FACULTEIT ECONOMIE EN
BEDRIJFSWETENSCHAPPEN

KATHOLIEKE
UNIVERSITEIT
LEUVEN

ROUTING PROBLEMS WITH PROFITS AND PERIODICITY

Proefschrift Voorgedragen tot
het Behalen van de Graad van
Doctor in de Toegepaste
Economische Wetenschappen

door

Sofie COENE

NUMMER 310 2009

Committee

Prof. Dr. Frits C. R. Spieksma (Advisor)	*Katholieke Universiteit Leuven*

Prof. Dr. Carlo Filippi	*Universita' degli Studi di Brescia*
Prof. Dr. Peter Goos	*Universiteit Antwerpen*
Prof. Dr. Maurice Queyranne	*University of British Columbia*
Prof. Dr. Gerhard J. Woeginger	*Eindhoven University of Technology*
Prof. Dr. Willy Gochet	*Katholieke Universiteit Leuven*
Prof. Dr. Roel Leus	*Katholieke Universiteit Leuven*

Acknowledgments

Deze eerste pagina's zijn wellicht de belangrijkste omdat iedereen ze zeker leest. Terecht, want zonder de personen die in dit deel vermeld worden was dit werk nooit geworden wat het nu is. Dit alles heb ik enkel kunnen verwezelijken dankzij de hulp en steun van een aantal personen die ik bij deze dan ook van harte wil bedanken.

Eerst en vooral gaat mijn dank uit naar mijn promotor, Frits Spieksma. Dankjewel Frits, ik prijs mij erg gelukkig met jou als raadgever en begeleider. Dankjewel voor al de kansen die je mij gegeven hebt, de aangename werksfeer, de vele conferenties waaraan ik heb mogen deelnemen, mijn verblijf in Italië en de vrijheid om mijn weg te zoeken. Naast de tijd en energie die je in dit werk gestoken hebt, wil ik je vooral bedanken voor het vertrouwen dat je steeds in mij gehad hebt. Op die manier heb je het beste in mij naar boven weten te halen.

Daarnaast wil ik ook de leden van mijn doctoraatscommissie van harte danken. Hun opmerkingen en kritische vragen zijn dit werk zeker ten goede gekomen. In mijn studentenjaren heb ik reeds mogen ervaren dat Willy Gochet een groot statisticus is, maar ik heb de laatste jaren gemerkt dat ook operationeel onderzoek weinig geheimen kent voor hem. Het was dan ook een voorrecht hem onder de leden van mijn commissie te mogen rekenen.

Roel Leus weet als geen ander hard werken en ontspanning te combineren, hij heeft mij geleerd dat onderzoek zich niet enkel achter een bureau afspeelt. Ik heb hem leren kennen als een erg gepassioneerd onderzoeker, met veel oog voor detail, zonder daarbij de realiteit uit het oog te

verliezen. Hij heeft er dan ook voor gezorgd dat ik ondanks al die theorie de praktijk niet vergat.

Peter Goos was reeds van bij het begin bij dit project betrokken, en het is dan ook mede dankzij hem dat ik de kans gekregen heb dit te verwezenlijken. Zijn specialiteit ligt eerder bij statistiek dan bij operationeel onderzoek, en het is vanuit die optiek dat hij de hoofdstukken steeds grondig en kritisch heeft gelezen. Indien dit werk ook een beetje toegankelijk is voor niet-OR specialisten is dat vooral aan hem te danken.

I experienced what it is to take a course from Gerhard Woeginger, and, even though I was lost several times (my fault, not his), his passion for combinatorial optimization was really inspiring. He is a big name in the field and I feel very privileged and thankful for his original ideas and suggestions to this work.

I also wish to express my gratitude towards Maurice Queyranne. Already for many years he is a very respected member of the OR community and I am very thankful that he agreed to be part of my commission. His insightful comments and suggestions demonstrated that he read the manuscript thoroughly.

Il quarto capitolo è stato scritto in collaborazione con Carlo Filippi e Elisa Stevanato che ringrazio di cuore. Carlo mi ha accolto a Brescia per qualche mese, ed è stato un'esperienza molto preziosa. Grazie mille Carlo, per le numerose discussione e per aver letto la tesi molto attentamente.

Dankzij mijn collega's van "'t vijfde" heb ik de voorbije jaren in een erg aangename omgeving kunnen werken. Thank you Huijuan, you were the best office mate ever. You are not only a great colleague and chatting partner but also a very good friend; I wish you and your family all the best for the future. Grazie Fabrizio per le numerose chiacchiere in italiano. Dankjewel Sarah voor al de inspirerende gesprekken, al je goede raad en de vele uren in de klimzaal.

Ik heb de laatste 4 jaar in alle vrijheid kunnen werken mede dankzij de financiële steun van het Fonds voor Wetenschappelijk Onderzoek en de KULeuven, mijn dank daarvoor.

Verder wil ik ook al mijn goede vrienden en vriendinnen bedanken, en

in het bijzonder Laura, Saskia, en Paquita, omdat ik al zovele jaren altijd op jullie kan rekenen. Ook mijn familie ben ik heel erg dankbaar voor hun steun, de gezellige familiebijeenkomsten en voor al de kansen die ik steeds gehad heb; dankjewel Mama, Papa, Hannelore, Sarah en Jonas.

Tot slot rest er mij nog één erg belangrijke persoon te bedanken. Je geduld, vertrouwen, trots, vriendschap, maar bovenal je liefde hebben mij doorheen deze jaren geleid en gemotiveerd. Pepijn, dankjewel dat ik bij jou mag thuiskomen.

Sofie Coene Leuven, mei 2009.

Summary

This thesis deals with several routing problems with profit and/or periodicity component. More in particular, routing problems with a particular objective function and on specific network topologies are studied. Different concepts are introduced in Chapter 1. Special attention is paid to a customer-oriented routing problem known as the minimum latency problem or traveling repairman problem. The objective in this problem is to minimize total waiting time of the customers and thus maximizing service towards the customers. A setting is considered where not all customers need to be served, but serving a customer yields a certain profit for the server. In some settings customers require service regularly, as is the case in a periodic routing problem. On arbitrary graphs these problems are known to be NP-hard; looking at more restricted settings such as a line or a tree may yield polynomial time algorithms.

Chapter 2 deals with a minimum latency problem with profits (TRPP) on a line metric. This problem is NP-hard on a tree (Sitters (2002)); however, on the line, the problem can be solved in polynomial time by a dynamic programming algorithm. This dynamic programming algorithm can be extended to a problem where multiple identical servers are available. However, when servers are non-identical and there are release dates at the customers, the problem is NP-hard. Further, it can happen that deadlines are present at the customers. When there is a uniform deadline for all the customers the dynamic programming algorithm is no longer applicable and the complexity is not immediately clear. Arbitrary deadlines make the problem NP-hard. Finally, also the reverse problem is studied,

i.e. customers need to be visited as late as possible (but before a certain deadline).

In Chapter 3 a traveling salesman problem (TSP) with profits on a tree metric is considered. This problem is studied from a bi-objective point of view, meaning that at the same time costs are minimized and profits maximized. To deal with these multiple objectives the concept of Pareto-optimality is introduced. A solution with cost C and profit P is Pareto optimal if no other solution exists with $C' \leq C$ and $P' \geq P$. In this problem, not a single optimal solution is searched for but a set of optimal solutions. It is shown that on a tree metric, determining the set of Pareto optimal solutions of a TSP with profits is NP-hard. However, a pseudo-polynomial algorithm exists and the efficient set can be approximated efficiently by a FPTAS. When restricting the search to a subset of the Pareto set, i.e. the extreme supported efficient points, this set can be determined in polynomial time. These results still hold on more general graphs that satisfy the Kalmanson conditions.

In Chapters 2 and 3, focus is on complexity of routing problems with profits on specific graphs. In Chapter 4 a periodicity aspect is included in the periodic latency problem with profits. Customers require service on a regular basis according to their personal required frequency. A route is feasible if all customers in the route are visited according to their frequency. Complexity of this problem is studied in several settings for profits and frequency and in different metric spaces. It is shown that whenever a feasible solution exists, also a periodic feasible solution exists. The periodic latency problem with profits describes a periodic latency problem where not all customers need to be visited, the goal is to find a route for a single server collecting a maximal amount of profit. For arbitrary values of profits and frequencies the problem is solvable in polynomial time on the line and on the circle; on a star graph the problem becomes NP-hard. Only in cases where profits and periodicities have equal values for all customers the problem is solvable in polynomial time also on a tree. The multiple server periodic latency problem with profits and the periodic latency problem where the goal is to minimize the number of servers necessary to serve all

customers, are only solvable in polynomial time on the line and the circle. In all other situations they are NP-hard.

A more standard periodic routing problem is described in Chapter 5, dealing with a case study of a periodic vehicle routing problem (PVRP). The case described is a problem encountered by a Belgium transportation company responsible for collecting waste at slaughterhouses, butchers, and supermarkets. Two instances are studied, an instance with 262 customers and an instance with 48 customers. The problem is a PVRP with some additional constraints specific to the instances; all this is translated in a mathematical model. Due to the large amount of variables, however, heuristics are developed to compute good feasible solutions. Applying traditional strategies where first customers are assigned to days of the week and then routed or vice versa, routes are obtained gaining up to 15.5% in costs compared with the original routes.

A different periodic routing problem is discussed in the final chapter. Chapter 6 deals with a motion control problem for a placement machine. Such a placement machine consists of three main parts, i.e. a robot arm, a feeder, and a board, and all these parts can move. A set of components is positioned initially in the feeder and needs to be placed by the arm on the board. The different parts of the machine are routed such that all components are placed onto the board as fast as possible. Often this problem is solved using a (non-optimal) Greedy strategy. The problem, however, can be solved optimally in polynomial time by formulating it as a linear program (under the relevant Tchebychev metric). This LP yields a reduction in assembly time compared to the Greedy method.

Samenvatting

Dit proefschrift behandelt verschillende routeplanning problemen met een winst component en/of met periodiciteit. We focussen op een aantal specifieke doelfuncties en dit op verschillende netwerk topologieën. In Hoofdstuk 1 worden een aantal concepten toegelicht die centraal staan in dit werk. Deze bestaan uit 4 belangrijke elementen: klantgerichte routeplanning, winsten, periodiciteit en netwerk topologieën. In klantgerichte routeplanning tracht men de som van de wachttijden van de klanten te minimaliseren om zo een optimale service te garanderen. In dit werk komt een dergelijk probleem met winsten aan bod, i.e. met elke klant wordt een bepaalde winst geassocieerd. De leverancier is niet verplicht al de klanten te bedienen maar als hij een klant bedient, verzamelt hij de winst. In verscheidene toepassingen zal een klant regelmatig bediend worden met een bepaalde frequentie, dit wordt behandeld in periodieke routeplanning problemen. Bovenstaande problemen zijn NP-moeilijk wanneer algemene netwerken beschouwd worden. Beperken we ons echter tot klanten gelegen op een lijn of op een boom, kan in vele gevallen een polynomiaal algoritme ontwikkeld worden.

Hoofdstuk 2 handelt over het minimum latency probleem met winsten waarbij de klanten op een lijn liggen. Dit probleem op een boom is NP-moeilijk (Sitters (2002)), op de lijn echter kan dit probleem met polynomiale tijdscomplexiteit opgelost worden met behulp van een dynamisch programmeringalgoritme. Dit dynamisch programmeringalgoritme kan uitgebreid worden naar problemen met meerdere, identieke leveranciers. Het probleem met meerdere, niet-identieke leveranciers en klanten met “re-

lease dates" is NP-moeilijk. Ook het probleem waarbij deadlines aanwezig zijn komt aan bod. Het dynamisch programmeringalgoritme is niet langer bruikbaar, zelfs niet wanneer deze deadlines gelijk zijn voor al de klanten; de complexiteit van dit probleem is nog open. Willekeurige deadlines leiden tot NP-moeilijkheid van het probleem. We beschouwen uiteindelijk ook een omgekeerde situatie waarbij klanten zo laat mogelijk, maar vòòr een bepaalde deadline, bezocht wensen te worden.

De aandacht in Hoofdstuk 2 gaat vooral uit naar een latency probleem met winsten in een lijn metriek; in Hoofdstuk 3 bestuderen we een klassiek handelsreizigersprobleem maar met winsten en met klanten gepositioneerd op een boomnetwerk. Dit probleem wordt bestudeerd als een bi-objectief probleem: tezelfdertijd wordt getracht de kosten te minimaliseren en de winsten te maximaliseren. Om met deze meerdere objectieven om te gaan is het concept Pareto-optimaliteit belangrijk. Een oplossing met een bepaalde kost C en winst P is Pareto-optimaal als er geen andere toelaatbare oplossing bestaat waarvoor $C' \leq C$ and $P' \geq P$. Er wordt dus niet gezocht naar één enkele optimale oplossing maar naar een set van oplossingen. Het vinden van de Pareto-optimale oplossingen voor het handelsreizigersprobleem met winsten in een boom metriek is NP-moeilijk. Dit probleem is echter wel oplosbaar in pseudo-polynomiale tijd met behulp van een dynamisch programmeringalgoritme en de set van Pareto-optimale oplossingen kan efficiënt benaderd worden met een FPTAS. Een subset van de Pareto-optimale oplossing, i.e. de extreem ondersteunde efficiënte oplossingen, kan wel berekend worden door een algoritme met polynomiale tijdscomplexiteit. Deze resultaten blijven gelden wanneer we meer algemene netwerktopologieën beschouwen die voldoen aan de Kalmanson voorwaarden.

In Hoofdstukken 2 en 3 wordt de complexiteit bestudeerd van enkele routeplanning problemen met winsten op bepaalde netwerken. In Hoofdstuk 4 komt hierbij nog een periodiciteitaspect in het periodiek latency probleem met winsten. Elke klant dient regelmatig bediend te worden in overeenstemming met zijn/haar persoonlijke vereiste frequentie. Een route is enkel toelaatbaar indien al de klanten in de route in overeenstemming

met de vereiste frequenties bezocht worden. We bestuderen de complexiteit van dit probleem voor verschillende waarden van winsten en frequenties en op verschillende netwerken. Er wordt aangetoond dat indien er een toelaatbare oplossing bestaat, er ook een periodieke toelaatbare oplossing bestaat. We beschouwen drie varianten: periodieke latency met winsten, periodieke latency met winsten en meerdere leveranciers, en periodieke latency met meerdere leveranciers. In het periodieke latency probleem met winsten dienen niet al de klanten bezocht te worden en wordt een route gezocht voor de leverancier zodat de winst gemaximaliseerd wordt. Het probleem is oplosbaar in polynomiale tijd voor willekeurige waarden voor de winsten en frequenties indien het onderliggende netwerk een lijn of cirkel is; op een ster is het probleem al NP-moeilijk. Enkel indien winsten en frequenties gelijk zijn voor al de klanten is het probleem ook op een boom oplosbaar in polynomiale tijd. Het periodieke latency probleem met meerdere leveranciers is een analoog probleem waarbij voor elke leverancier een route gezocht wordt zodat totale winst gemaximaliseerd is. Het periodieke latency probleem zoekt routes waarbij al de klanten bediend worden met een minimaal aantal leveranciers. Beide problemen zijn enkel oplosbaar in polynomiale tijd op een lijn of een cirkel. In al de overige situaties zijn deze NP-moeilijk.

Een meer standaard periodiek routeplanning probleem wordt beschreven in Hoofdstuk 5. In dit hoofdstuk behandelen we een casus van een periodiek routeplanning probleem. De behandelde casus beschrijft een probleem van een Belgisch transportbedrijf verantwoordelijk voor het ophalen van slachtafval bij beenhouwers, supermarkten en slachthuizen. Er zijn 2 instanties gegeven, een instantie met 262 klanten en een instantie met 48 klanten. Het probleem is een voorbeeld van een PVRP met een aantal extra beperkingen specifiek voor deze instanties; dit wordt vertaald in een wiskundig model. Door het grote aantal variabelen richten we ons echter op heuristieken om een goede toelaatbare oplossing te bekomen. Enkele traditionele strategieën worden toegepast waarbij het probleem in twee fasen opgelost wordt. In een eerste fase worden klanten aan dagen van de week toegewezen en in een tweede fase worden de routes voor iedere

dag opgesteld; of omgekeerd. Op deze manier verkrijgen we routes die tot 15.5% goedkoper zijn vergeleken met de oorspronkelijke routes.

Tot slot wordt een heel ander periodiek routeplanning probleem bestudeerd in Hoofdstuk 6. In dit hoofdstuk beschouwen we een motion control probleem voor een plaatsingsmachine. Een dergelijke machine bestaat uit drie belangrijke onderdelen, i.e. een robot arm, een magazijn, en een bord; al deze onderdelen kunnen bewegen. Initieel bevindt zich een set van componenten in het magazijn; deze componenten worden door de arm op bepaalde posities op het bord geplaatst. De verschillende onderdelen van de machine dienen gerouteerd te worden zodanig dat al de componenten zo snel mogelijk op het bord geplaatst worden. Dit probleem wordt in de literatuur vaak opgelost met behulp van een (niet-optimale) Greedy methode. We kunnen dit probleem echter optimaal oplossen in polynomiale tijd door het te formuleren als een lineair programma (LP). Dit LP leidt tot een reductie in assemblagetijd vergeleken met oplossingen berekend met Greedy.

Table of contents

Chapter 1

Introduction

Routing problems have become very important the last 50 years; that is not only clear from the vast amount of research that has been done and is still being done on the subject. The number of hits on Google Scholar when typing "routing" amounts to 1200000 and in Google "routing" even yields 31900000 hits. Thus, not only in academics but also in our every day life we are confronted with routing problems. Routing does not only refer to vehicle routing, it also occurs on computer networks, the internet, telephone networks, in manufacturing environments and robotics.

Many have studied the vehicle routing problem (VRP) and its variants since Dantzig and Ramser introduced the problem in 1959. The variant with a single vehicle and unbounded capacity, i.e. the traveling salesman problem (TSP), was already studied in 1930 by Karl Menger (Menger (1932)). Only much more recently, especially since the paper of Afrati et al. (1986), more research has been done on a customer-oriented variant of the TSP, namely the traveling repairman problem (TRP), also known as the minimum latency problem (MLT). The latency objective (explained in 1.1) has turned out to be quite relevant in a variety of domains. It is especially suited for measuring customer service. In particular, in routing, apart from travel distance and travel costs, service to the customers can be an important aspect.

