The reason is that the objective function is more complex. We give some definitions first.

Definition 4.6.1. *Given a terminal or non-terminal R of $(G, Places)$, we define the boundary of R as the set $\partial R = \{i \in Places(R) : \text{either } i - 1 \notin Places(R) \text{ or } i + 1 \notin Places(R)\}$ (where numbers are cyclic: $0 = n$ and $n + 1 = 1$). We also define the set $Filling(R) = \{h : \partial R \rightarrow \{1, \dots, n\}\}$.*

For example, if $n = 10$ and $Places(R) = \{2, 3, \dots, 7\}$, then the boundary of R is the set $\{2, 7\}$. We observe that the start symbol S has an empty border. The initial tour is $\pi = (\pi(1), \pi(2), \dots, \pi(n))$.

Given a string $\alpha = a_1 \dots a_r$ of places and a number $1 \leq i \leq n$, we interpret the pair (α, i) as the decision of placing the i -th city in place a_1 , the $i + 1$ -th city in place a_2, \dots , and the $i + r - 1$ -th city in place a_r . For each pair (α, i) , we define the function $h_{(\alpha, i)} : Places(\alpha) \rightarrow \{i, \dots, i + r - 1\}$ as $h_{(\alpha, i)}(a_j) = i + j - 1$ for $1 \leq j \leq r$, and its inverse $h_{(\alpha, i)}^{-1} : \{i, \dots, i + r - 1\} \rightarrow Places(\alpha)$ as $h_{(\alpha, i)}^{-1}(j) = a_{j-i+1}$ for $i \leq j \leq i + r - 1$. These functions relate cities with places according to the pair (α, i) . We define the cost of a pair (α, i) as the sum of the cost of the edges determined by (α, i) , that is,

$$c(\alpha, i) = \sum_{s: \{s, s+1\} \subset Places(\alpha)} c_{\pi(h_{(\alpha, i)}(s)), \pi(h_{(\alpha, i)}(s+1))}.$$

Let us fix a nonterminal R . Given a function $h \in Filling(R)$, we say that a pair (α, i) (where $\alpha \in L(G_R)$ and $1 \leq i \leq n$) is *compatible with h* if $h(j) = h_{(\alpha, i)}(j)$ for all $j \in \partial R$. That is, they are compatible when h and $h_{(\alpha, i)}$ agree in the cities placed in each place of ∂R . We observe that, given $\alpha \in L(G_R)$ and $h \in Filling(R)$, there is at most one city i such that (α, i) is compatible with h . Given a terminal or non terminal R , $h \in Filling(R)$ we define

$$V(h, R) = \min\{c(\alpha, i) : \alpha \in L(G_R) \text{ and } (\alpha, i) \text{ compatible with } h\}.$$

This cost is infinite whenever there is no pair (α, i) compatible with h . In the case of the

start symbol S , the set $Filling(S)$ is empty, and therefore $V(h, S) = V(S) = \min\{c(\alpha) : \alpha \in L(G)\}$. Similarly, given a production rule $p : A \rightarrow R_1 R_2$ and $h \in Filling(A)$, we denote by $V(h, p) = \min\{c(\alpha, i) : \alpha \in L(G_{p(A)}) \text{ and } (\alpha, i) \text{ compatible with } h\}$. We remind that $L(G_{p(A)})$ is the set of strings α generated by the production rule p . We observe that given $p : A \rightarrow R_1 R_2$, the boundary of A is contained in $\partial R_1 \cup \partial R_2$. We say that $(h_1, h_2) \in Filling(R_1) \times Filling(R_2)$ is *compatible with* $h \in Filling(A)$ if they agree on the cities they place in ∂A , that is, $h(i) = h_s(i)$ for all $1 \leq s \leq 2$, and for all $i \in \partial A \cap \partial R_s$. Finally, we define the joint cost of two 2-tuples (h_1, R_1) and (h_2, R_2) as the sum of the costs of the edges they define jointly, that is,

$$c(h_1, R_1, h_2, R_2) = \sum_{k \in \partial R_1, k+1 \in \partial R_2} c_{\pi(h_1(k)), \pi(h_2(k+1))} + \sum_{k \in \partial R_2, k+1 \in \partial R_1} c_{\pi(h_2(k)), \pi(h_1(k+1))}.$$

We refer to 2-tuples of the form (h, R) or (h, p) as states of the dynamic programming recursion. The best tour generated by the grammar has value $V(S)$. The following is a dynamic programming recursion for computing the best tour in an inverse neighborhood defined by a grammar in normal form.