Consider for instance a school bus routing problem that was described

in a Master's thesis by Artois (2004). Given is a school for disabled children, each of whom needs to be picked up every morning, and brought home every evening. It would not be convenient nor fair if some children spend a very long time on the bus and others only a few minutes. The aim is to find routes for a set of busses such that the maximum travel time over the children is minimized. Optimal solutions for problems with such a customer-oriented objective tend to differ from solutions that are found by simply minimizing total travel time or distance. More generally, the influence of customer service on competitiveness has been clearly established, see e.g. McMullan and Gilmore (2008).

The title of the thesis consists of three parts: routing, profits, and periodicity. In this introductive chapter, section 1.1 further introduces the minimal latency problem, a problem informally described in the previous paragraph. Further, we introduce the two other keywords of this thesis, i.e. profits and periodicity, in sections 1.2 and 1.3 respectively. In section 1.4 some attention is paid to specific network topologies considered in this thesis, i.e. the line and tree network. Section 1.5 presents the outline of the remainder of this thesis.

1.1 The minimum latency problem

The minimum latency problem (MLT), also known as the traveling repairman problem (TRP), is defined as follows: given are a set of n customers, distances d_{ij} between each pair of customers i and j ($1 \leq i, j \leq n$), and a server traveling at unit speed. Then, the goal is to find a route for the server visiting all the customers such that total latency (i.e. waiting time) for the customers is minimized. The value of a solution to this problem is very different from the value of a solution to the classical traveling salesman problem (TSP), see the example depicted in Figure 1.1. In this example 4 customers request service and a single server is positioned at the depot. The route constructed in this example has a total cost of $a + b + c + d$ for the TSP, but for the MLT the cost would be much higher, namely $4 \times a + 3 \times b + 2 \times c + d$.

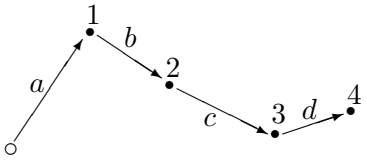


Figure 1.1: *Distance versus latency*

As mentioned in the introduction, an early paper on the MLT (or TRP) was published by Afrati et al. (1986); they developed a dynamic programming algorithm for the TRP where all customers are positioned on a line. Indeed, this is not a trivial problem as can be seen from the example in Figure 1.2. The distances between consecutive customers are indicated under the line. Intuitively, one expects the best route to be to travel first all the way to the left and back, or vice versa. However, in case of a latency objective that could yield a suboptimal solution. In the given example, traveling to the right and then to the left (y_1-y_2-y_3-y_4-x_1-x_2) yields a route with total latency equal to $6\times1+5\times99+4\times10+3\times10+2\times130+1\times190=1021$; vice versa ($x_1$-$x_2$-$y_1$-$y_2$-$y_3$-$y_4$) yields a route with total latency equal to $6\times10+5\times190+4\times201+3\times99+2\times10+1\times10=2141$. The optimal route for this example is first traveling to y_1, then returning to serve x_1, again changing direction to serve y_2 through y_4 and finishing in x_2 (y_1-x_1-y_2-y_3-y_4-x_2). Total latency is then $6\times1+5\times11+4\times110+3\times10+2\times10+1\times320=871$. Notice that distances traveled early in the route have a very large impact on the total cost of the route. Apart from Afrati et al. (1986),

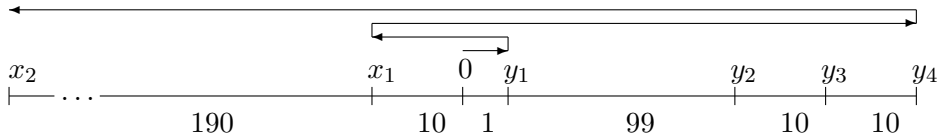


Figure 1.2: *Example MLT on the line*

also Minieka (1989), Wu et al. (2004), and García et al. (2002) studied exact algorithms for the MLT. Also, as MLT belongs to the class of NP-complete problems, many authors have focused on approximation algo-

rithms, see e.g. Blum et al. (1994), Goemans and Kleinberg (1998), Sitters (2002), Arora and Karakostas (2003). Recently, Salehipour et al. (2008) proposed a metaheuristic for the MLT. Few papers deal with the MLT with multiple servers (kMLT), examples are Fackaroenphol et al. (2007) and Jothi and Raghavachari (2007). Also in scheduling the latency objective is very common, the time-dependent TSP (TDTSP) is a scheduling problem where the cost of executing a job not only depends on the preceding job but is a non-decreasing function of the completion time, see e.g. Picard and Queyranne (1978).

In the remaining part of this section we propose a mathematical formulation for both the MLT and k-MLT.

A mathematical formulation for MLT Given are a set of n customers and a depot ($N = \{1, \ldots, n\} \cup \{0\}$), and a set of distances d_{ij}, $i, j \in N$. A single server travels at unit speed. x_{ijp} is a binary variable that is equal to 1 if customer j is visited after customer i at position p, 0 otherwise.

Minimize

$$\sum_{p=1}^{n} \sum_{i \in N} \sum_{j \in N} d_{ij}(n - p + 1) x_{ijp} \tag{1.1}$$

subject to

$$\sum_{j=1}^{n} x_{0j1} = 1 \tag{1.2}$$

$$\sum_{i=1}^{n} x_{ijp} - \sum_{i=1}^{n} x_{j,i,p+1} = 0 \qquad \forall p = 1, \ldots, n-1, \forall j \in N \tag{1.3}$$

$$\sum_{i=1}^{n} \sum_{p=1}^{n} x_{ijp} = 1 \qquad \forall j \in N \tag{1.4}$$

$$x_{ijp} \in \{0, 1\} \qquad \forall i, j \in N, \forall p = 1, \ldots, n \tag{1.5}$$

Exactly one arc leaves the depot to a point j that will be visited in position 1 (1.2). In every point j where an arc arrives at position p an arc also leaves point j to a point i that is then visited at position $p + 1$ (1.3).

Every customer is visited exactly once (1.4).

A set-partitioning formulation for k-MLT Again, a set of n customers and the inter-customer distances d_{ij} are given. Multiple servers (k) are available, all traveling at unit speed and with starting position at the depot. Consider the set R of all feasible routes, where a route $r \in R$ has an associated latency L_r. Variable x_r is equal to 1 if route r is selected, and 0 otherwise.

$$\text{Minimize} \sum_{r \in R} L_r x_r \tag{1.6}$$

$$\text{subject to} \sum_{r \in R_i} x_r = 1 \qquad \forall i \in N \tag{1.7}$$

$$\sum_{r \in R} x_r \leq k \tag{1.8}$$

$$x_r \in \{0, 1\} \qquad \forall r \in R \tag{1.9}$$

with $R_i = \{r \in R |$ customer i in route $r\}$, $1 \leq i \leq n$.
A set of at most k routes is selected such that total latency cost is minimized. Constraint (1.7) ensures that each customer is in exactly one selected route, and at most k routes can be selected due to constraint (1.8).

These formulations are relevant for chapters 2 and 4. In a column generation approach to solve (1.6) - (1.9), the pricing problem is equivalent to a minimum latency problem where not all customers need to be served. We define this problem as the minimum latency problem with profits, which is dealt with in chapter 2. Periodic latency problems are introduced in chapter 4.

1.2 Profits in routing problems

Routing problems with profits differ from standard routing problems in two important aspects: (i) a profit is included for visiting a customer and (ii)

not every customer must be visited. While many papers exist on routing where all customers need to be served – we refer among others to the book edited by Toth and Vigo (2002) and the recent book edited by Golden et al. (2008) – the number of papers on routing problems with profits is much more limited. In the survey of Feillet et al. (2005) on the traveling salesman with profits (TSPP) (notice that this is the single vehicle case with unbounded capacity), a distinction is made between three different cases, depending on the objective function. In the Orienteering Problem (OP) the goal is to maximize total collected profit given a constraint on the maximal cost or distance. In the prize-collecting TSP (PCTSP) on the other hand, the goal is to minimize total cost given a required minimal amount of profit to be collected. The objective in the Profitable Tour Problem (PTP) is a combination of both profits and costs. Only few papers describe a true bi-objective approach for the TSP, e.g. Keller and Goodchild (1988), Jozefowiez et al. (2008a), and Bérubé et al. (2009).

The vehicle routing problem with profits is dealt with in a number of papers. Chao et al. (1996) introduced the OP with multiple tours, which is a special case of the VRP with profits, under the name Team Orienteering Problem (TOP). Butt and Cavalier (1994) address the multiple tour maximum collection problem (MTMCP) in the context of recruiting football players from high schools. Aksen and Aras (2005) focus on the capacitated VRP with profits and time deadlines (VRPP-TD); the capacitated team orienteering and profitable tour problem is recently been dealt with by Archetti et al. (2008). An exact algorithm for TOP is described by Boussier et al. (2007). Jozefowiez et al. (2008b) recently published a survey on the existing research related to multi-objective optimization in routing problems.

Tricoire et al. (2008) describe a generalization of the TOP, i.e. the multi-period orienteering problem with multiple time windows (MUPOPTW). They present a problem where sales representatives have to visit customers periodically on a long-term basis and have to acquire new customers. This is to our knowledge the only paper dealing with profits and periodicity simultaneously.

In chapter 2 we deal with the traveling repairman problem with profits on the line where in the objective function profits and costs are combined, as in the PTP for the traveling salesman problem. In chapter 3 both objectives are considered simultaneously in a bi-objective approach for the TSP. In chapter 4 we introduce the periodic minimum latency problem with profits, both for a single server and for multiple servers. We elaborate further on periodicity in the next section.

1.3 Periodicity in routing problems

A routing problem cannot always be considered as a unique occurrence; often there is a certain periodicity involved, meaning that customers require service a number of times within a certain period of time T. Then, planning of the routes occurs over a time period of several days (or even several weeks) in which customers have different frequencies of visit. This problem is known as the Periodic Vehicle Routing Problem (PVRP). As in the traditional VRP, customer locations each with a given demand per visit are given, as well as a set of (capacitated) vehicles. Each customer requires a certain service level, that can be expressed as a number of visits over a time horizon T, as a maximal time lag between visits, or as a predefined visit pattern. The problem consists in designing vehicle routes that meet the given service requirements.

This problem was introduced by Beltrami and Bodin (1974) for the collection of municipal waste, and it was first defined as the "Period Routing Problem" by Christofides and Beasley (1984). Since then, many variants to the problem have been described in literature, for an overview see Mourgaya and Vanderbeck (2006) and the references contained therein. Apart from the collection of waste, many other applications occur, such as collection of raw materials for manufacturing companies (Alegre et al. (2007)), or maintenance of elevators (Blakely et al. (2003)). The periodic vehicle routing problem with service choice (PVRP-SC) (Francis et al. (2006)) allows more operational flexibility as it allows service frequency to be a decision of the model.

Periodic routing problems also occur in scheduling of robotic cells; Crama and van de Klundert (1997) investigate a cyclic scheduling problem for a robotic flowshop. The goal is to find a sequence of robot moves to maximize the throughput rate or minimize the cycle time of the system.

Chapter 5 contains a case study of a periodic vehicle routing problem. In this case study, a vehicle can perform multiple trips per day; a similar problem is described by Angelelli and Speranza (2002) as the periodic vehicle routing problem with intermediate facilities (PVRP-IF) and by Alonso et al. (2008) as the site-dependent multi-trip periodic vehicle routing problem (SDMTPVRP).

The problem dealt with in chapter 4, periodic latency problem (PLP), allows some flexibility concerning service frequency. Every customer has an upper bound on the period that may pass between two consecutive visits; within that period the server is free to visit the customer as often as it suits him. Further, the server is not required to serve all customers. Notice that this problem contains more flexibility in service frequency compared to the PVRP but less flexibility compared to the PVRP-SC.

In chapter 6 we describe a routing problem for different parts of a placement machine. This type of machines is used for the assembly of printed circuit boards (PCB). A movement pattern for the different parts of the machine is then repeated for a whole batch of identical PCB's.

1.4 Network topologies

Routing problems with profits and/or periodicity on arbitrary graphs are generally NP-hard. That is why in chapters 2, 3, and 4 we focus on complexity of these problems on easier network topologies. More specifically we study situations where customers are positioned on a straight line or on a tree network. Several routing applications exist where these networks are relevant. Psaraftis et al. (1990) describe "shoreline problems" where ships that visit ports along a shoreline are scheduled. Tsitsiklis (1992) describes an application where servers visit customers located along a highway. These problems can be extended to problems on a tree com-

bining the shoreline with the incoming rivers or including some side roads of the highway (Karuno et al. (1997)). Another line-problem is a setting where multiple elevators need to serve a set of requests (Friese and Rambau (2006)). Vehicle routing problems on tree networks have been studied in several papers, for instance, Labbé et al. (1991), Averbakh and Berman (1996), Muslea (1997), Lim et al. (2005) and Chandran and Raghavan (2008). Further, Karuno et al. (1997) consider routing problems in buildings with several floors connected by an elevator. The structure of each floor consisting of a corridor and rooms can then be represented as a subtree and the elevator joins all subtrees. This structure occurs often in hospitals, warehouses, hotels, and offices. This setting returns in chapter 4 where we describe a problem of setting up rounds for doctors and nurses in a hospital. A tree-shaped network is also considered in the stacker-crane problem, see (Coja-Oghlan et al. (2006)). Stacker cranes are automated item transportation devices that operate e.g. in warehouses. In chapter 2 we focus on problems on the line, in chapter 3 mainly problems on a tree metric are considered. In chapter 4 we study complexity of the periodic latency problems in different metric spaces, among which the line and the tree.

1.5 Thesis outline

Chapters 2 and 3 focus on routing problems with profits; chapters 5 and 6 deal with periodic routing problems. The "in between" chapter 4 combines profits and periodicity.

Chapter 2 focuses on a variant of the minimum latency problem. Profits are associated to customers and not all customers need to be served. The goal is to obtain a maximal amount of profit while profits at the customers go down linearly over time as long as the customer is not served. We refer to this problem as the traveling repairman problem with profits (TRPP). In general TRPP is NP-hard (this follows directly from NP-hardness of the TRP). However, when we restrict ourselves to the line as a metric space, we show that the TRPP on the line is solvable in polynomial time using a

dynamic programming algorithm. Further, we prove that TRPP on the line with multiple identical servers is also in P. When dealing with multiple non-identical servers and release dates the problem becomes strongly NP-hard. This result settles an open problem in de Paepe et al. (2004). Complexity of several other variants, including weights and repair times, is still open. This chapter has been published in Operations Research Letters (Coene and Spieksma (2008)).

Chapter 3 deals with a bi-objective traveling salesman problem where the underlying graph is a tree. Not all customers need to be served, however, serving a customer yields a certain profit. We analyze this problem from a bi-objective point of view, i.e. we simultaneously minimize cost and maximize profit. We show that finding all efficient points is NP-hard and we develop a fully polynomial time approximation scheme (FPTAS). For some specific cost and profit structures though, the efficient points can be determined in polynomial time. Further, the identification of extreme supported efficient points only takes $O(n^2)$. This chapter has been submitted. (Coene, Filippi, Spieksma, and Stevanato (2008b)).

In chapter 4, profits and periodicity are combined in the periodic latency problem with profits (PLPP). Each customer has an associated frequency and profit; the goal is to find a repeatable route for the server collecting a maximal amount of profits without violating the individual requested frequencies of the customers visited. We also consider a problem where all customers need to be served (PLP), the goal is to minimize the number of servers needed to serve all customers according to their frequency. In the periodic latency problem with multiple servers (MPLPP), both profits and multiples servers are combined. The goal is to find a route for each server maximizing total collected profit. We prove that when a feasible solution exists, it must hold that a periodic feasible solution exists as well. We give polynomial-time algorithms and NP-hardness results for these problems depending upon the topology of the underlying network. (Coene, Spieksma, and Woeginger (2009))

In chapter 5 we present a case study of a periodic vehicle routing problem. It concerns a routing problem of a Belgium company collecting waste

at slaughterhouses, butchers, and supermarkets. The problem is modeled using an integer programming formulation. The case consists of two instances with slightly different characteristics. We propose two solution approaches dealing differently with assigning customers to the days of the planning horizon and the routing component for every day in the planning horizon. The methods were implemented using ILOG Dispatcher. When comparing the resulting routes with the routes constructed by the company, considerable improvement was realized. This chapter has been published as a research report (Coene, Arnout, and Spieksma (2008a)).

In chapter 6 we deal with a motion control problem for a placement machine. We consider a machine that consists of 3 basic parts: a worktable, a feeder rack and a robot arm. Components to be assembled are positioned in the feeder rack, are picked up by the robot arm from the feeder rack and placed upon the worktable. The goal is to find movement patterns for each of this parts such that total time to assemble a printed circuit board (PCB) is minimized. A popular strategy to deal with this kind of problems is a greedy strategy. This strategy, however, does not always yield an optimal solution. We propose an LP formulation for this motion control problem and compare this solution strategy to Greedy. We show that assembly times can be reduced using the LP model. This chapter has been published in OR Spektrum (Coene, van Hop, van de Klundert, and Spieksma (2008c)).

Chapter 2

Profit-based latency problems on the line

The latency problem with profits is a generalization of the minimum latency problem. In this generalization it is not necessary to visit all customers, however, visiting a customer may bring a certain revenue. More precisely, in the latency problem with profits, a server and a set of n customers, each with corresponding profit p_i $(1 \leq i \leq n)$, are given. The single server is positioned at the origin at time $t = 0$ and travels with unit speed. When visiting a customer, the server receives a revenue of $p_i - t$, with t the time at which the server reaches customer i $(1 \leq i \leq n)$. The goal is to select customers and find a route for the server such that total collected revenue is maximized. We formulate a dynamic programming algorithm to solve this problem when all customers are located on a line. We also consider the problem on the line with k servers and prove NP-completeness for the latency problem on the line with k non-identical servers and release dates. In this proof we also settle the complexity of an open problem in de Paepe et al. (2004).

2.1 Introduction

Consider the following problem. Given are a set of n customers located in some metric space and profits p_i associated with each customer i, $1 \leq i \leq n$. In addition, a single server is given, positioned at the origin at time $t = 0$. The server travels at unit speed. If the server serves customer i at time t, the revenue collected by the server equals $p_i - t$. The goal is to select customers and to find a route for the server serving the selected customers, such that total collected revenue is maximal. We refer to this problem as the traveling repairman problem with profits, or TRPP for short. We restrict ourselves here to the line as a metric space. Notice that in the TRPP (i) not every customer needs to be served, and (ii) the revenue collected at a customer depends on the time needed to reach that customer.
In this chapter we show:

- how a dynamic program solves the TRPP on the line in polynomial time, thereby generalizing a classical result from Afrati et al. (1986),
- how this result can be generalized to the problem with multiple identical servers (referred to as MTRPP on the line),
- that the problem with multiple non-identical servers and release dates for each customer, is NP-hard.

In the proof of the latter result we settle the complexity of an open problem mentioned in de Paepe et al. (2004).
This chapter is organized as follows. In Section 2.2 we describe the literature for the traveling repairman problem and we motivate the TRPP on the line. In Section 2.3 we describe the dynamic programming algorithm. We settle the complexity of different variants of MTRPP in Section 2.4. Several observations concerning deadlines are discussed in Section 2.5. In the last section we state some open problems.

2.2 Literature and motivation

The TRPP is a generalization of the well-known traveling repairman problem (TRP), also known as the minimum latency problem (MLT). In this problem, no profits are given and the goal is to serve all customers with minimal total latency. Afrati et al. (1986) give an $O(n^2)$ dynamic program for the TRP on the line which was later improved to $O(n)$ by García et al. (2002). In Sitters (2004) it is proven that the TRP on the line with release dates is (weakly) NP-hard. Minieka (1989) shows that the problem on a tree network is polynomial for trees with unit weights and develops a polynomial time algorithm for the problem on weighted trees when the number of leaves is bounded. The TRP on weighted trees is proven to be strongly NP-hard by Sitters (2002). The problem on the line with multiple identical servers is solvable in $O(n^4)$, see Wu (2000) and Averbakh and Berman (1994). We refer to Wu et al. (2004) and the references contained therein for papers dealing with exact algorithms for the TRP (i.e. the problem with an arbitrary metric space).

In de Paepe et al. (2004) a framework is described dealing with the computational complexity of dial-a-ride problems (which include latency problems, and in particular latency problems on the line). However, as far as we are aware there is no work on latency problems with profits. This is in contrast with the situation for the traveling salesman problem (TSP), see Feillet et al. (2005) for a survey on the TSP with profits.

Motivation

As indicated above, a variety of latency problems arise in many different settings. However, a common characteristic is the focus on minimizing total waiting time of the customers. Here we consider a profit-oriented objective. In situations where a server receives some revenue by performing a service, this objective can be more appropriate. Of course, one needs to balance in some way the waiting times and the profits. This can be done in various ways (bounding the total waiting time by a constant from above, bounding the total revenue realized by a constant factor from below). Here,

however, we propose to combine profit and latency in a single objective (as described in the introduction). By doing so, it is clear that a latency problem with a profit objective is at least as hard as the corresponding "ordinary" latency problem.

We restrict the analysis in this chapter to a linear profit function (i.e. $p_i - t$) in the objective function. We use this particular function because it arises in the pricing problem of an integer programming formulation modeling the latency problem with multiple servers. Indeed, consider a situation with a set K with non-identical servers (meaning that their speed may differ) whose job is to service a set of n customers on the line. We now describe a set-partitioning formulation for this problem. We define c_{rk} as the total latency of a feasible route r ($r \in R$; with R the set of feasible routes) served by server k ($k \in K$). Then, variable x_{rk} is equal to 1 if route r is served by server k and 0 otherwise.

$$\text{Minimize} \sum_{r \in R} \sum_{k \in K} c_{rk} x_{rk} \tag{2.1}$$

$$\text{subject to} \sum_{r \in R_i} \sum_{k \in K} x_{rk} = 1 \qquad \text{for } i = 1, \ldots, n; \tag{2.2}$$

$$\sum_{r \in R} x_{rk} \leq 1 \qquad \text{for } k \in K; \tag{2.3}$$

$$x_{rk} \in \{0, 1\},$$

with $R_i = \{r \in R |$ customer i in route $r\}$, $1 \leq i \leq n$. The objective is to minimize total latency (2.1), there is a constraint for each customer stating that it must be served exactly once (2.2) (see also Friese and Rambau (2006)), and every server can be used at most once (2.3). When solving the linear programming relaxation of this integer program, the dual variables u_i ($i = 1, \ldots, n$) corresponding to constraints (2.2) act as the profits in the resulting pricing problem. Indeed, verifying for a fixed server k whether a violated dual constraint exists, amounts to minimizing $c_{rk} - \sum_{i: i \in r} u_i$. This corresponds to the objective function in the TRPP.

In this chapter we consider the line-metric. Practical examples where this metric is relevant are e.g. "shoreline"-problems (Psaraftis et al. (1990)).

These problems arise when scheduling and routing cargo-ships to visit a number of ports which are usually located along a shoreline. Friese and Rambau (2006) describe a setting with multiple elevators which need to serve a set of requests.
Summarizing, the TRPP is interesting because (i) it is a first attempt to combine a latency objective with profits, (ii) latency problems on the line are relevant and their complexity is often unresolved (de Paepe et al. (2004)), (iii) the TRPP occurs as the pricing problem of an integer programming formulation modeling the latency problem with multiple servers, as is explained above.

2.3 A polynomial algorithm for the TRPP

We represent an instance of the TRPP on the line as depicted in Figure 2.1. The server starts in the origin $x_0 = y_0 = 0$. In this section, we refer to the customers left of the origin as $x_1, x_2, \ldots, x_r$, and to the customers right of the origin as $y_1, y_2, \ldots, y_q$. To each customer a profit p_{x_i} resp. p_{y_j} (with $i = 0, \ldots, r; j = 0, \ldots, q$) is associated. Notice that we sometimes identify a customer with its position on the line. We make a distinction between "served" customers and "visited" customers. A customer is served when a server has performed a service at that customer and has collected profit there; when the server passes a customer without performing the service, this customer has been visited but not served. The goal is to select customers and to find a route for the server such that total profits of the customers served minus the latency of the corresponding route is maximal. Clearly, the TRPP on the line generalizes the traveling repairman problem on the line: in case each of the profits is huge, it is optimal to serve all customers, and the problem reduces to finding a route with minimal latency. In general however, not every customer needs to be served in the TRPP; observe however that, for those customers that are served, it is optimal to visit the customer the first time the server passes by. The optimal route thus has a spiral shape as illustrated in Figure 2.1 (see de Paepe et al. (2004)).

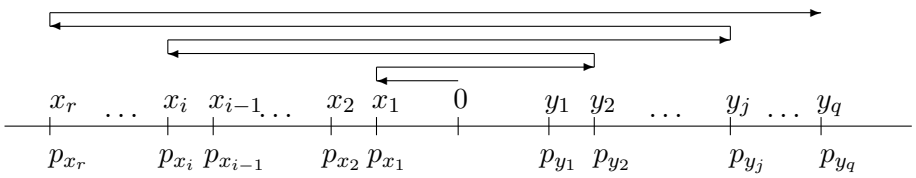


Figure 2.1: *The TRPP on the line*

Given an instance of TRPP it is not yet clear which customers, and in particular how many customers, need to be selected. We deal with this issue by proposing a procedure that keeps track of the number of customers to be visited.
We now define the ingredients of our dynamic program (DP). A state in our DP is denoted by $[x_i, y_j, l]$ which corresponds to the situation where the server is positioned in x_i (as its leftmost visited customer), where the rightmost visited customer is y_j, and where l customers are to be served outside that interval ($0 \leq i \leq r$, $0 \leq j \leq q$, $0 \leq l \leq r + q$). Similarly, in the state $[y_j, x_i, l]$ the server is at position y_j (as its rightmost visited customer), the leftmost visited customer is x_i, and l customers will be served outside this interval.

Definition 2.1.

$\forall i, j, l$: $P[x_i, y_j, l]$ ($P[y_j, x_i, l]$) equals the maximal value of the difference between

(i) the profits of the customers served in $[x_i, y_j]$ ($[y_j, x_i]$), and

(ii) the latency costs incurred for these customers served in $[x_i, y_j]$ ($[y_j, x_i]$) and for the l customers that will be served outside this interval.

We refer to $P[x_i, y_j, l]$ ($P[y_j, x_i, l]$) as the revenue.

Notice that the route followed by the server after having reached state

$[x_i, y_j, l]$ does not depend on the route followed within $[x_i, y_j, l]$. Thus, only the set of customers yielding a maximal value for $P[x_i, y_j, l]$ will be selected in an optimal solution. This observation is instrumental for the correctness of our DP algorithm, that we will now describe. First, we define $t[x_i, y_j]$ as the distance between customers x_i and y_j (for all i, j). In an initialization step we set the revenue in all states equal to $-\infty$; then we compute the revenue in a state as follows.

$$\text{For } l = 0, \ldots, r+q:$$

$$P[x_0, y_0, l] = P[y_0, x_0, l] = 0.$$

$$\text{For } i = 0, \ldots, r; j = 0, \ldots, q; l = 1, \ldots, r+q:$$

$$\begin{aligned} P[x_i, y_j, l] = \max\{ & \\ P[x_{i-1}, y_j, l+1] + p_{x_i} - (l+1) \times t[x_{i-1}, x_i], & \\ P[x_{i-1}, y_j, l] - (l) \times t[x_{i-1}, x_i], & \\ P[y_j, x_{i-1}, l+1] + p_{x_i} - (l+1) \times t[y_j, x_i], & \\ P[y_j, x_{i-1}, l] - (l) \times t[y_j, x_i] \quad \} & \end{aligned} \tag{2.4}$$

$$\begin{aligned} P[y_j, x_i, l] = \max\{ & \\ P[y_{j-1}, x_i, l+1] + p_{y_j} - (l+1) \times t[y_{j-1}, y_j], & \\ P[y_{j-1}, x_i, l] - (l) \times t[y_{j-1}, y_j], & \\ P[x_i, y_{j-1}, l+1] + p_{y_j} - (l+1) \times t[x_i, y_j], & \\ P[x_i, y_{j-1}, l] - (l) \times t[x_i, y_j] \quad \}. & \end{aligned} \tag{2.5}$$

Observe that $P[x_0, y_j, l] = P[y_0, x_i, l] = -\infty$ for $i > 0$ and $j > 0$. Then total revenue is:

$$TotalRevenue = \max\{\max_{i,j}\{P[x_i, y_j, 0]\}, 0\}.$$

Theorem 2.2. *Algorithm DP is correct.*

Proof. We establish correctness of (2.4) (the arguments for (2.5) are similar) by using induction on i for a fixed j, thereby proving Theorem 2.2. Correctness of (2.4) is shown by arguing that it leads to values for $P[x_i, y_j, l]$ that satisfy Definition 2.1. In case $i = 0$, we already observed that $P[x_0, y_j, l] = -\infty$ (which is in agreement with Definition 2.1).
We will use induction and assume that $P[x_{i-1}, y_j, l+1]$ and $P[x_{i-1}, y_j, l]$ satisfy Definition 2.1. The question now is whether $P[x_i, y_j, l]$ computed using (2.4) satisfies Definition 2.1. Consider the revenue realized when the server is positioned in x_i, while y_j is the rightmost visited customer. We distinguish two cases: x_i is served and x_i is not served. Consider first the case where x_i is served; as it is the last customer visited, it is the last customer served. Then, revenue can be broken down into three terms: (i) the revenue realized after serving a set of customers in $[x_{i-1}, y_j]$ when $l+1$ customers will be served outside this interval, (ii) the travel time between the previous customer visited and x_i, taking into account the $(l+1)$ customers left to be served, (iii) the profit p_{x_i}. The previous customer visited cannot lay outside $[x_i, y_j]$, in fact it is either x_{i-1} or y_j . Indeed, all customers within $[x_{i-1}, y_j]$ are met before x_i and it is optimal to visit them the first time the repairman passes by. Terms (ii) and (iii) are independent of the set of customers served and thus of the revenue realized in state $[x_{i-1}, y_j, l+1]$. As a consequence, only the set of customers yielding the maximal revenue in $[x_{i-1}, y_j, l+1]$ can lead to an optimal solution. It follows that the induction hypothesis tells us that the first term is accurately described by $P[x_{i-1}, y_j, l+1]$ or by $P[y_j, x_{i-1}, l+1]$. In addition, the second term equals $(l+1) \times t[z, x_i]$ where z denotes the previous customer visited (x_{i-1} or y_j), and finally the third term equals p_{x_i}. Now consider the case where x_i is not served. This means that only l customers are left to be served outside $[x_{i-1}, y_j]$ and no profit is realized in x_i. Revenue can be broken down into two terms: (i) the revenue realized after having served a set of customers in $[x_{i-1}, y_j]$ and l customers will be served out-

side this interval and (ii) the travel time between the previous customer visited (x_{i-1} or y_j) and x_i, taking into account the l customers left to be served. Again, term (ii) is independent of the set of customers served and thus the revenue realized in state $[x_{i-1}, y_j, l]$. As a consequence, only the set of customers yielding the maximal revenue in $[x_{i-1}, y_j, l]$ can lead to an optimal solution. It follows that the induction hypothesis tells us that the first term is accurately described by $P[x_{i-1}, y_j, l]$ or by $P[y_j, x_{i-1}, l]$. In addition, the second term equals $l \times t[z, x_i]$ where z denotes the previous customer visited.
By taking the maximum of the four terms considered above, we see that $P[x_i, y_j, l]$ equals its definition. A similar argument holds for $P[y_j, x_i, l]$. □

To estimate the complexity of DP with $n = r + q$: we have at most n^3 possible states, and since every state has 4 elements in its maximization function, the time complexity of the algorithm is $O(n^3)$, and we can state our result.

Corollary 2.3. *DP solves TRPP on the line in time $O(n^3)$.*

Let us consider a number of directions in which Corollary 2.3 can be extended. First, notice that the distances $t[x_i, y_j]$ need not be symmetric. Thus, situations where the line models a river with customers located at upstream and downstream positions are solvable by DP. Next, we can extend Corollary 2.3 to other geometries. Consider n customers and one server positioned on a circle. Observe that in an optimal solution there is a circle segment between two customers not traversed by the server. Hence, by considering $O(n)$ TRPP instances on the line we can solve the problem on the circle.
We can also extend our result to the TRPP when the customers are positioned on the endpoints of a tree with width three and with positive real lengths on the edges. The observation here is that, in any optimal solution, the spokes at either end of the tree are visited in increasing order of their

Figure 2.2: *A tree with width three*

lengths (Blum et al. (1994)). Figure 2.2 shows an example of such a tree with r customers on the left side and q customers on the right side of the tree.

A state in DP $[x_i, y_j, l]$ then represents the situation in which x_i is the longest spoke visited at the left side and the current position of the server, y_j is the longest spoke visited at the right side, l is the number of customers left to be served. Similarly, we can define state $[y_j, x_i, l]$ with the current position of the server being y_j and x_i the longest spoke visited on the other side. DP can now be run using the states as defined here.

Thirdly, Corollary 2.3 can be extended to the TRPP on the line with common release dates. In this variant of the TRPP, a release date M is associated with every customer, implying that a customer cannot be served before $t = M$. Notice that, if M is large, this is equivalent to choosing a starting location for the server, since then the server can travel and choose the position where it will start at time $t = M$. Of course, in any optimal solution the server, starting at the origin at $t = 0$, will at time M either be in M or in $-M$ or at one of the customers in between these two points and wait there until the release date is reached. Thus, the server has at most n choices for its starting position at time $t = M$ and again DP is able to solve this problem.

A fourth extension is the TRPP on the line with constant repair times. Suppose that there is a repair time h at every customer which is the same for all customers. This changes the revenue of a route only by a constant factor $\frac{1}{2}S(S+1)h$ where S is the total number of customers visited in the route. Thus, adding constant repair times has no influence on the computational complexity of the TRPP.

Finally, it is interesting to consider different profit functions in the objective, such as e.g. $p_i - t^2$. In DP, though, the waiting time t of a customer i is built up gradually in every iteration. In fact, one can argue that correctness of DP implies $f(x + y) = f(x) + f(y)$, which suggests that DP can only be applied in case of a continuous linear function f. A function for which this property holds is $p_i - w_i t$, with w_i the weight of customer i. DP deals with this weighted version of TRPP by making w_i copies of each customer i, each with a profit equal to $\frac{p_i}{w_i}$ and with a distance 0 from each other. We solve the resulting instance using DP. An optimal solution will have the property that either all or none of the copies of a customer i will be served. Notice that - assuming a binary encoding - this is no longer a polynomial time procedure, but pseudo-polynomial.

2.4 The complexity of MTRPP

In this section we discuss the TRPP with multiple servers (denoted by MTRPP). Given are the positions of n customers on the line and their associated profits p_i $(1 \leq i \leq n)$ and k servers, characterized by a speed s_j $(1 \leq j \leq k)$ and a starting location. The goal is to select customers and to find k routes serving each selected customer, such that total collected revenue is maximized. Recall that the revenue collected at a customer depends on the time needed to reach that customer. Observe that, if all servers are in the origin at $t = 0$ and if all servers have equal speed, the problem becomes trivial: one server travels to the left and a second server travels to the right, and when they meet a customer that contributes positively to the revenue, the customer is visited. In case the starting locations of the k servers are arbitrary (but given), and the servers have identical speed, we claim the following:

Theorem 2.4. *MTRPP on the line with multiple identical servers is solvable in polynomial time.*

Proof. We only give a short sketch of the proof since it is similar to the proof of Wu (2000) showing polynomial time solvability of the k-traveling repairmen problem on the line. First of all, notice that the servers will never pass by one another because they all have the same speed. A solution to the MTRPP on the line with identical servers thus consists in a division of the line into k consecutive intervals and solving the TRPP on the line for each interval. Consider a line with n customers positioned at $x_1, x_2, \ldots, x_n$ and k servers positioned at $z_1, z_2, \ldots, z_k$. An interval on this line is denoted by $I(i, j)$, containing customers $(i, i+1, \ldots, j)$ with $i \leq j$. Then, define $P(i, j)$ as the maximal revenue of a set of routes visiting customers $(i, i+1, \ldots, j)$ by the repairmen whose origins are within the interval $I(i, j)$. In a first phase compute $P(i, j)$ for each interval $I(i, j)$ containing exactly one origin. This can be done by computing revenue for each server k for its largest possible interval (i.e. the largest interval containing only server k) using DP. Then, the maximal revenue for every smaller interval containing only server k is also known. Indeed, DP computes revenue for all combinations of i and j with l equal to zero. Total time complexity of this first phase is thus $O(n^4)$. In a second phase we select one interval for every server. For $1 < i < k$ and $z_i \leq m < z_{i+1}$ we compute $P(1, m) = \max_{z_{i-1} \leq j < z_i} \{P(1, j) + P(j+1, m)\}$, and total revenue $P(1, n) = \max_{z_{k-1} \leq j < z_k} \{P(1, j) + P(j+1, n)\}$. Time complexity of the second phase is $O(n^2)$; total time complexity is thus dominated by the first phase and equal to $O(n^4)$. □

However, when the k servers have arbitrary speeds and customers have release dates $r_i \geq 0$, we claim that:

Theorem 2.5. *MTRPP on the line with non-identical servers and release dates is strongly NP-hard.*

We will prove NP-hardness for the MTRPP with non-identical servers and release dates by settling the complexity of an open problem mentioned in de Paepe et al. (2004), i.e. $Q|s = t, r_j|line|\sum C_j$. This problem is a latency problem ($\sum C_j$) on the line, with k nonidentical servers (Q) and

release dates (r_j); the phrase '$s = t$' refers to the fact that the customers only need to be visited (there is no transportation of customers). Notice that the customers do not have profits and each customer needs to be served. The goal is to find routes for every server such that total latency is minimal. Recall that the profit variant of this problem, MTRPP with non-identical servers and release dates, is a generalization of this problem and thus at least as hard.
We transform numerical matching with target sums (NMTS) to $Q|s = t, r_j|line|\sum C_j$.
In an instance of NMTS we are given positive integers x_i ($1 \leq i \leq m$), y_j ($1 \leq j \leq m$) and b_l ($1 \leq l \leq m$). The question is whether there exists a collection of m triples (i, j, l) such that (i) $x_i + y_j = b_l$ for each triple, and (ii) each integer in the input occurs exactly once. This problem is proven to be NP-hard by Garey and Johnson (1979).