1. If a is a terminal with $Places(a) = j$, then for $h \in Filling(a)$:

$$V(h, a) = 0.$$

2. Suppose p is the production rule $A \rightarrow R_1 R_2$. Then, for $h \in Filling(A)$:

$$V(h, p) = \min\{V(h_1, R_1) + V(h_2, R_2) + c(h_1, R_1, h_2, R_2) : (h_1, h_2) \in Filling(R_1) \times Filling(R_2) \text{ and compatible with } h\}.$$

3. If R is a non-terminal, then for $h \in Filling(R)$:

$$V(h, R) = \min\{V(h, p) : p \text{ is a production rule that applies to } R\}.$$

The following theorem holds.

Theorem 4.6.2. *Let G be a sequence grammar in normal form with K production rules and n cities. Let $M := \max\{|Filling(R)| : R \text{ nonterminal of } G\}$. Then the algorithm given above computes the best neighbor in the neighborhood $inv(N)(\pi)$ in time $O(KnM^2)$.*

Proof. The number of stages (h, R) and (h, p) of the DP algorithm is $O(KM)$. The total time to compute $V(h, R)$ for all R and all $h \in Filling(R)$ from the values $V(h, p)$ is also $O(KM)$ since each state (h, p) is associated to a unique state (h, R) .

For each production rule of the form $A \rightarrow R_1R_2$ and for all $h \in Filling(A)$, the time to compute $V(h, p)$ from $V(h_1, R_1), V(h_2, R_2)$ is $O(M^2n)$. This is so since the total number of pairs $(h_1, h_2) \in Filling(R_1) \times Filling(R_2)$ is $O(M^2)$, and the time to compute $c(h_1, R_1, h_2, R_2)$ is $O(n)$. Therefore the time to compute $V(h, p)$ for all production rules p and for all $h \in Filling(A)$ is $O(KnM^2)$.

The total time to compute $V(S)$ is then $O(KnM^2)$. □

A trivial bound in the value of M is n^n and thus the bound proved is not polynomial. However, M is at most n^2 for a number of neighborhoods described in Section 4.3: the Adjacent Interchange, 2-opt, Pyramidal tour, Twisted Sequence, Semi-Twisted, Sequence, Dynasearch and Weak Dynasearch Neighborhoods. It is bounded by a polynomial in n for the Balas-Simonetti and the Enlarged Balas-Simonetti Neighborhoods. Among these neighborhoods, the non-symmetric ones (and therefore the interesting cases for this subsection) are the Pyramidal and the Balas-Simonetti neighborhoods. When applied to compute the best tour in the inverse pyramidal neighborhood or in the inverse Balas-Simonetti neighborhood, the algorithm has better running times than the bound $O(Kn^{2M+1})$, as the next subsection shows.

4.6.3 Running times for optimizing on the inverse neighborhood

The following is written for grammars in normal form, i.e. when there are at most two nonterminals on the right hand side of any production rule. We assume that each nonterminal R has associated a list $L(R)$ with the elements of $Filling(R)$, and that each $h \in L(R)$ has associated a list $L(h)$ with the states $\{(h, p) : p \text{ applies to } R \text{ and } L(h, p) \neq \emptyset\}$ (where $L(h, p)$

is defined in a few lines after). Similarly, we assume that each production rule p has associated a list $L(p)$ with the elements of $Filling(p)$. For each production rule $p : A \rightarrow R_1 R_2$ with two nonterminals on the right hand side, and for each $h \in L(p)$, we associate a list $L(h, p)$ with the elements $\{(h_1, h_2) : (h_1, h_2) \in Filling(R_1) \times Filling(R_2) \text{ and compatible with } h\}$. These data structures don't change from iteration to iteration of the local search algorithm. Therefore they are computed once.

The next proposition shows that we can optimize over the inverse pyramidal neighborhood with the same running time as over the pyramidal neighborhood itself.