Proof. We construct an instance of $Q|s = t, r_j|line|\sum C_j$ by specifying the speeds and the starting location of the servers, and the release dates and the location of the customers as follows. There are $k := m$ servers, the starting location of each server is the origin, and each server j has speed b_j (i.e. $s_j = b_j$, $j = 1, \ldots, m$). There are $2m$ customers, m customers are located to the left of the origin at $-x_i$, $1 \leq i \leq m$ (the "left" customers), and m customers are located to the right of the origin at y_j, $1 \leq j \leq m$ (the "right" customers). The release date of each left customer equals $M \equiv \max_i x_i$ (i.e., $r_i = M$). The release date of each right customer equals $M + 1$ (i.e., $r_j = M + 1$). The question is: does there exist a solution to $Q|s = t, r_j|line|\sum C_j$ such that total latency is equal to $mM + m(M + 1)$? This completes the description of an instance of $Q|s = t, r_j|line|\sum C_j$.
We now establish correspondence between the two questions. Clearly, if there exists a numerical matching with target sums, we can direct each server j to the appropriate left location, where it waits till the release date M, serves the customer at that location, travels to the appropriate right location (as given by the matching), arrives at time $M + 1$ (due to

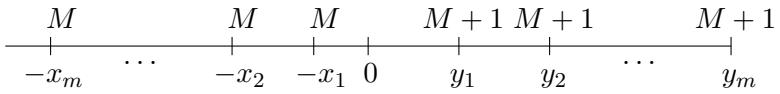


Figure 2.3: *MTRPP with release dates*

the existence of a matching) and serves the right customer. Total latency equals $mM + m(M + 1)$.
If there exists a solution with total latency $mM + m(M + 1)$, it follows that each customer must be served at its release date. This implies that the m servers have to be present at $t = M$ at the locations of the m left customers, and at $t = M + 1$ the servers need to be present at the m right customers. Hence, each server j travels from a unique left customer at $-x_i$ to a unique right customer at y_j in one time unit. It follows that NMTS has a solution. □

Corollary 2.6. *$Q|s = t, r_j|line|\sum C_j$ is strongly NP-hard.*

Notice that Yu et al. (2004) show that NMTS remains hard even when $x_i = i$ $(i = 1, \ldots, m)$ and $y_j = j$ $(j = 1, \ldots, m)$. It follows that the TRPP problem remains hard when the customers are one distance-unit apart.

2.5 Some observations

2.5.1 Deadlines

Consider the TRPP with a uniform deadline d, such that after a time period $t = d$, all profits drop to zero and traveling is no longer profitable. In this case elapsed travel time plays an important role and DP no longer yields an optimal solution. This is clear from the example in Figure 2.4. Profits are mentioned in the figure above the line, distances under the line; four customers request service and the server starts in the origin. The deadline is set to $d = 122$. The optimal value for $P[y_2, x_1, 1]$ is equal to 133 with corresponding route $0 - y_1 - x_1 - y_2$; total travel time is 122, which

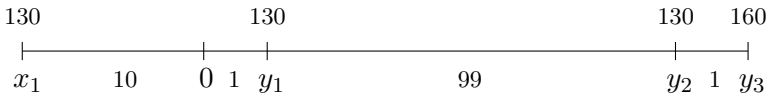


Figure 2.4: *Example TRPP with uniform deadlines*

is equal to d. It follows that $P[y_3, x_1, 0] = -\infty$ as $t > d$. When considering a suboptimal solution for $P[y_2, x_1, 1]$, namely route $0 - x_1 - y_1 - y_2$ with value 119 and total travel time 120, the optimal value for $P[y_3, x_1, 0]$ is found with a total value of 278 and $t = 121 < d$. Thus, our DP cannot be used and the complexity of this problem is not immediately clear.

The TRPP with arbitrary deadlines is a generalization of the minimum latency problem with deadlines and is thus at least as hard. This problem is proven to be NP-hard by Afrati et al. (1986) and they show that a pseudo-polynomial algorithm exists. Analogue as in section 2.3, this algorithm can be transformed to a pseudo-polynomial algorithm for the TRPP with deadlines. More specifically, this can be done by adding an entry to each state in the DP denoting the time elapsed.

2.5.2 The election problem in Chile

Suppose that in Chile new elections are coming up and there is, naturally, a candidate running for president. During his or her campaign, he or she will travel through the country (which has a long and narrow shape, reducible to a line) and visit a number of cities. However, the closer to the election date a city is visited, the more people in that city will stay convinced to vote for him/her. Thus, we are given a set of cities and a single server which starts traveling at some moment in time $t \geq 0$. At every city x_i, a revenue equal to t is collected, with t the number of time units passed before city x_i is reached. There is a deadline d (the election date) after which no more revenue can be collected. The metric space is restricted to the line. We distinguish between two cases depending on the value of d.

In the case that d is large enough, this election problem is equivalent to the minimum latency problem. Indeed, consider an instance of MLT

with n customers positioned on the line and an instance of the election problem with the same n positions on the line. A minimum latency tour, starting and ending in the origin, has a total cost of $C = \sum_i t_i$, with t_i the time elapsed before serving customer i. This tour in reverse order then is an optimal solution to the election problem with a total revenue of $P = \sum_i (d - t_i)$, with $d - t_i$ the time before reaching position i. Thus, "d large enough" means $d \geq t_{tot}$, with t_{tot} total travel time in the optimal solution to MLT. Notice that in MLT customers are served the earliest possible whereas in the election problem customers are visited the latest possible. As a consequence, customers are certainly served the first time the server passes by in the MLT while in the election problem this position will only be served the last time the server passes there. Notice that, in this case, we can add initial profits to the profit function of each city, i.e. revenue equals $p_i + t_i$. This may take the size of a city into account. Total profit is then equal to $P = \sum_i p_i + \sum_i (d - t_i)$.

When d is restrictive, meaning that $d < t_{tot}$, the problem can be shown to be equivalent to the TRPP. As before, we consider an instance of the election problem with revenue t for each customer i, with t the number of time units passed before reaching customer i; we define a variable T as the total travel time of the server and the server is free to determine its starting position. It is clear that in an optimal solution, the final customer visited will be served at time d and the first customer at time $d - T$. Consider an instance of the TRPP with the same set of customers and with each customer i having an initial profit $p_i = d$; the initial position of the server is to be decided by the algorithm (this adds a factor n to the complexity of DP). Then, in an optimal solution to the TRPP, the first customer visited yields a profit equal to d, and the final customer visited yields a profit equal to $d - T$. Thus, an optimal solution to the election problem with a restrictive deadline can be found by applying DP to the equivalent TRPP instance and reversing the obtained route. Clearly, it is also in this case possible to add an extra profit to take the size of a city into account: the revenue then equals $p_i + t_i$ for a client i. However, in the corresponding TRPP this means that initial profits at the clients are equal to $p_i + d$ and

after d time units they drop to 0; this is a TRPP with common deadlines which we cannot solve with DP, see supra.

2.6 Open problems

The complexity of the following latency problems on the line with profits is open at this moment:

(i) MTRPP with non-identical servers

(Notice also that the complexity of the problem without profits is still open, even if all the servers are positioned in the origin, see de Paepe et al. (2004)),

(ii) the weighted TRPP (we showed in Section 2.3 how DP can be used to solve this problem in pseudo-polynomial time; this however does not settle complexity of weighted TRPP),

(iii) it is unclear whether the $O(n)$ algorithm for the TRP in García et al. (2002) could be used for the TRPP. Potentially, this could give a speedup of the current $O(n^3)$ bound of DP.

Chapter 3

The traveling salesman problem on trees: balancing profits and costs

Traveling Salesman Problems with Profits (TSPP) are generalizations of the traveling salesman problem (TSP) where it is not necessary to visit all vertices. A profit is associated with each vertex. The goal consists in determining a route through a subset of nodes that simultaneously minimizes the travel cost and maximizes the collected profit. We analyze this problem from a bi-objective point of view, focusing on problems where the underlying graph has a tree metric. We show that finding all efficient points is NP-hard, and we propose a practical FPTAS for solving this problem. We also show that finding all extreme supported efficient points can be done in polynomial time. Finally, we show that finding a single supported efficient point is provably easier than finding all efficient points. Some special cases are considered where the particular profit/cost structure or graph topology allows the efficient points to be found in polynomial time.

3.1 Introduction

Traveling salesman problems with profits are bicriteria versions of the traveling salesman problem (TSP), where two opposite objectives need to be optimized, one pushing the salesman to travel (to collect profit associated with vertices) and the other inciting him or her to minimize travel costs (with the right to drop vertices). In this light, solving TSPs with profits should result in finding a set of feasible solutions such that neither objective can be improved without deteriorating the other. Usually these two optimization criteria appear either in a single objective function and/or as a constraint. For a survey on single-objective approaches see Feillet et al. (2005). An attempt to address the traveling salesman problem with profits as a bicriteria problem was made by Keller and Goodchild (1988), who refer to the problem as the multi-objective vending problem; their approach consists of sequentially solving single-objective versions of the problem. Recently Jozefowiez et al. (2008a) proposed a metaheuristic method to build up an approximate description of the efficient solution set and Bérubé et al. (2009) developed an exact approach. For an overview on multi-objective routing problems we refer the reader to Ehrgott (2000) and Jozefowiez et al. (2008b).

Relevance

In this chapter we consider the TSP with profits as a bi-objective optimization problem, focusing on the case where the underlying graph is a tree. It is clear that trees are fundamental network topologies, and many practical problems feature a tree as the underlying graph. More in particular, vehicle routing problems on trees have been discussed in, for instance, Labbé et al. (1991), Averbakh and Berman (1996), Muslea (1997), Lim et al. (2005), Chandran and Raghavan (2008), and the references contained therein. Motivation for the tree topology comes usually from transportation contexts where the underlying network is a railway network (in pit mines, for instance), a river network, or a sparse road network (in rural areas). Karuno et al. (1997) study the problem of scheduling and routing

a vehicle, such as an automated guided vehicle, in a building with a simple structure of corridors (each floor corresponds to a subtree and each room to a leaf vertex). In the works mentioned above it is required to visit each location. In this work, we relax this requirement and assume that a given profit is incurred when a customer is visited. Notice further that we deal here with a bi-objective problem featuring a single vehicle with unbounded capacity, i.e., featuring a traveling salesman.

Our Results

We notice that finding all efficient points (see Section 3.3) is NP-hard. In a quite general context, approximating the set of efficient solutions (also known as the Pareto curve) has been dealt with by Papadimitriou and Yannakakis (2000). Their work immediately implies the existence of a so-called FPTAS to find an ϵ-approximate Pareto curve for our problem. We exploit the analogy between our problem and a precedence constrained knapsack problem studied by Johnson and Niemi (1983) to revise a pseudo-polynomial dynamic programming approach proposed in Johnson and Niemi (1983) and adapt it to our problem. Then, we develop a simple FPTAS for our bi-objective problem, using ideas from Erlebach et al. (2002) for the multi-objective knapsack problem. For finding the set of extreme supported efficient points we propose a $O(n^2)$ algorithm, where n is the number of vertices in the tree, and we show that this is a lower bound. Finding one supported efficient point, corresponding to a given combination of the two objectives, takes only linear time. Thus, we prove that, when the graph is a tree, computing the set of all efficient solutions is more difficult than computing the set of all extreme supported efficient solutions (assuming $P \neq NP$), which in turn is proven to be more difficult than computing a single supported efficient point. Specific complexity classes exist for problems with multiple optimal solutions (T'kindt et al. (2005)); we extensively come back to this in Section 3.6.

The chapter is organized as follows. In Section 3.2 we define the three problems studied and we give some theoretical background on bi-objective optimization. In Section 3.3 we study the complexity of finding all effi-

cient points (referred to as Problem 1) and we develop a FPTAS for this problem. Some polynomially solvable special cases are also analyzed. In Section 3.4, a polynomial time algorithm is developed for finding the set of extreme supported efficient points (Problem 2), thereby also settling complexity of finding only one supported efficient point (Problem 3). In Section 3.5 we explain how we can extend our results to graphs satisfying the Kalmanson criteria. Section 3.6 deals with complexity classes specific for multi-objective problems. We finish with a conclusion and an overview of our results in Section 3.7.

3.2 The bi-objective TSP with profits on trees: three problems

Let $T = (V, E)$ be a tree where $V = \{0, 1, 2, \ldots, n\}$ is a set of $n + 1$ vertices and E is a set of edges. We consider vertex 0 (the depot) as the root. Let a profit p_i be associated with each vertex $i \in V$ and a cost c_{ij} be associated with each edge $(i, j) \in E$. We will assume that profits and costs are strictly positive, with the only exception that $p_0 = 0$. A *feasible subtour* $S = (V(S), E(S))$ of T is a circuit that starts and finishes at vertex 0, visits each vertex of $V(S) \subseteq V$ exactly once, and consists of the resulting edges $E(S) \subseteq E$. Notice that we distinguish between *visiting* a customer and *passing* a customer. A customer is only visited when the server collects profit at that customer and profit can only be collected once at every customer. Feasible subtours are then identified by subtrees containing vertex 0, where every subtree T' corresponds to a family of equivalent subtours, characterized by the order in which its vertices are visited. In every subtour corresponding to T', each edge is traversed twice, so the cost of the subtour is twice the sum of the costs of the edges of T' and the profit is the sum of the vertex profits. Notice further that we assume a given position of the salesman; this is relevant since, in contrast to the ordinary TSP, one does not need to visit all vertices. However, all positive results we state also hold for the case where one is allowed to

choose the salesman's position (at the expense of a factor n in the running times).

The cost of a feasible subtour S is then

$$c(S) = 2 \sum_{(i,j)\in E(S)} c_{ij},$$

whereas the profit of S is

$$p(S) = \sum_{i\in V(S)} p_i.$$

The goal is to find a feasible subtour that minimizes the total length of the tour and simultaneously maximizes the profit gained. We refer to Ehrgott (2005) and Hoogeveen (1992) for an introduction to multicriteria optimization. Here, we review basic terminology.

A feasible subtour S is *Pareto optimal* if there exists no feasible subtour S' such that $c(S') \leq c(S)$ and $p(S') \geq p(S)$, and at least one inequality is strict. A point $(\gamma, \pi) \in \mathbb{R}^2$ is an *efficient point* if there exists a Pareto optimal subtour S such that $c(S) = \gamma$ and $p(S) = \pi$. Let $\mathcal{E}$ denote the set of efficient points. In Figure 3.1 every point represents a feasible solution associated with a subtour S. The bold dots correspond to efficient points.

An efficient point $(\gamma, \pi) \in \mathbb{R}^2$ is *supported* if there exists a scalar $\lambda \in [0, 1]$ and a feasible subtour S such that S maximizes $\lambda p(S) - (1-\lambda)c(S)$ and $c(S) = \gamma$, $p(S) = \pi$. A supported efficient point that is an extreme point of the convex hull of all solution points, is called an *extreme* supported efficient point (Ehrgott (2005)). Let $\mathcal{SE}$ denote the set of extreme supported efficient points (see Figure 3.2). Clearly, $\mathcal{SE} \subseteq \mathcal{E}$.

The goal of our work is to investigate the difficulty of the bicriteria TSP with profits on trees. We distinguish the following three problems.

Problem 1 ($\mathcal{E}$) Find all efficient points and, for each of them, a corresponding Pareto optimal subtour.

Problem 2 ($\mathcal{SE}$) Find all extreme supported efficient points and, for each of them, a corresponding Pareto optimal subtour.

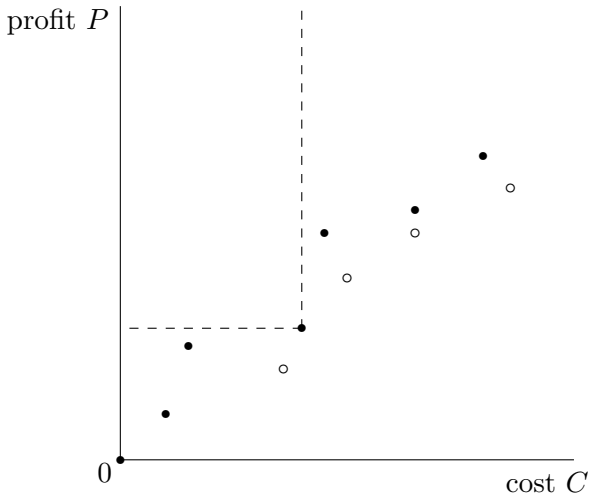


Figure 3.1: *Pareto Optimality*

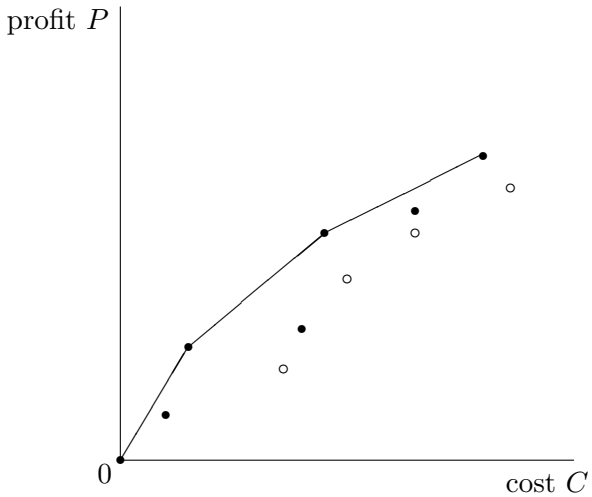


Figure 3.2: *Supported Set*

Problem 3 For a given value of $\lambda \in [0, 1]$, find an associated extreme supported efficient point and a corresponding Pareto optimal subtour. (Notice that this is a mono-objective problem.)

Although the topology of our setting is restricted, the questions are ambitious since —in Problems 1 and 2— we are not aiming for a single solution, but instead for a set of solutions. Of course, in order to do so in polynomial time, the number of points in the set should be polynomial in the input size. Notice that the above problems are sorted by decreasing complexity. Indeed, computing a single supported efficient point is not harder than computing the set of extreme supported efficient points, which in turn is not harder than computing the set of efficient solutions. We prove that there is, from a complexity point of view, a true difference among the three problems since it turns out that, assuming that $P \neq NP$, Problem 1 is more difficult than Problem 2. Also, we show that Problem 2 is in fact more difficult than Problem 3.

Problems 1, 2, and 3 are inherently related to some well-known problems in literature. The following three optimization problems with single objective are usually considered (see also Feillet et al. (2005)).

Orienteering Problem (OP) Find a feasible subtour of maximum profit among those with cost at most C, see, e.g., Golden et al. (1987).

Prize-Collecting TSP (PCTSP) Find a feasible subtour of minimum cost among those with profit at least P. This problem is originally defined by Balas (1989).

Profitable Tour Problem (PTP) Find a feasible subtour S maximizing the difference $p(S) - c(S)$. This problem is defined as PTP by Dell'Amico et al. (1995).

Notice that solving the PTP is equivalent to finding the supported efficient point corresponding to the combination of the two objectives with $\lambda = 1/2$. Thus Problem 3 is a slight generalization of PTP.

An algorithm for Problem 1 can be used to solve both OP and PCTSP. An algorithm for Problem 3 can be used to solve PTP. We then have the following.

Lemma 3.1. *If either OP or PCTSP are NP-hard then Problem 1 is NP-hard. If PTP is NP-hard then Problem 3 is NP-hard.*

The inverse connection between Problem 1 on the one hand and OP and PCTSP on the other hand is given through the following procedure, where it is assumed, without loss of generality, that the profits are integers.

Algorithm Mono-Bi-Objective

(i) initialize the list of Pareto optima with $S = \emptyset$ and the list of efficient points with $(0, 0)$; set $P_{tot} = \sum_{i \in V} p_i$ and $P = 1$;

(ii) find an optimal subtour S'' for the PCTSP with profit at least P;

(iii) find an optimal subtour S' for the OP with cost at most $c(S'')$;

(iv) append S' to the list of Pareto optima, and $(c(S'), p(S'))$ to the list of efficient points (notice that $c(S') = c(S'')$); if $p(S') < P_{tot}$ then set $P = p(S') + 1$ and go to step 2, else return the list of Pareto optima and the list of efficient points.

Algorithm Mono-Bi-Objective builds up the set of efficient points by solving $|\mathcal{E}|$ instances of PCTSP and $|\mathcal{E}|$ instances of OP. We thus have:

Theorem 3.2. *If all of the following conditions hold:*

(i) the number of efficient points is polynomial in the size of the input,

(ii) problem OP is polynomially solvable,

(iii) problem PCTSP is polynomially solvable,

then Problems 1 and 2 are polynomially solvable.

3.3 Problem 1 on trees

3.3.1 Complexity

We first observe here that, on trees, computing the set of Pareto optimal points (i.e., Problem 1) is NP-hard (Theorem 3.3), and that OP on a tree is equivalent to a knapsack problem with precedence constraints on the items.

Theorem 3.3. *Problem 1 on a tree T is NP-hard, even if T is a star.*

Proof. We show this by proving that the OP on a tree is NP-hard. It then follows from Lemma 3.1 that Problem 1 is NP-hard. Consider the knapsack problem, defined by a set of n items and a knapsack with capacity B. Each item $i \in \{1, 2, \ldots, n\}$ has associated an element weight w_i and a value q_i. The problem consists in finding a subset $S \subseteq \{1, 2, \ldots, n\}$ of maximum value $\sum_{i \in S} q_i$ among those with total weight $\sum_{i \in S} w_i$ not exceeding capacity B.
Now consider an instance of the OP on a tree consisting of $n + 1$ vertices and n edges such that each edge connects the origin (vertex 0) with a vertex i, $1 \leq i \leq n$, where each vertex (except the origin) represents a customer. Notice that the resulting graph is known as a star. Assign to each edge $(i, 0)$ a cost $c_{i0} := \frac{1}{2} w_i$, associate with each vertex $i \neq 0$ a profit $p_i := q_i$, and set the maximum allowable cost C equal to B. Recall that the OP consists in finding a subtour maximizing the total profit among those with total cost not greater than C. One easily verifies that there is a one-to-one correspondence between the solutions of the knapsack problem and the solutions of OP. □

The proof of the above result suggests in fact an equivalence between the knapsack problem and OP on a star. Thus, by using a dynamic programming approach for the knapsack problem, we can solve Problem 1 on a star in pseudo-polynomial time. It is then natural to ask whether the latter is true also for an arbitrary tree.

In order to give an answer, we resort to the *partially ordered knapsack* problem, see also Johnson and Niemi (1983) and Samphaiboon and

Yamada (2000). Partially ordered knapsack is a generalization of the knapsack problem that takes into account precedence relations between items. These precedence relations are modeled using a graph where each vertex corresponds to an item. Then, item i is a predecessor of j if there is an arc from i to j. An item can only be selected in the knapsack if all its predecessors have been included. In particular, if the graph representing the precedence relations is an *out-tree*, i.e., a directed tree where all arcs are oriented away from a distinguished root vertex, then we have an *out-tree knapsack problem*.

More precisely, let $T = (V(T), A(T))$ be an out-tree, and let a nonnegative value q_j and a nonnegative weight w_j be associated with every vertex $j \in V(T)$. Furthermore, let a positive capacity B be given. A vertex subset $V' \subseteq V(T)$ is called *closed under predecessor* if $j \in V'$ and $(i, j) \in A(T)$ imply $i \in V'$. The out-tree knapsack problem consists in finding a subset $V' \subseteq V(T)$ which is closed under predecessor, such that $\sum_{j \in V'} w_j \leq B$, and $\sum_{j \in V'} q_j$ is maximized. Johnson and Niemi (1983) proposed an efficient dynamic programming procedure that solves the out-tree knapsack problem in $O(nQ^\star)$ time, where $n = |V(T)|$ and $Q^\star$ is the optimal value of a solution. We observe the following.

Proposition 3.4. *The OP on a tree is equivalent to the out-tree knapsack problem.*

Proof. Consider an instance of the OP on a tree. There is a vertex i with an associated profit p_i $(i = 1, \ldots, n)$ and there is a depot, vertex 0. Each edge ij in the tree has a cost c_{ij}, and there is a maximum cost C. The corresponding out-tree knapsack problem has an item i for each vertex in the tree, with value $q_i := p_i$ and weight $w_i := 2c_{ji}$, where j is the unique predecessor of i in the oriented tree $(i = 1, \ldots, n)$. The budget B is equal to C. It is clear that a solution to the out-tree knapsack problem is equivalent to a solution of OP on the original tree and vice versa. □

A brute force approach for the solution of Problem 1 on trees is the following. First, solve an instance of OP, for every value of cost (capacity)

between 1 and $C_{tot} = 2\sum_{(i,j)\in A} c_{ij}$, by using Johnson and Niemi's algorithm. This results in a list of feasible solutions ordered in C. From this ordered list we can easily select the efficient points as follows. Go through the list in increasing order of C. Consider two neighboring points (γ', π') and (γ'', π''), with cost $\gamma' < \gamma''$. Notice that the resulting feasible solutions all have different costs. Then, if $\pi'' \leq \pi'$, eliminate (γ'', π'') and move to the next element in the list. If $\pi'' > \pi'$ immediately move to the next element in the list. Only the efficient points remain. Total time complexity is then $O(nC_{tot}P_{tot})$, which is pseudo polynomial. However, we can do better as is explained in the following section.

3.3.2 A dynamic programming algorithm for Problem 1 on trees

In this section we review the "left-right" dynamic programming algorithm by Johnson and Niemi (1983) for the out-tree knapsack problem, revised to fit Problem 1 on a tree.

Let $0, 1, 2, \ldots, n$ be a depth first ordering of the vertices of tree $T = (V, E)$, starting with the depot (the root). Let $d(i)$ be the number of children of node i, going away from the depot. Obviously, if the node i is a leaf then $d(i) = 0$.

For each $i \in V$ and $0 \leq s \leq d(i)$, we define $T[i, s]$ as the subtree of T induced by i, the first s children of i taken in order of index, all their successors, and all vertices in V with index lower than i (see Figure 3.3)

We define the left-right ordering of the subtrees as follows:

(a) $T[i, s]$ precedes $T[i, s + 1]$ for all $i \in V$ and $s \in \{1, \ldots, d(i) - 1\}$;

(b) if j is the sth child of i then $T[i, s - 1]$ precedes $T[j, 0]$ and $T[j, d(j)]$ precedes $T[i, s]$.

Notice that with the above ordering, every subtree contains all the subtrees that precede it. From the initial tree $T[0, 0] = \{0\}$ we gradually expand first down the left edge of the tree and then across the tree to the right. Notice further that some trees may be identical. More precisely,

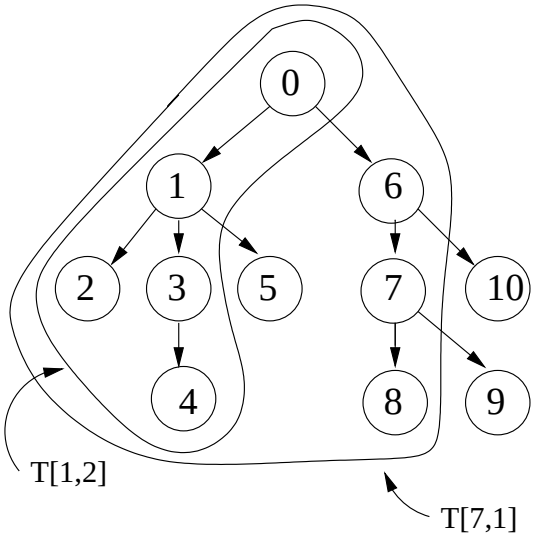


Figure 3.3: *Examples of subtrees in left-right approach*

if j is the sth child of i then $T[j, d(j)] = T[i, s]$. In other words, each time we have to backtrack while searching the tree in depth-first order, we replicate the same subtree. Anyway, we consider all defined subtrees as distinct objects. As a consequence, the total number of considered subtrees is exactly $2n + 1$.

A *q-subtour for* $T[i, s]$ is a feasible subtour that visits vertex i, cannot visit vertices not in $T[i, s]$ and has total profit equal to q. Notice that, in the previous example, although $T[j, d(j)] = T[i, s]$, a q-subtour for $T[j, d(j)]$ must contain j, while a q-subtour for $T[i, s]$ needs not.

Let again $C_{tot} = 2\sum_{(i,j)\in E} c_{ij}$ and $P_{tot} = \sum_{i\in V} p_i$. For each triple $[i, s, q]$ with $i \in \{0, 1, \ldots, n\}$ and $s \in \{0, 1, \ldots, d(i)\}$ we define $C[i, s, q]$ as the minimum cost of a q-subtour for $T[i, s]$. If there is no q-subtour for $T[i, s]$ then we define $C[i, s, q] = \infty$. This obviously happens for $q < 0$ and $q > P_{tot}$.

We compute function C by the following algorithm, which is a restatement of the left-right approach in Johnson and Niemi (1983).

Algorithm LR-DP

(i) (Initialization) $C[0,0,0] = 0$; $C[0,0,q] = \infty$ for $q = 1, 2, \ldots, P_{tot}$;

(ii) (Recursion) for all subtrees $T[i,s]$ sorted in left-right order:
for all $q = 0, 1, \ldots, P_{tot}$:

(a) if $s = 0$ then

$$C[i,0,q+p_i] = C[k,z-1,q] + 2c_{k,i} \tag{3.1}$$

where i is the zth child of k;

(b) if $s \in \{1, \ldots, d(i)\}$ then

$$C[i,s,q+p_i] = \min\{C[i,s-1,q+p_i], C[j,d(j),q+p_i]\} \tag{3.2}$$

where j is the sth child of i.

The recursion step can be explained as follows. In step (ii)(a), i is the largest element in the subtree and, by definition, i will be visited. Thus, the cost of this state is equal to the cost of a subtour with profit q in the largest subtree of T not containing i, increased with the extra cost of visiting i. In (ii)(b), either the subtour with profit $q + p_i$ visits the sth child of i, customer j, or it does not. If j is not visited then the cost of this subtour will be equal to the cost of a subtour with the same profit in a subtree not containing j. If customer j is visited, then the cost of this subtour is equal to the cost of a subtour (with the same profit) that by definition contains j. The minimum of both determines the cost of this state.

The advantage of algorithm LR-DP with respect to more intuitive dynamic programming procedures, based on a "bottom-up" search strategy of the tree, is that the evaluation of every entry of the array C takes a constant time. As a consequence, the complexity of LR-DP is linear in the number of entries in C, which is $(2n+1)P_{tot}$.

It is easy to verify that the values $C[i,s,q]$ returned by algorithm LR-DP satisfy the definition. As a consequence, the collection of ordered pairs

$$(C[0,d(0),0],0), (C[0,d(0),1],1), \ldots (C[0,d(0),P_{tot}],P_{tot}),$$

contains the efficient set $\mathcal{E}$. As in the previous section, we can easily select the efficient points from this ordered list.

In order to reconstruct the Pareto optimal subtours associated with the efficient points just computed, we keep track of the computations done in algorithm LR-DP by using a new function W with the same domain as C. More precisely, in step (ii)(a) of algorithm LR-DP, after computing the value of $C[i, 0, q + p_i]$, we set $W[i, 0, q + p_i] = (k, z - 1, q)$, where i is the zth child of k; in step (ii)(b), after computing the value of $C[i, s, q + p_i]$, if $C[i, s, q+p_i] = C[i, s-1, q+p_i]$ then we set $W[i, s, q+p_i] = (i, s-1, q+p_i)$, if $C[i, s, q + p_i] = C[j, d(j), q+p_i]$, where j is the sth child of i, then we set $W[i, s, q + p_i] = (j, d(j), q + p_i)$. For example, since the value of $C[1, 0, p_1]$ derives from $C[0, 0, 0]$, we set $W[1, 0, p_1] = (0, 0, 0)$.

At the end of algorithm LR-DP, starting from an entry of W corresponding to an efficient point (γ, π), we initialize a vertex subset $V' = \{0\}$. Then, we backtrack on the entries of W until we reach the entry equal to $(0, 0, 0)$. At each step, if we find an entry $(i, 0, q)$, then we add i to V'; if we find an entry $(j, d(j), q)$ then we add j to V'. When we reach $(0, 0, 0)$, V' contains the vertices that form the Pareto optimal subtour corresponding to (γ, π).

This procedure takes at most $O(n)$ steps for every efficient point; this implies that, in the worst case, we can compute all Pareto optimal subtours in $O(nP_{tot})$.

We summarize the above discussion by the following statement.

Theorem 3.5. *Problem 1 on trees can be solved in $O(nP_{tot})$ time.*

3.3.3 A FPTAS for Problem 1 on trees

It is well-known that the existence of pseudo-polynomial dynamic programs lead, under certain conditions, to the existence of polynomial time approximation schemes, see Woeginger (1999). From Papadimitriou and Yannakakis (2000) we know that such a polynomial time approximation scheme must exist for our bi-criterion setting. In this section we investigate how to approximate the efficient set in an effective way. We start with a

few definitions.

For $\epsilon \geq 0$, a pair (γ, π) is called an *ϵ-approximation* of a pair $(\gamma^\star, \pi^\star)$ if $\gamma \leq (1+\epsilon)\gamma^\star$ and $\pi \geq \pi^\star/(1+\epsilon)$. A set $\mathcal{E}'$ of points in the cost-profit space is called an *ϵ-approximation of the efficient set* $\mathcal{E}$ if, for every $(\gamma^\star, \pi^\star) \in \mathcal{E}$ there exists a point $(\gamma, \pi) \in \mathcal{E}'$ such that (γ, π) is an ϵ-approximation of $(\gamma^\star, \pi^\star)$. Notice that the closer ϵ is to zero, the better the approximation of the efficient set.

An algorithm that runs in polynomial time in the size of the input and that always outputs an ϵ-approximation of the efficient set is called an ϵ-approximation algorithm. A *polynomial time approximation scheme* (PTAS) for the efficient set is a family of algorithms that contains, for every fixed constant $\epsilon > 0$, an ϵ-approximation algorithm A_ϵ. If the running time of A_ϵ is polynomial in the size of the input and in $1/\epsilon$, the family of algorithms is called a *fully polynomial time approximation scheme* (FPTAS).

In this section we develop a FPTAS for Problem 1 on trees. We use the standard idea for developing a FPTAS for a knapsack problem, i.e., we scale the profits and apply an exact dynamic programming approach with the scaled profits. A FPTAS for the out-tree knapsack based on this idea is suggested in Johnson and Niemi (1983). However, such a scheme does not translate directly to a FPTAS for Problem 1 on trees. Indeed, in Johnson and Niemi (1983) a classical partitioning of the profit space into intervals of equal size is used, that guarantees a bound on the absolute error on every generated point. The bound is chosen so that when the maximum admissible cost (weight) is reached, the relative error ϵ is guaranteed. However, in order to get an ϵ-approximation of the efficient set, we require an algorithm that computes a feasible solution with a relative error ϵ for every possible cost and profit value. To this end, we use the partition of the profit space suggested by Erlebach et al. (2002) for the multi-objective knapsack problem.

We partition the profit space in u intervals:

$$[1, (1+\epsilon)^{1/n}), [(1+\epsilon)^{1/n}, (1+\epsilon)^{2/n}),$$

$$[(1+\epsilon)^{2/n}, (1+\epsilon)^{3/n}), \ldots, [(1+\epsilon)^{(u-1)/n}, (1+\epsilon)^{u/n})$$

with $u := \lceil n \log_{1+\epsilon} P_{tot} \rceil$. Notice that the union of all intervals generates the whole profit range $[1, P_{tot})$, and that u is of order $O(n(1/\epsilon) \log P_{tot})$[1], hence polynomial in the length of the input and in $1/\epsilon$. We can see that in every interval, the upper endpoint is $(1+\epsilon)^{1/n}$ times the lower endpoint. Then, we adapt algorithm LR-DP to the new interval profit space. We consider as profits the value 0 and the u lower endpoints of the intervals. For convenience, we denote by ℓ_w the lower endpoints, with $w = 1, 2, \ldots, u$, and we define $\ell_0 = 0$.