Proposition 4.6.3. *The running time of the generic dynamic programming algorithm of Section 4.6.2 in the inverse of the Pyramidal Tour Neighborhood is $O(n^2)$.*

Proof. The number of production rules of the Pyramidal Tour Neighborhood is $K = O(n)$. Given a non-terminal $A_{j,n}$, there are two production rules that apply to it: $p : A_{j,n} \rightarrow j, A_{j+1,n}$ or $p : A_{j,n} \rightarrow A_{j+1,n}, j$. These particular type of production rules imply that $Filling(A_{j,n}) \subseteq \{h : \text{either } h(j) = n - j + 1 \text{ or } h(j) = n\}$, that is, $Filling(A_{j,n}) = O(n)$. The number of production rules that can be applied to a nonterminal is at most two. Therefore, the time to compute $V(h, R)$ for all $h \in L(R)$ is $O(n)$. Given a production rule p , say $p : A_{j,n} \rightarrow A_{j+1,n}, j$ (the analysis of the production rule $p : A_{j,n} \rightarrow j, A_{j+1,n}$ is similar), the list $L(p)$ has $O(n)$ elements. For each $(h, p) \in L(p)$, the list $L(h, p)$ has 1 element when $h(j) = j, h(n) \neq j + 1$. The number of (h, p) such that $|L(h, p)| = 1$ is $O(n)$. The list $L(h, p)$ has $O(n)$ element when $h(j) = j, h(n) = j + 1$. The number of (h, p) such that $|L(h, p)| = O(n)$ is $O(n)$. Therefore, the total time to compute $V(h, p)$ for all p , all $h \in Filling(p)$ from the values $V(h, A)$ is $O(Kn) = O(n^2)$.

Therefore the total time to run the generic dynamic programming algorithm for the inverse Pyramidal Tour Neighborhood grammar is $O(n^2)$. \square

Balas and Simonetti showed that the Balas-Simonetti neighborhood has exponential size and can be searched in linear time (for fixed parameter k). In Section 4.4.10 we showed that we can express the inverse Balas-Simonetti neighborhood as a particular context-sensitive grammar, and that we can optimize over it in linear time. As an alternative approach, the

next proposition shows that the optimizer defined in the previous section finds the optimal solution in the inverse Balas-Simonetti neighborhood in linear time, too.

Proposition 4.6.4. *The running time of the generic dynamic programming algorithm of Section 4.6.2 in the inverse of the Balas-Simonetti Neighborhood is $O(2^{3k}k^{2k+2}n)$.*

Proof. The number of production rules of the Balas-Simonetti Neighborhood is $K_{BS} = O(k2^k n)$. A sequence of the inverse Balas-Simonetti Neighborhood satisfies that $|\sigma(i) - i| \leq k$, and thus $Filling(R) \subseteq \{h : |h(i) - i| \leq k \text{ for all } i \in Places(R)\}$ for any non-terminal R . Therefore, $|Filling(R)| = O((2k)^{2k})$. Given $h \in L(R)$, the number of elements of the list $L(h, R)$ is $O(k)$ since at most one production rule p is compatible with (h, R) and at most k elements of $Filling(p)$ are compatible with h . Therefore, the total time to compute $V(h, R)$ for all R and for all $h \in Filling(R)$ is $O(K_{BS}2^{2k}k^{2k+1}) = O(2^{3k}k^{2k+2}n)$. Given a production rule p , its list $L(p)$ has $O((2k)^{2k})$ elements (again because of the fact that $|\sigma(i) - i| \leq k$ for all $\sigma \in inv(N_{BS})$). For each $(h, p) \in L(p)$, the list $L(h, p)$ has $O(k)$ elements. Therefore, the total time to compute $V(h, p)$ for all p , all $h \in Filling(p)$ from the values $V(h, A)$ is $O(K_{BS}2^{2k}k^{2k+1}) = O(2^{3k}k^{2k+2}n)$.

Therefore the total time to run the generic dynamic programming algorithm for the inverse Balas-Simonetti Neighborhood grammar is $O(2^{3k}k^{2k+2}n)$. \square

4.7 Conclusions and Open problems

We present a language to generate exponentially large neighborhoods for sequencing problems using grammars. This is the first language proposed for generating exponentially large neighborhoods. We develop generic dynamic programming solvers that determine the optimal neighbor in a neighborhood generated by a grammar for the Traveling Salesman Problem, the Linear Ordering Problem, the Minimum Latency Problem, or a Weighted Bipartite Matching Problem with Side Constraints, in time polynomial in the size of the problem and the number of rules of the grammar. Using this framework we unify a variety of previous results in defining and searching efficiently exponentially large neighborhoods for the trav-

eling salesman problem. We also present efficient algorithms for enumerating the size of neighborhoods and for deciding whether two neighborhoods are distinct in the case that the generating grammars are context-free and unambiguous. We show that regular, context-free, and context-sensitive sequence grammars generate different neighborhoods. For example, the Pyramidal neighborhood cannot be efficiently generated by a regular grammar. Moreover, we show that a sequence grammar that generates the complete neighborhood must have an exponential number of production rules. Along the way, we find some interesting questions that we could not answer. Some of them deal with the ambiguity/unambiguity property.