A *scaled q-subtour for* $T[i, s]$ is a feasible subtour that visits vertex i, cannot visit vertices not in $T[i, s]$ and has total profit *at least* q.

Instead of C, we consider a different function, denoted $\widetilde{C}$. For each triple $[i, s, \ell_w]$ with $i \in \{0, 1, \ldots, n\}$, $s \in \{0, 1, \ldots, d(i)\}$, and $w \in \{0, 1, \ldots, u\}$, we define $\widetilde{C}[i, s, \ell_w]$ as the minimum cost of a scaled ℓ_w-subtour for $T[i, s]$. If there is no scaled ℓ_w-subtour for $T[i, s]$, then we define $\widetilde{C}[i, s, \ell_w] = \infty$.

Algorithm Scaled-LR-DP

(i) (Initialization) $\widetilde{C}[0, 0, \ell_0] = 0$; $\widetilde{C}[0, 0, \ell_w] = \infty$ for $w = 1, 2, \ldots, u$;

(ii) (Recursion) for all subtrees $T[i, s]$ sorted in left-right order:
for all $w = 0, 1, \ldots, u$:
let $r = \max\{j : \ell_j \leq \ell_w + p_i\}$ (i.e., ℓ_r is the largest lower endpoint not greater than $\ell_w + p_i$);

(a) if $s = 0$ then

$$\widetilde{C}[i, 0, \ell_r] = \widetilde{C}[k, z-1, \ell_w] + 2c_{k,i} \tag{3.3}$$

where i is the zth child of k;

(b) if $s \in \{1, \ldots, d(i)\}$ then

$$\widetilde{C}[i, s, \ell_r] = \min\{\widetilde{C}[i, s-1, \ell_r], \widetilde{C}[j, d(j), \ell_r]\} \tag{3.4}$$

[1] $\log_a b = \frac{\log b}{\log a}$; $\lim_{x \to 0} \log(1+x) = x$

where j is the sth child of i.

(iii) (Output) return the points

$$(\gamma_w, \pi_w) = (\widetilde{C}[0, d(0), \ell_w], \ell_w) \qquad (w = 0, 1, \ldots, u).$$

In order to obtain the feasible subtours corresponding to the returned points in the cost-profit space, we may use the array W already described for algorithm LR-DP. Notice that in Scaled-LR-DP we directly return the last row of the array $\widetilde{C}$, containing only efficient points (in every state we calculate the cost when profit is equal or larger than ℓ_w).

Theorem 3.6. *Algorithm Scaled-LR-DP is a FPTAS for Problem 1 on trees with time complexity* $O(n^2(1/\epsilon)\log(P_{tot}))$.

Proof. This proof follows the lines used in Erlebach et al. (2002).

Let the subtrees be numbered according to the left-right ordering, i.e.,

$$T_0 = T[0,0],\ T_1 = T[1,0], \ldots,\ T_{2n-1} = T[0, d(0)].$$

We show that algorithm Scaled-LR-DP returns an ϵ-approximation of the efficient set. More precisely, we show the following claim.

Claim 1. *For every* $m \in \{1, 2, \ldots, 2n+1\}$*, let* $T_m = T[i, s]$*, and let* h *be the largest index of a vertex belonging to* T_m*. After performing the Recursion step on* $T[i, s]$*, for the optimal cost function* C *and the approximate cost function* $\widetilde{C}$ *there exists, for every entry* $C[i, s, q + p_i]$ *an entry* $\widetilde{C}[i, s, \ell_r]$ *such that:*

(a) $\widetilde{C}[i, s, \ell_r] \leq C[i, s, q + p_i]$,

(b) $(1 + \epsilon)^{h/n}\ell_r \geq q + p_i$.

Notice that when $m = 2n + 1$ then $T[i, s] = T[0, d(0)]$, $h = n$, and conditions (a) and (b) above imply that the point set returned by algorithm Scaled-LR-DP is an ϵ-approximation of the efficient set.

We prove the claim by induction on the tree index m.

The basis of the induction is given by $T_1 = T[1,0]$, where $h = 1$. We distinguish between two cases:

Case $q = 0$**.** We have:

$$C[1,0,p_1] = C[0,0,0] + 2c_{0,1} = 2c_{0,1} = \widetilde{C}[0,0,0] + 2c_{0,1} = \widetilde{C}[1,0,\ell_r]$$

where $\ell_r = \max\{j : \ell_j \leq p_1\}$. This proves that property (a) holds with equality.

Property (b) follows from the fact that p_1 and ℓ_r are in the same interval, hence:

$$(1+\epsilon)^{1/n}\ell_r \geq p_1$$

Case $q \geq 1$**.** We have:

$$C[1,0,q+p_1] = C[0,0,q] + 2c_{0,1} = \infty.$$

Let $r = \max\{j : \ell_j \leq q+p_1\}$ and $w = \max\{1, \min\{j : \ell_r \leq \ell_j + p_1\}\}$. Then, $r = \max\{j : \ell_j \leq \ell_w + p_1\}$ as required by step 2 of algorithm Scaled-LR-DP, and $\ell_w \geq 1$, so that

$$\widetilde{C}[1,0,\ell_r] = \widetilde{C}[0,0,\ell_w] + 2c_{0,1} = \infty.$$

Then property (a) holds. Furthermore, ℓ_r and $q + p_i$ are in the same interval, so that:

$$(1+\epsilon)^{1/n}\ell_r \geq q + p_1$$

This ends the basis of the induction.

Assume that the claim is true for any m, $1 \leq m < 2n+1$ and consider $T_{m+1} = T[i,s]$. Again, we distinguish between two cases.

Case $s = 0$**.** In this case the highest index in $T_{m+1} = T[i,0]$ is given by vertex i, then $h = i$. Property (a) follows from (3.1) and (3.3):

$$C[i,0,q+p_i] = C[k,z-1,q] + 2c_{k,i} \geq \widetilde{C}[k,z-1,\ell_w] + 2c_{k,i} = \widetilde{C}[i,0,\ell_r]$$

where the inequality holds by the inductive hypothesis and $r = \max\{j : \ell_j \leq \ell_w + p_1\}$. Now we consider property (b). By the induction hypothesis, the claim holds with $T[k,z-1]$, where $h = i-1$, then:

$$(1+\epsilon)^{(i-1)/n}\ell_w \geq q.$$

Hence we have:

$$\ell_w + p_i \geq q/(1+\epsilon)^{(i-1)/n} + p_i \geq (q+p_i)/(1+\epsilon)^{(i-1)/n}$$

and as ℓ_r and $\ell_w + p_i$ are in the same interval, with ℓ_r as lower bound, it holds that

$$(1+\epsilon)^{1/n}\ell_r \geq \ell_w + p_i \geq (q+p_i)/(1+\epsilon)^{(i-1)/n}.$$

From these relations property (b) follows immediately:

$$(1+\epsilon)^{i/n}\ell_r \geq q + p_i$$

Case $s > 0$**.** From (3.2) we have $C[i,s,q+p_i] = C[i,s-1,q+p_i]$ or $C[i,s,q+p_i] = C[j,d(j),q+p_i]$.

In the first case we have

$$C[i,s,q+p_i] = C[i,s-1,q+p_i] \geq \widetilde{C}[i,s-1,\ell_r] \geq \widetilde{C}[i,s,\ell_r]$$

for some lower endpoint ℓ_r, where the first inequality follows from the induction hypothesis and the second inequality follows from (3.4). This proves property (a). Concerning property (b), let k be the largest index of a vertex in $T[i,s-1]$. Clearly, $h > k$, so that

$$(1+\epsilon)^{h/n}\ell_r \geq (1+\epsilon)^{k/n}\ell_r \geq q + p_i$$

where the second inequality follows from the inductive hypothesis.

In the second case we have similarly

$$C[i,s,q+p_i] = C[j,d(j),q+p_i] \geq \widetilde{C}[j,d(j),\ell_r] \geq \widetilde{C}[i,s,\ell_r]$$

for some lower endpoint ℓ_r, where the first inequality follows from the induction hypothesis and the second inequality follows from (3.4). This again proves property (a). As already noticed, $T[i,s]$ and $T[j,d(j)]$ are in fact the same tree, so that the largest index h of a vertex in both trees is the same. Hence, property (b) follows directly from the inductive hypothesis.

From Theorem 3.5 and the bound on the number of different lower endpoints it follows that the complexity of Scaled-LR-DP is $O(n^2(1/\epsilon)\log(P_{tot}))$.

□

3.3.4 Some special cases

In this section we analyze some special cases of trees or cost/profit structures where Problem 1 is polynomially solvable.

Trees with equal profits or equal costs

We have proven that Problem 1 is hard, even on a star. However, Theorem 3.5 implies that in the special case where the sum of all the profits P_{tot} is polynomial in n, Problem 1 is polynomially solvable.

If all vertex profits are equal (but the edge costs are arbitrary), we may assume $p_i = 1$ for all $i \in V \setminus \{0\}$, so that $P_{tot} = n$. In this case, algorithm LR-DP solves Problem 1 in $O(n^2)$ time.

If all edge costs are equal (but the vertex profits are arbitrary), we may assume $c_{ij} = 1$ for all $(i, j) \in E$. In this case, the cost of a feasible subtour is twice the number of visited vertices. Hence there are at most $n + 1$ efficient points (exactly $n+1$ if all vertex profits are strictly positive). For all $i \in V \setminus \{0\}$, let $i^\star$ denote the parent of i along the unique path from i to 0. Let $M = \max_i\{p_i\} + 1$. We modify the edge costs and the vertex profits as follows:

$$\begin{aligned} c'_{ii^\star} &= -p_i + M \quad \text{for all } i \in V \setminus \{0\}, \\ p'_i &= 1 \qquad\qquad\ \ \text{for all } i \in V \setminus \{0\}. \end{aligned}$$

With the modified costs and profits we are back to the case of general costs and equal profits considered above. It easy to see that there is a one-to-one correspondence between the efficient points (and corresponding feasible subtours) in the modified problem and the original one. Thus Problem 1 in the case of equal costs and arbitrary profits on a tree can still be solved in $O(n^2)$ time by algorithm LR-DP.

The line

Let $T = (V, E)$ be a path, let the edge costs be arbitrary positive and let the vertex profits be arbitrary positive (except $p_0 = 0$). We distinguish between two cases.

The depot is an extreme vertex If the depot is an extreme point, we may assume that the vertices are numbered from left to right, so that the depot 0 is the leftmost vertex. Then it is easy to see that there are exactly $n+1$ feasible subtours (from the void subpath to the whole path) and they are all Pareto optimal. Furthermore, every feasible subtour corresponds to a different efficient point. A feasible subtour is identified by its rightmost vertex i.

In this situation, both OP and PCTSP can be solved in $O(n)$ time (Angelelli et al. (2008)). And in fact, the more general Problem 1 can also be solved in $O(n^2)$ time by algorithm Mono-Bi-Objective. However, the following algorithm solves Problem 1 in $O(n)$ time.

Algorithm Extreme Path

(i) set $\mathcal{E} = \{(\gamma_0, \pi_0)\} = \{(0,0)\}$;

(ii) for all $i = 1, \ldots, n$ set $\gamma_i = \gamma_{i-1} + 2c_{i-1,i}$ and $\pi_i = \pi_{i-1} + p_i$; append (γ_i, π_i) to $\mathcal{E}$.

Algorithm Extreme Path returns the ordered list of efficient points

$$\mathcal{E} = \{(\gamma_0, \pi_0), (\gamma_1, \pi_1), \ldots, (\gamma_n, \pi_n)\},$$

where the Pareto optimal subtour associated with (γ_i, π_i) is simply the subpath from 0 to i and back ($i = 0, 1, \ldots, n$).

The depot is an internal vertex If the depot is not an extreme point then Problem 1 is a little less straightforward to solve. We start by defining a lower bound on any algorithm solving this problem.

Theorem 3.7. *Problem 1 on the line can have $O(n^2)$ efficient points.*

Proof. Consider the following instance of Problem 1 on the line. We are given a set of n vertices on the line and one depot positioned at the origin. $\lfloor n/2 \rfloor$ vertices are positioned to the left of the origin and $\lceil n/2 \rceil$ vertices are positioned to the right of the origin. For all consecutive vertices i and

j to the left of the origin it holds that $c_{ij} = 1$ and $p_i = 1$. Let us define d_i as the distance from a customer i to the furthest left customer. For all customers j to the right of the origin it holds that $c_{ij} > d_i$, where i is an immediate predecessor of j. The profit of customer j, p_j, is larger than the sum of the profits of all customers on the left side of j. In this instance, every combination of a left customer and a right customer yields a pareto optimal solution. □

We now describe an algorithm solving this problem. Let the depot 0 be positioned in any point of the path, so in general there are n_L vertices on the left of the depot and there are n_R vertices on the right of the depot, with $n_L + n_R = n$. In this case, any feasible subtour S is just a subpath, containing vertex 0, and traversed twice, from 0 to a left vertex i, then from i to a right vertex j, and finally from j back to 0.

The total number of such subpaths becomes $n_L \cdot n_R \leq n^2/4 = O(n^2)$. We may distinguish three types of feasible subtours: those where the depot is the leftmost vertex, those where the depot is the rightmost vertex, and those where the depot is an internal vertex. Every subtour of the third type is obtained by gluing a subtour of the first type and a subtour of the second type.

In order to solve Problem 1, we then proceed as follows. In a first phase we evaluate the costs and profits of all feasible subtours of the first type and of the second type. In a second phase, we use this information to evaluate all feasible subtours of the third type. In a third and final phase, we build up the set of efficient points by sorting the feasible subtours and eliminating non-Pareto ones.

More precisely, we suggest the following algorithm:

Algorithm Internal Path

(i) call Extreme Path to generate the list $\mathcal{E}_R$ of points corresponding to feasible subtours of the first type;

(ii) call Extreme Path to generate the list $\mathcal{E}_L$ of points corresponding to feasible subtours of the second type;

(iii) set $\mathcal{P} = \emptyset$;

(iv) for all (γ', π') in $\mathcal{E}_R$ and (γ'', π'') in $\mathcal{E}_L$: insert in $\mathcal{P}$ points (γ', π'), (γ'', π''), and $(\gamma' + \gamma'', \pi' + \pi'')$ (the subtour correponding to the third point is obtained by gluing the subtours corresponding to the first and second points);

(v) sort all the points in increasing order of γ and go through this list, eliminating non-efficient points (as explained in the end of Section 3.2).

Step (i) requires $O(n_R) = O(n)$ time, step (ii) requires $O(n_L) = O(n)$ time, step (iv) requires $O(n_R \cdot n_L) = O(n^2)$ time; step (v) requires $O(n^2 \log n^2) = O(n^2 \log n)$ time. Hence algorithm Internal Path solves Problem 1 in $O(n^2 \log n)$ time. Recall from Theorem 3.7 that any algorithm solving Problem 1 on the line needs at least $O(n^2)$ time.

Problem 1 on a cycle

In case $G = (V, E)$ is a cycle, we assume that $E = \{(0,1), (1,2), \ldots, (n-1, n), (n, 0)\}$. We traverse the cycle *clockwise* if we go from 0 to 1, then from 1 to 2, and so on. There are four types of feasible subtours:

(i) tours going from 0 to i (≥ 0) clockwise and then coming back counter-clockwise;

(ii) tours going from 0 to i (> 0) counter-clockwise and then coming back clockwise;

(iii) tours going from 0 to i (> 0) clockwise, coming back counter-clockwise beyond 0 up to j ($> i$) and finally going back to 0 again clockwise;

(iv) the whole cycle.

Notice that in cases 1, 2 and 3 it is not necessary to travel further than half the total cost of the cycle, otherwise it would be better to visit the whole cycle. It is then possible to evaluate all tours and to build up the efficient set by modifying algorithm Internal Path in a straightforward manner.

3.4 Problem 2 on trees

In this section, we consider the identification of extreme supported efficient points. Problem 3 (i.e., finding just one supported efficient point corresponding to a given weighted sum of the objectives) can be solved in $O(n)$ time by using essentially the same algorithm proposed by Angelelli et al. (2008) for PTP on a tree. We show here that Problem 2 (i.e., finding all extreme supported efficient points) can be done in $O(n^2)$ time by solving a parametric linear program with a very special structure. A similar result is obtained by Hoogeveen (2005) who gives an example of a bicriteria problem where the number of efficient points is not polynomially bounded, but the number of supported solutions is. We also show that $O(n^2)$ is a lower bound on *any* algorithm for Problem 2.

Let $T = (V, E)$ be a tree, rooted at vertex 0 (the depot). For all $i \in V \setminus \{0\}$, let $i^\star$ denote the parent of i along the unique path from i to 0. For all $i \in V$, let $\delta(i) \subseteq V$ be the set of children of i; if i is a leaf then clearly $\delta(i) = \emptyset$. We associate with every vertex i a binary variable x_i, which equals 1 if and only if i belongs to a subtour. Notice that $x_i = 1$ implies $x_{i^\star} = 1$. Problem 2 on a tree corresponds to the following parametric program, where $\lambda \in [0, 1]$ is the parameter:

$$\begin{array}{rll} \max & \sum_{i \in V \setminus \{0\}} (\lambda p_i - 2(1-\lambda) c_{ii^\star}) x_i & \\ \text{subject to} & x_0 = 1 & \\ & x_i - x_{i^\star} \leq 0 & (i \in V \setminus \{0\}) \\ & x_i \in \{0, 1\} & (i \in V) \end{array} \tag{3.5}$$

Vertices are selected such that the sum of the weighted differences between the profits and costs is maximized. A feasible solution consists in a set of *connected* vertices which are also connected to the depot. This is enforced in the constraints in (3.5). The coefficient matrix of problem (3.5) is TUM, since it is the transpose of the node-arc incidence matrix of tree T, where all edges are oriented to the root. Thus we may relax the integrality constraints to nonnegativity constraints. The dual of the relaxed problem

is the following:

$$\begin{array}{rll} \min & y_0 & \\ \text{subject to} & y_0 \geq \sum_{j\in\delta(0)} y_j & \\ & y_i \geq (\lambda p_i - 2(1-\lambda)c_{ii^\star}) + \sum_{j\in\delta(i)} y_j & (i \in V \setminus \{0\}) \\ & y_i \geq 0 & (i \in V \setminus \{0\}) \end{array} \tag{3.6}$$

An optimal solution of problem (3.6) can be described as follows:

$$y_i(\lambda) = \max\{0; (\lambda p_i - 2(1-\lambda)c_{ii^\star}) + \sum_{j\in\delta(i)} y_j(\lambda)\} \quad (i \in V \setminus \{0\}) \tag{3.7}$$

Indeed, y_0 is minimized if the sum of the y_i is minimized; in an optimal solution y_i will never be bigger than needed by the model.
For any fixed value of λ, the unique solution of equations (3.7) can be computed in $O(n)$ time going backward from the leaves to the root. By fixing $\lambda = 1/2$ we get an algorithm for PTP, as proposed in Angelelli et al. (2008). However, we need to enumerate the solutions for all $\lambda \in [0, 1]$.

If $\sum_{j\in\delta(0)} y_j = y_0(\lambda) = 0$ then the optimal subtour is empty. Otherwise, by complementary slackness, an optimal subtour visits all vertices i where $y_i(\lambda) > 0$ and $y_j(\lambda) > 0$ for all ancestors of i. More precisely, consider the set-valued function $E^\star : [0, 1] \mapsto 2^E$, where $E^\star(\lambda) = \{(i, i^\star) \in E : y_i(\lambda) > 0\}$, and let $T(\lambda) = (V, E^\star(\lambda))$. For any given λ, an optimal subtour visits the vertices belonging to the connected component of $T(\lambda)$ containing the depot (vertex 0). Thus, in order to enumerate all supported subtours, it is sufficient to record the different values of function $E^\star$.

Suppose that we increase parameter λ continuously from 0 to 1. By expanding equation (3.7) recursively, one can see that $y_i(\lambda)$ is a nondecreasing, piecewise linear and convex function of the parameter λ, for all i. As a consequence, if $\lambda' < \lambda''$ then $E^\star(\lambda') \subseteq E^\star(\lambda'')$. Furthermore, $E^\star(0) = \emptyset$ (corresponding to the trivial empty tour) and $E^\star(1) = E$ (corresponding to the complete tour of all vertices). Thus, $E^\star$ changes its value in only $K \leq n - 1$ breakpoints, $0 < \lambda_1 < \lambda_2 < \ldots < \lambda_K < 1$.

In order to compute the breakpoints of $E^\star$, we proceed as follows. For any given $\lambda' \in (0, 1)$, let $R(\lambda') \subset V \setminus \{0\}$ be the set of the roots

of the connected components of $T(\lambda')$, excluding the depot. For all $i \in R(\lambda')$, let $V_i(\lambda')$ be the vertices of the corresponding connected component. Finally, let $\lambda'' > \lambda'$ be sufficiently close to λ'. By expanding equation (3.7) recursively, we see that

$$y_i(\lambda) = \max\{0; \sum_{j \in V_i(\lambda')} (\lambda p_j - 2(1-\lambda)c_{jj^\star})\} \qquad \text{for all } i \in R(\lambda'),\ \lambda \in [\lambda', \lambda''] \tag{3.8}$$

If we set $\alpha_i(\lambda') = \sum_{j \in V_i(\lambda')} p_j$ and $\beta_i(\lambda') = 2\sum_{j \in V_i(\lambda')} c_{jj^\star}$ then we may write (3.8) as

$$y_i(\lambda) = \max\{0; [\alpha_i(\lambda') + \beta_i(\lambda')]\lambda - \beta_i(\lambda')\} \qquad \text{for all } i \in R(\lambda'),\ \lambda \in [\lambda', \lambda''] \tag{3.9}$$

It follows that equations (3.9) are valid as long as

$$\lambda'' \leq \min_{i \in R(\lambda')} \left\{ \frac{\beta_i(\lambda')}{\alpha_i(\lambda') + \beta_i(\lambda')} \right\}$$

The right hand side of the above inequality is the next breakpoint.

We search for all breakpoints and the corresponding supported subtours by the following algorithm.

Algorithm Extreme Supported Efficient Points

(i) (Initialization)

  (a) $\mathcal{SE} = \{(0,0)\}$ (list of extreme supported efficient points), $\mathcal{S} = \{\{0\}\}$ (list of the vertex sets covered by the supported subtours), $R = V \setminus \{0\}$ (set of roots), flag = FALSE (if flag = TRUE then a new supported subtour has been found);

  (b) for all $i \in R$: set $\alpha_i = p_i$ and $\beta_i = 2c_{ii^\star}$, and set $V_i = \{i\}$;

(ii) (Breakpoint computation) for all $i \in R$:

  (a) set $\lambda_i = \beta_i/(\alpha_i + \beta_i)$;

  (b) let $\lambda_{\min} = \min_{i \in R}\{\lambda_i\}$ and $R_{\min} = \{i \in R : \lambda_i = \lambda_{\min}\}$;

(iii) (Connected components updating) for all $i \in R_{\min}$:

(a) set $\alpha_{i^\star} = \alpha_{i^\star} + \alpha_i$, $\beta_{i^\star} = \beta_{i^\star} + \beta_i$, and $V_{i^\star} = V_{i^\star} \cup V_i$;

(b) for all $j \in \delta(i)$: set $j^\star = i^\star$;

(c) remove i from R and from $\delta(i^\star)$;

(d) if $i^\star = 0$ then set flag = TRUE;

(iv) (Optimal subtour storing) if flag = TRUE then

(a) append (β_0, α_0) to $\mathcal{SE}$ and append V_0 to $\mathcal{S}$;

(b) flag = FALSE;

(v) (Termination test) if $R \neq \emptyset$ then go to Step (iii), else return $\mathcal{SE}$ and $\mathcal{S}$.

Theorem 3.8. *Algorithm Extreme Supported Efficient Points solves Problem 2 on a tree in $O(n^2)$ time.*

Proof. Correctness follows from the previous discussion. In particular, let $\lambda'_{\min}$ and $\lambda''_{\min}$ be two values of $\lambda_{\min}$ computed in two successive iterations of the algorithm. By construction, the vertex set V'_0 appended to $\mathcal{S}$ after the computation of $\lambda'_{\min}$ corresponds to an optimal subtour for all $\lambda \in [\lambda'_{\min}, \lambda''_{\min}]$, where $\lambda'_{\min} < \lambda''_{\min}$. Hence, V'_0 corresponds to a vertex of the convex hull of the efficient points.

Concerning complexity, at every iteration at least one vertex is removed from the set of roots, hence the iterations are at most n. The complexity of every iteration is $O(n)$. □

Example Consider the tree given in Figure 3.4; we are given a tree with 8 vertices, vertex 0 is the root node, the other vertices have a associated profit and distances are mentioned along the edges. We solve this example of Problem 2 using the algorithm described above. After the initialization step, all λ_i are calculated: $\lambda_1 = 0.5$, $\lambda_2 = 0.769$, $\lambda_3 = 0.471$, $\lambda_4 = 0.4$, $\lambda_5 = 0.286$, $\lambda_6 = 0.462$, $\lambda_7 = 0.5$. The vertex with lowest λ_i (vertex 5) is added to $R_{min} = \{5\}$ and removed from N. All children of vertex 5 are assigned to its predecessor, vertex 2, and the values of α_2 and β_2

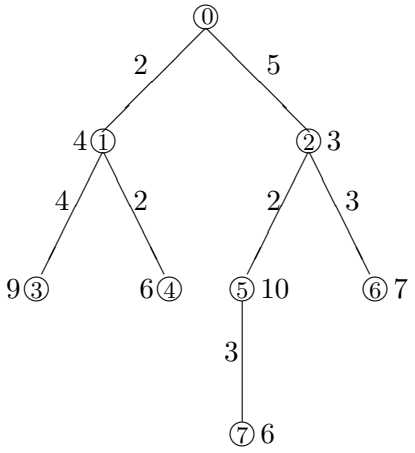


Figure 3.4: *Example: Problem 2 on a tree*

are adjusted: $\alpha_2 = \alpha_2 + \alpha_5$ and $\beta_2 = \beta_2 + \beta_5$. The new critical value $\lambda_2 = 0.519$ and the set $V_2 = \{5\}$ and the tree is adjusted as in Figure 3.5(a). Vertex 5 is not connected to the origin, thus, in step 2 the new λ_{min} is calculated: $\lambda_4 = 0.4$ is the next minimal value for λ_{min}. Its predecessor is vertex 1, such that λ_1 is now equal to 0.444 and $V_1 = 4$. The resulting tree is depicted in Figure 3.5(b). The next minimal λ_i is $\lambda_1 = 0.444$; its predecessor is the origin which means that we go trough step 4, where set $\{0, 1, 4\}$ is added to the set of supported efficient subtours, the resulting tree is 3.5(c). We continue in this way, vertex 6 is selected and $V_2 = \{5, 6\}$, see Figure 3.5(d). Then, $\lambda_3 = 0.471$ is minimal and a new supported efficient subtour $\{0, 1, 4, 3\}$ is found. Finally, in Figure 3.5(e) vertices 2 and 7, both with $\lambda = 0.5$ are selected, yielding a third supported efficient subtour $\{0, 1, 4, 3, 2, 5, 6, 7\}$. We found the set of the three extreme supported efficient solutions.

Theorem 3.9. *Solving Problem 2 on a tree has a $O(n^2)$ lower bound.*

We prove this theorem by showing that an instance of Problem 2 exists where representing the output takes $O(n^2)$. We provide an instance with $O(n)$ supported efficient solutions; each such a solution containing $O(n)$ elements.

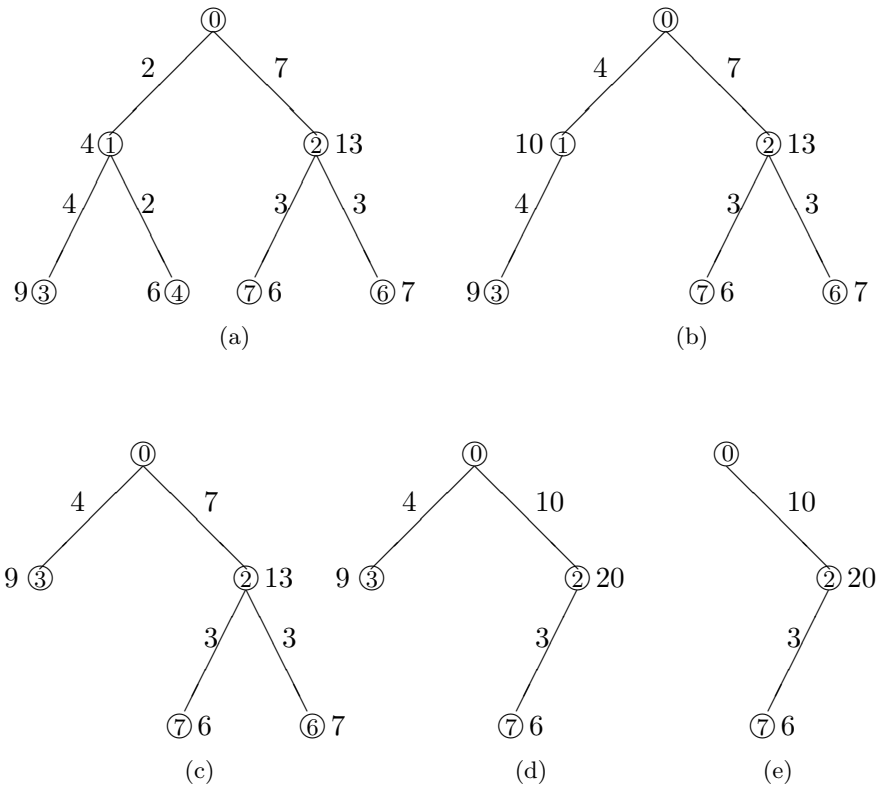


Figure 3.5: *Example (contd)*

Consider an instance of Problem 2 with n customers positioned on the n spokes of a star graph, each at a distinct distance c_{oi} from the center of the graph, i.e. the origin. Each customer has a profit $p_i = 1$. A solution to this instance of Problem 2 consists of a set of supported efficient points. Each supported efficient point represents a set, say S, of visited customers corresponding to a certain value $\lambda_{\max} = \max\{\lambda_i | i \in S\}$, with $\lambda_i = \frac{2c_{0i}}{1+2c_{0i}}$. Notice that $c_{0i} > c_{0j}$ if and only if $\lambda_i > \lambda_j$. Now for each pair of supported efficient points S_1 and S_2, with $|S_1| < |S_2|$, it must hold that $S_1 \subset S_2$. There is one point i in S_2 that is not in any $S_1, S_1 \subset S_2$, and for such a point i it holds that $\lambda_i > \lambda_j$ for all $j \in S_1$. Hence, for each supported

efficient point S_1 there is exactly one supported efficient point S_2 visiting exactly one element more which has a higher value for λ, and thus for c, compared to any element in S_1. It then holds that there are exactly n different supported efficient solutions and each supported efficient solution contains $O(n)$ elements. This means that the output of Problem 2 takes $O(n^2)$ □

We realize that the above argument relies on a particular format for representing the output. Other, more compact formats are conceivable. In particular, one might consider as an alternative format representing the set of all supported efficient subtours as a single sequence of vertices such that the i-th supported efficient solution consists of all vertices in the sequence up to position i. However, we claim here that, even under this format, a sorting step is necessary to find this sequence.

Notice that we can solve Problem 2 on the line in $O(n)$. On a line it holds that if you visit a vertex i, you also visited all customers between i and the origin. Thus, λ_i must be at least equal to $\frac{2c_{0i}}{\sum_{0\leq j\leq i} p_j + 2c_{0i}}$, for all $0 \leq i \leq n$. Now, at each side of the origin it holds that if $c_{0i} > c_{0j}$ and $\lambda_i < \lambda_j$, then customer j will only be visited in an efficient subtour if customer i is visited as well. Thus, those customers can be merged into one customer with $\lambda = \lambda_i$. Finally, select the remaining customers in increasing order of λ in order to form the supported efficient subtours.

3.5 Extension: Kalmanson matrices

In this section we show that our results can be extended to problems where the distance matrix is a Kalmanson matrix. An $n \times n$ distance matrix C is a Kalmanson matrix if the following conditions are satisfied:

$$c_{ij} + c_{kl} \leq c_{ik} + c_{jl} \qquad \forall 1 \leq i < j < k < l \leq n, \tag{3.10}$$

$$c_{il} + c_{jk} \leq c_{ik} + c_{jl} \qquad \forall 1 \leq i < j < k < l \leq n. \tag{3.11}$$

These conditions unite two special cases of distance matrices, i.e. trees and convex point sets in the Euclidean plane (Klinz and Woeginger (1999)).

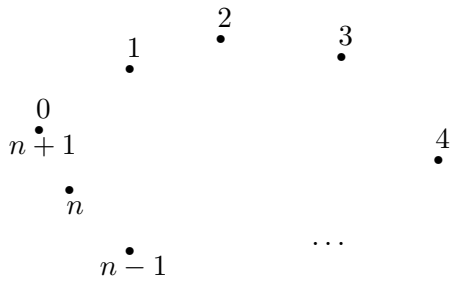


Figure 3.6: *Convex point set*

We can extend our results for Problems 1, 2, and 3 to distance matrices satisfying the Kalmanson conditions. Deĭneko et al. (1995) show that checking whether an ordering of nodes exists such that the resulting matrix is a Kalmanson matrix can be done in $O(n^2)$ time.

3.5.1 Problem 1 on Kalmanson matrices

Problem 1 can be solved in pseudo-polynomial time when the distances satisfy the Kalmanson conditions. Consider an instance of Problem 1 with n customers and an $n \times n$ distance matrix C, with C a Kalmanson matrix, more specifically we consider a convex point set for ease of interpretation.

Kalmanson (1975) proved optimality of the TSP tour $\langle 1, 2, \ldots, n-1, n \rangle$ in case the distance matrix fulfills the conditions (3.10) and (3.11).

Theorem 3.10. *Problem 1 on graphs satisfying the Kalmanson conditions can be solved in $O(n^2 P_{tot})$ time.*

Proof. We develop a pseudo-polynomial dynamic programming algorithm that solves Problem 1 when the distances satisfy the Kalmanson conditions. Consider for instance a convex point set with n points representing positions of n customers. Points 0 and $n+1$ represent the origin (see Figure 3.6). Each customer i has an associated profit p_i and the distance between two customers i and j is given by c_{ij}. In an optimal solution it holds that if customer i is served, it is served before any customer $j > i$. Indeed, suppose a server serves customers i, $i+1$, $i+2$, and j ($j > i$) in

sequence $\langle i, i+2, i+1, j\rangle$ with total cost $c_{i,i+2} + c_{i+1,i+2} + c_{i+1,j}$. This will never be better then visiting the customers in sequence $\langle i, i+1, i+2, j\rangle$ which has a cost of $c_{i,i+1} + c_{i+1,i+2} + c_{i+2,j}$. Due to the Kalmanson conditions it holds that $c_{i,i+1} + c_{i+2,j} \leq c_{i,i+2} + c_{i+2,j}$. Thus, an optimal tour is always a subsequence of the optimal TSP-tour and only arcs (i, j) with $i < j$ are used. We then define $C[k, q]$ as the minimum cost for serving customer set $S \subseteq \{0, 1, \ldots, k\}$ with total profit equal to q and customers 0 and k served. If no such q-subtour exists, $C[k, q] = \infty$. We compute function C by the following algorithm.

(i) (Initialization) $C[0, 0] = 0; C[0, q] = \infty$;

(ii) (Recursion) for all $k = 0, 1, \ldots, n+1$; for all $q = 0, 1, \ldots, P_{tot}$:

$$C[k, q + p_k] = \min_{0 \leq l < k} \{C[l, q] + c_{lk}\} \tag{3.12}$$

The collection of ordered pairs

$$(C[n+1, 0], 0), (C[n+1, 1], 1), \ldots, (C[n+1, P_{tot}], P_{tot})$$

then contains the efficient set $\mathcal{E}$. In this algorithm $O(nP_{tot})$ states are computed and each state takes $O(n)$, yielding a total time complexity of $O(n^2 P_{tot})$. Notice that the algorithm described above works for any distance matrix satisfying the Kalmanson conditions, not only for convex point sets. □

Analogue as for Problem 1 on trees a FPTAS can be derived from this dynamic programming algorithm.