1. Can we check in polynomial time whether a context-free sequence grammar is unambiguous?
2. Given an ambiguous sequence grammar, can we transform it into an unambiguous sequence grammar?
3. Can extend Theorems 4.5.18 and 4.5.19 to the case where grammars are ambiguous?
4. Can we optimize in polynomial time on the inverse neighborhood defined by a grammar?
5. Given a sequence grammar, can we prove/disprove whether any sequence is reachable from any other sequence?

Bibliography

- [1] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.
- [2] R. Agarwal, Ö. Ergun, J. B. Orlin, and C. N. Potts. Improvement graphs in large-scale neighborhood search: a case study in scheduling. Working paper, School of Ind. and Sys. Engr., Georgia Institute of Technology, Atlanta, 2003.
- [3] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.
- [4] K. Altinkemer and B. Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6:149–158, 1987.
- [5] K. Altinkemer and B. Gavish. Heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24:294–297, 1990.
- [6] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics*, 46:654–670, 1999.
- [7] Sanjeev Arora. Polynomial-time approximation schemes for euclidean tsp and other geometric problem. *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 2–12, 1996.
- [8] T. Asano, N. Katoh, H. Tamaki, and T. Tokushima. Covering points in the plane by k -tours: a polynomial approximation scheme for fixed k . Research Report RT0162, IBM Tokyo Research Laboratory, 1996.

- [9] T. Asano, N. Katoh, H. Tamaki, and T. Tokushima. Covering points in the plane by k -tours: towards a polynomial time approximation scheme for general k . *Proceedings of STOC '97*, pages 275–283, 1997.
- [10] F. Aurenhammer. On-line sorting of twisted sequences in linear time. *BIT*, 28:194–204, 1998.
- [11] E. Balas. New classes of efficiently solvable generalized traveling salesman problem. *Annals of Operations Research*, 86:529–558, 1996.
- [12] E. Balas and N. Simonetti. Linear time dynamic programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, 13(1):56–75, 2001.
- [13] J. Beardwood, J. L. Halton, and J. M. Hammersley. The shortest path through many points. *Proc. Cambridge Phil. Soc.*, 55:299–327, 1959.
- [14] W. J. Bell, L. M. Dalberto, M. L. Fisher, A. J. Greenfield, R. Jaikumar, P. Kedia, R. G. Mack, and P. J. Prutzman. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *INTERFACES*, 13:4–23, 1983.
- [15] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 163–171. ACM Press, 1994.
- [16] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Sys. Sci.*, 13:335–379, 1976.
- [17] R. E. Burkard, V. G. Deineko, R. van Dal, J. A. A. van der Veen, and G. J. Woeginger. Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review*, 40(3):496–546, 1998.
- [18] R. E. Burkard, V. G. Deineko, and G. J. Woeginger. The traveling salesman problem and the PQ-tree. *Mathematics of Operations Research*, 23:613–623, 1998.

- [19] J. Carlier and P. Villon. A new heuristic for the traveling salesman problem. *RAIRO - Operations Research*, 24:245–253, 1990.
- [20] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Research Report 388, Carnegie-Mellon University, Pittsburgh, PA, 1976.
- [21] R. K. Congram. *Polynomially searchable exponential neighborhoods for sequencing problems in combinatorial optimisation*. PhD thesis, University of Southampton, UK, 2000.
- [22] R. K. Congram, C. N. Potts, and S. L. van de Velde. An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem. *INFORMS Journal on Computing*, 14:52–67, 2002.
- [23] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, New York, NY, 1990.
- [24] M. Dror and M. Ball. Inventory/routing: Reduction from an annual to a short-period problem. *Naval Research Logistics*, 34:891–905, 1987.
- [25] M. Dror and P. Trudeau. Savings by split delivery routing. *Transportation Science*, 23:141–145, 1989.
- [26] Ö. Ergun. *New neighborhood Search Algorithms Based on Exponentially Large Neighborhoods*. PhD thesis, Operations Research Center, MIT, Cambridge, 2001.
- [27] Ö. Ergun and J. B. Orlin. Dynamic Programming Methodologies in Very Large Scale Neighborhood Search Applied to the Traveling Salesman Problem. Working Paper 4463-03, MIT Sloan School of Management, Cambridge, 2003.
- [28] Ö. Ergun, J. B. Orlin, and A. Steele-Feldman. A computational study on a large-scale neighborhood search algorithm for the vehicle routing problem with capacity and distance constraints. Working paper, School of Ind. and Sys. Engr., Georgia Institute of Technology, Atlanta, 2002.

- [29] Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. New York: Wiley, 3rd. edition, 1968.
- [30] L. Few. The Shortest Path and the Shortest Path through n Points. *Mathematika*, 2:141–144, 1955.
- [31] G. Gallo, G. Longo, S. Nguyen, and S. Pallotino. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42:177–201, 1993.
- [32] P. C. Gilmore, E. L. Lawler, and D. B. Schmoys. *The traveling salesman problem*, chapter Well solved special cases. John Wiley, Chichester, 1985.
- [33] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1994.
- [34] G. Gutin, A. Yeo, and A. Zverovitch. *The TSP and Its Variations*, chapter Exponential Neighborhoods and Domination Analysis for the TSP, pages 223–256. Kluwer Academic, 2002.
- [35] M. Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Math. Oper. Res.*, 10:527–542, 1985.
- [36] M. Haimovich, A. H. G. Rinnooy Kan, and L. Stougie. *Vehicle Routing, Methods and Studies*, chapter Analysis of Heuristics for Vehicle Routing Problems, pages 47–61. 1988.
- [37] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.
- [38] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, Plenum Press, 1972.
- [39] P. S. Klyaus. The structure of the optimal solution of certain classes of traveling salesman problems. *Vestsi Akad. Nauk BSSR, Phys. Math. Sci., Minsk (in Russian)*, pages 95–98, 1976.

- [40] M. W. Krentel. Structure in locally optimal solutions. *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 216–221, 1989.
- [41] E. L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.
- [42] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling salesman problem: a guided tour of combinatorial optimization*. Wiley, Chichester, 1985.
- [43] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *Handbooks in Operations Research and Management Science, Volumen 4: Logistics of Production and Inventory*, chapter Sequencing and Scheduling: Algorithms and Complexity, pages 445–522. North-Holland, New York, 1993.
- [44] C. L. Li and D. Simchi-Levi. Worst-case analysis of heuristics for the multi-depot capacitated vehicle routing problems. *ORSA J. Comput.*, 2:64–73, 1990.
- [45] L. Michel and P. Van Hentenryck. Localizer: A modeling language for local search. *INFORMS Journal of Computing*, 11(1):1–14, 1999.
- [46] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem. *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 402–408, 1996.
- [47] V. G. Deineko and G. J. Woeginger. A study of exponential neighborhoods for the Travelling Salesman Problem and for the Quadratic Assignment Problem. *Mathematical Programming, Ser. A*, 87:519–542, 2000.
- [48] J. B. Orlin, A. P. Punnen, and A. S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33:1201–1214, 2004.

- [49] C. H. Papadimitriou. The probabilistic analysis of matching heuristics. *Proc. 15th Annual Allerton Conference on Communication, Control, and Computing*, pages 368–378, 1977.
- [50] C. H. Papadimitriou and K. Steiglitz. On the complexity of local search for the traveling salesman problem. *SIAM Journal on Computing*, 6:76–83, 1977.
- [51] C. N. Potts and S. L. van de Velde. Dynasearch—iterative local improvement by dynamic programming. Part I. The traveling salesman problem. Preprint, Faculty of Mathematical Studies, University of Southampton, UK, 1995.
- [52] V. I. Sarvanov and N. N. Doroshko. The approximate solution of the traveling salesman problem by a local algorithm that searches neighborhoods of exponential cardinality in quadratic time. *Software: Algorithms and Programs, Mathematical Institute of the Belorussian Academy of Sciences, Minsk (in Russian)*, 31:8–11, 1981.
- [53] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.
- [54] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1996.
- [55] L. Stougie. *Design and analysis of algorithms for stochastic integer programming*. CWI Tract 37, Stichting Mathematisch Centrum, Amsterdam, 1987.
- [56] H.M.M. ten Eikelder and R.J. Willems. Some complexity aspects of secondary school timetabling problems. In *Proceedings of PATAT 2000*, volume 87, pages 18–27, 2001.