3.5.2 Problem 2 on Kalmanson matrices

In this section we provide a parametric IP formulation for Problem 2 on convex point sets and we propose a polynomial time algorithm. Let $G = (V, E)$ be a convex point set with $V = \{0, 1, 2, \ldots, n, n+1\}$, where vertices 0 and $n+1$ both represent the origin, and $E = \{(i, j) | i, j \in V, j > i\}$ (recall that only edges (i, j) with $i < j$ are used in an optimal subtour). The costs

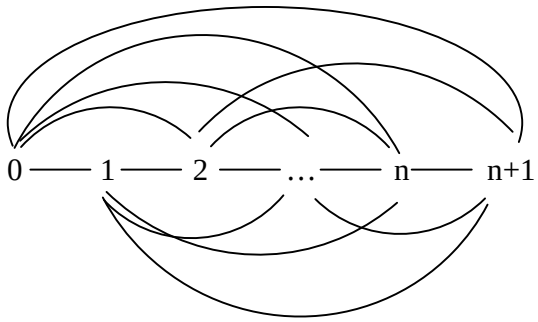


Figure 3.7: *The network*

of edges $(i, j) \in E$, are given by c_{ij} and the profits associated with each $i \in V \backslash \{n+1\}$ by p_i. We set $p_0 = 0$ and $c_{0,n+1}$=0. Recall that, given that the Kalmanson conditions hold, vertices are visited in increasing order of their index. Then, the following parametric program models Problem 2 on a convex point set, with parameter $\lambda \in [0, 1]$.

$$\begin{array}{rll} \min & \sum_{i=0}^{n} \sum_{j=1}^{n+1} (\lambda c_{ij} - (1-\lambda) p_i) x_{ij} & \\ \text{subject to} & \sum_{i=1}^{n+1} x_{0i} = 1 & \\ & \sum_{j=1}^{n+1} x_{ij} - \sum_{j=0}^{n} x_{ji} = 0 & (i \in V \setminus \{0, n+1\}) \\ & x_{ij} \in \{0, 1\} & (i, j \in V) \end{array} \tag{3.13}$$

A feasible solution consists in a connected route from 0 to n+1. Model (3.13) is equivalent to a network flow model, more specifically a shortest path problem on an acyclic graph, it follows that the coefficient matrix is TUM; we may thus relax the integrality constraints and solve the resulting LP model. For a given value of λ, the optimal solution can be found easily by solving this LP. Problem 3 can thus be solved in polynomial time. Finding the set of optimal solutions for all $\lambda \in [0, 1]$, comes down to solving a parametric shortest path problem. Figure 3.7 represents the network, the cost on each edge (i, j) is given by $d_{ij} = \lambda c_{ij} - (1-\lambda) p_i = \lambda(c_{ij} + p_i) - p_i$. For $\lambda = 0$, the shortest path is $\langle 0, 1, 2, \ldots, n, n+1 \rangle$ with total cost $C(0) = -\sum_{i=1}^{n} p_i$. However, for $\lambda = 1$, the shortest path is $\langle 0, n+1 \rangle$, and total cost is $C(1) = c_{0,n+1} = 0$. We claim the following.

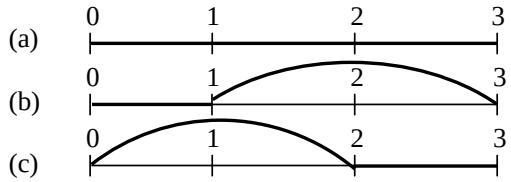


Figure 3.8: *Representation proof Proposition 3.11*

Proposition 3.11. *For the parametric shortest path problem in (3.13) it holds that if customer i is not visited in an optimal solution for λ_j, customer i will also not be visited in any optimal solution for $\lambda \geq \lambda_j$.*

Proof. Suppose that Proposition 3.11 does not hold. Then a vertex i exists for which $x_{ij} = 0$, $\forall j \in V$, for λ_i and there exists a $\lambda \geq \lambda_i$ for which $\exists j \in V$ such that $x_{ij} = 1$ in an optimal solution. This situation is depicted in Figure 3.8. In this figure only 4 vertices are represented, however any situation as described above can be reduced to Figure 3.8. The first and last vertex represent the origin and have profit $p_0 = 0$, vertices 1 and 2 have profits p_1 and p_2, respectively. The costs on the edges are given by $d_{ij} = \lambda(c_{ij} + p_i) - p_i$. The route in Figure 3.8 (a) visits all vertices, this situation is optimal for all $0 \leq \lambda \leq \lambda_a$; the route in Figure 3.8 (b) is optimal for any λ for which $\lambda_a \leq \lambda \leq \lambda_b$; the route in Figure 3.8 (c) is then optimal for any λ for which $\lambda_b \leq \lambda \leq \lambda_c < 0$. In (a) all vertices are visited and total cost $C(\lambda) = f_a(\lambda) = \lambda(c_{01}+c_{12}+c_{23}+p_1+p_2)-(p_1+p_2)$, in (b) vertex 2 is not visited and total cost is $C(\lambda) = f_b(\lambda) = \lambda(c_{01}+c_{13}+p_1)-(p_1)$, and in (c) vertex 2 is again visited but vertex 1 is not and $C(\lambda) = f_c(\lambda) = \lambda(c_{02} + c_{23} + p_2) - (p_2)$. In an optimal solution it holds that $C(\lambda) \leq 0$ and functions $f_a(\lambda)$, $f_b(\lambda)$, $f_c(\lambda)$ are linear in λ. Then, for $f_a(\lambda) = f_b(\lambda) = f_c(\lambda) = 0$ it must hold that $\lambda_{f_a} \leq \lambda_{f_b} \leq \lambda_{f_c}$, thus yielding the following expressions for λ_{f_a}, λ_{f_b} and λ_{f_c}.

$$\lambda_{f_a} = \frac{p_1 + p_2}{c_{01} + c_{12} + c_{23} + p_1 + p_2} \tag{3.14}$$

$$\lambda_{f_b} = \frac{p_1}{c_{01} + c_{13} + p_1} \tag{3.15}$$

$$\lambda_{f_c} = \frac{p_2}{c_{02} + c_{23} + p_2} \tag{3.16}$$

Then,

(i) $\lambda_{f_a} \leq \lambda_{f_b}$ if

$$\begin{aligned} \frac{p_1 + p_2}{c_{01} + c_{12} + c_{23} + p_1 + p_2} &\leq \frac{p_1}{c_{01} + c_{13} + p_1} \\ p_2 &\leq p_1 \frac{c_{12} + c_{23} - c_{13}}{c_{01} + c_{13}} \end{aligned}$$

(ii) $\lambda_{f_b} \leq \lambda_{f_c}$ if

$$\begin{aligned} \frac{p_1}{c_{01} + c_{13} + p_1} &\leq \frac{p_2}{c_{02} + c_{23} + p_2} \\ p_1 \frac{c_{02} + c_{23}}{c_{01} + c_{13}} &\leq p_2 \end{aligned}$$

(i) & (ii)

$$\begin{aligned} p_1 \frac{c_{02} + c_{23}}{c_{01} + c_{13}} &\leq p_1 \frac{c_{12} + c_{23} - c_{13}}{c_{01} + c_{13}} \\ c_{02} + c_{13} &\leq c_{12}, \end{aligned} \tag{3.17}$$

which contradicts condition (3.11). □

$C(\lambda)$ is a piecewise linear and convex function of the parameter λ. Let $S^*(\lambda)$ be the set of served customers for parameter λ; S^* only changes at the breakpoints of $C(\lambda)$. Thus, in order to enumerate all supported subtours, it is sufficient to search for the subtours associated to the breakpoints of $C(\lambda)$. We know that $C(0) = -\sum_{i=0}^{n} p_i$ with $S^*(0) = \{0, 1, 2, \ldots, n+1\}$ the set of served customers and $C(1) = 0$ with $S^*(1) = \{0, n+1\}$. Due to Proposition 3.11 it follows that, for increasing λ, vertices will gradually

"drop out" of the path until for $\lambda = 1$ only the depot remains. Thus, for $\lambda' < \lambda''$, $S^*(\lambda') \supseteq S^*(\lambda'')$ and the value of S^* only changes at $k \leq n$ breakpoints. We develop an algorithm that can determine at which critical values of λ the optimal path changes. A direct arc (i, j) enters the optimal path (and the vertices between i and j drop out of the path), when

$$\sum_{k=i}^{j-1}(c_{k,k+1} + p_k)\lambda - \sum_{k=i}^{j-1} p_k = (c_{ij} + p_i)\lambda - p_i.$$

This yields a critical value of λ^*_{ij} for arc (i, j).

$$\lambda^*_{ij} = \frac{\sum_{k=i+1}^{j-1} p_k}{\sum_{k=i}^{j-1}(c_{k,k+1} + p_k) - c_{ij} - p_i};$$

for $\lambda = \lambda^*_{ij}$ arc (i, j) enters the shortest path and vertices $i + 1, \ldots, j - 1$ drop out of the solution.

Algorithm Parametric Shortest Path

(i) (Initialization)

(a) List of extreme supported efficient points $\mathcal{SE} = \{(P_{tot}, C_{tot})\}$,(for $\lambda_0 = 0$); list of covered vertices $S = \{0, 1, 2, \ldots, n, n + 1\}$, and the set of edges $\bar{E} = \{(i, j)|i, j \in V, j > i + 1\}$

(b) The position of a vertex i in the path is denoted by $pos(i) = i$ and the vertex at position i in the path is denoted by $v(i) = i$.

(c) For all $(i, j) \in \bar{E}$, set

$$\lambda^*_{ij} = \frac{\sum_{k=pos(i)+1}^{pos(j)-1} p_{v(k)}}{\sum_{k=pos(i)}^{pos(j)-1}\big(c_{v(k),v(k+1)} + p_{v(k)}\big) - c_{ij} - p_{pos(i)}}.$$

(ii) (Breakpoint computation)

(a) Let $\lambda_{min} = \min_{(i,j)\in\bar{E}}\{\lambda^*_{ij}\}$, and

(b) $R_{min} = \{(i,j) \in \bar{E} : \lambda^*_{ij} = \lambda_{min}\}$.

(c) Append λ_{min} with associated profit and cost to $\mathcal{SE}$.

(iii) (Updating) While $R_{min} \neq \emptyset$, do

(a) delete all vertices at positions $(pos(i)+1, \ldots, pos(j)-1)$ from S, and all related arcs from $\bar{E}$ and from R_{min};

(b) for all $k \geq j$, $k \in S$: set $pos(k) := pos(k) - (pos(j) - pos(i)) + 1$ and $v(pos(k)) := k$

(c) recompute all λ^*_{ij} for arcs $(l,m) \in \bar{E}$ for which $l < i$ and $m > j$.

(iv) (Output) If $S \neq \{0, n+1\}$ go to (ii); else return the set of supported efficient solutions $\mathcal{SE}$.

Theorem 3.12. *Algorithm Parametric Shortest Path solves Problem 2 on Kalmanson matrices in $O(n^3)$ time.*

Proof. Correctness follows from the previous discussion. Indeed, let λ'_{min} and λ''_{min} be computed in two successive iterations of the algorithm. The vertices visited for λ'_{min} are optimal for any $\lambda \in [\lambda'_{min}, \lambda''_{min}]$.

Initialization takes $O(n^2)$; at every iteration at least 1 vertex is removed, hence there are $O(n)$ iterations. Every iteration requires $O(n^2)$ due to (iii)(a) and (iii)(c). Total complexity is then $O(n^3)$. □

3.6 Complexity classes for multi-objective optimization problems

In this chapter, we use basic complexity classes from single-objective optimization theory to describe complexity of a bi-objective problem. However, specific complexity classes for multi-objective optimization problems exist and we apply these classes to our problem in this section. In multi-criteria optimization the goal is to find a set of (Pareto) optimal solutions. In this case, not only complexity in terms of the input, but also complexity in terms of the output is relevant. Indeed, when an exponential number

of optimal solutions exists, detecting a single solution in polynomial time does not solve the whole problem in polynomial time. Specific complexity classes for multi-criteria scheduling exist to capture these situations; for a survey we refer to T'kindt et al. (2005). A distinction is made between the optimization problem, counting problem and enumeration problem associated to a multi-criteria optimization problem. The optimization problem refers to the underlying single-criteria optimization problem, the basic complexity classes apply here, i.e. P, NP, NP-complete (decision problems), and NP-hard (or NPO-complete)(optimization problems). The counting problem is the problem of determining the number of optimal solutions regarding the objectives of the multi-objective problem. Note that the solution to this problem is not a solution for the original problem but a number. A counting problem belongs to complexity class $\#P$ if the corresponding decision or optimization problem belongs to NP or NPO respectively. The class $\mathcal{F}P \subset \#P$ denotes the class of polynomially solvable counting problems. In the enumeration problem the goal is to find all optimal solutions. It is clear that enumeration is at least as hard as counting. Several complexity classes have been dedicated to enumeration problems, depending on the size of the input and the size of the output, see T'kindt et al. (2005) for an overview. The class $\mathcal{E}NP$ is a direct generalization of class NPO. Let us now apply these classes to Problem 1 and Problem 2.

Theorem 3.13. *The counting problem associated with Problem 1 is $\#P$-complete.*

Proof. The underlying optimization problem of Problem 1, the OP on a tree, is equivalent to the knapsack problem (section 3.3) and thus NP-hard, which is equivalent to being NPO-complete. Ehrgott (2000) shows that the counting problem associated with the knapsack problem is $\#P$-complete; the same result thus holds for the counting problem associated with Problem 1. □

Theorem 3.14. *The enumeration problem associated to Problem 1 is $\mathcal{E}NP$-complete.*

Proof. For the proof of Theorem 3.14 we will use the following property.

Property 3.15. ***(T'kindt et al. (2005))*** *If an optimization problem P is NPO-complete and its counting problem is $\#P$-complete, then the associated enumeration problem is $\mathcal{E}NP$-complete.*

The correctness of Theorem 3.14 follows immediately from Property 3.15, from the proof of Theorem 3.3 and from Theorem 3.13. It thus holds that the enumeration problem, i.e. Problem 1 is (weakly) $\mathcal{E}NP$-complete. □

Theorem 3.16. *The enumeration problem associated to Problem 2 is in P.*

Proof. We developed an algorithm solving Problem 2 on the tree in polynomial time, the associated enumeration problem is thus in P. □

Given that the enumeration problem is easy it is clear that also the counting problem is easy, so we state the following.

Corollary 3.17. *The counting problem associated to Problem 2 belongs to $\mathcal{F}P$.*

3.7 Conclusions

In this chapter we have studied the traveling salesman problem with profits from a bi-objective point of view, on graphs with a tree metric. We have considered three problems: finding all efficient points (Problem 1); finding all extreme supported efficient points (Problem 2); finding one efficient point, corresponding to a given combination of the two objectives (Problem 3). For every problem, we have developed efficient algorithms. Moreover, we have analyzed some special cases, including problems on a path and

	Tree		Line		Kalmanson	
	LB	UB	LB	UB	LB	UB
Problem 1	$> n^k$(unless P=NP)	FPTAS	$O(n^2)$	$O(n^2 \log n)$	$> n^k$(unless P=NP)	FPTAS
Problem 2	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^3)$
Problem 3	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Table 3.1: *A summary of our complexity results: LB = lower bound on the complexity of a problem; UB = upper bound on the complexity corresponding to a proposed algorithm*

we extended our results to more general graphs satisfying the Kalmanson conditions. Table 3.1 summarizes our results.

There are some interesting extensions that could be studied in future research. When service times are added to the customers, it no longer holds that all predecessors of a customer i are visited when i is visited. It follows that Algorithm LR-DP can not be applied directly. A similar argument holds when a visit consists in a trip for the customer visited (pickup and delivery), or when time windows are added. In these cases it can happen that a customer is not visited the first time the server passes by, but is visited later on in the tour.

Chapter 4

Charlemagne's challenge: the periodic latency problem

Latency problems are characterized by their focus on minimizing total waiting time for all customers. We consider periodic latency problems; an extension of standard latency problems. In a periodic latency problem each customer has to be visited regularly. More precisely, given is a server traveling at unit speed, and a set of n customers with their positions. With each customer i a periodicity q_i is associated that is the maximal amount of time that is allowed to pass between consecutive visits of the server to customer i, $1 \leq i \leq n$. In a problem we denote as PLPP, the goal is then to find a repeatable route for the server visiting as many customers as possible without violating the q_i's of the customers visited. Further, we consider the PLP in which the number of servers needed to serve all customers is minimized. We give polynomial-time algorithms and NP-hardness results for these problems depending upon the topology of the underlying network.

Prologue

During his reign in the years 768-814, Charlemagne traveled constantly through his empire in Western Europe. Counts had been appointed to govern different pieces of Charlemagne's empire (called counties). On his travels, Charlemagne visited his counts regularly. One reason for these visits was to ensure loyalty of his counts. Indeed, when a count was not visited for a certain period, the count would no longer obey Charlemagne, and declare independence, thereby rising against the emperor. Clearly, this would force Charlemagne to act, and start an expensive war against the rebeling count. Charlemagne's challenge was to find a visiting sequence of his counts so that the time elapsed between two consecutive visits to a count would not exceed the "loyalty period" of that count.

4.1 Introduction

Consider the following problem. We are given a set of customers $N = \{1, 2, \ldots, n\}$ with their positions $x_1, x_2, \ldots, x_n$ in some metric space; for each pair of customers $i, j \in N$, there is a distance d_{ij}. We are also given a server that travels at unit speed (and always at full speed). There is a number q_i associated with every customer i which indicates the *periodicity* of customer i, $i \in N$. More precisely, q_i is the maximal amount of time that is allowed to pass between two consecutive visits to customer i, $i \in N$. Each customer $i \in N$ also has an associated profit p_i. A customer i is called *served* when the time elapsed between each two consecutive visits does not exceed q_i, $i \in N$. The goal is to find a travel-plan for the server which maximizes the total profit of the customers served. This travel plan can be represented by a list of customers (of infinite length) that prescribes the sequence in which the served customers are visited. Thus, in a feasible solution, (i) each served customer is visited an infinite number of times, and (ii) the time elapsed between two consecutive visits to customer i does not exceed q_i, $i \in N$. We assume that all data are integral. We call this problem the Periodic Latency Problem with Profits (PLPP).

Clearly, referring back to Charlemagne's challenge, a count is a customer, Charlemagne is the server, and the loyalty periods are represented by the q_i's. If the profit of a customer represents the area of the county, Charlemagne's challenge is to maximize the size of his empire without having to fight internal wars.

In this chapter, we also consider the problem where multiple servers are available and all customers need to be served. We assume here that a customer must be served by a single server; see Section 4.5 for the relevance of this assumption. The goal is then to minimize the number of servers required to serve all customers periodically. We call this problem the Periodic Latency Problem (PLP). Further, we dedicate a short section to the periodic latency problem with profits and multiple servers (MPLPP). Here again the goal is to find routes for the servers such that total collected profit is maximized.

Why 'latency'?

Latency problems are characterized by their focus on total waiting time as an objective function, see e.g. de Paepe et al. (2004) and the references contained therein. Latency problems differ from problems where travel time of the server is the objective. Notice that the problems that we study here, share the same fundamental property with latency problems: any period in time matters to all customers. That is why we refer to the problems described here as periodic latency problems; indeed, time matters for the customers, whereas the distance traveled by the server is of no interest.

Related Problems

Many routing and scheduling problems require a periodic solution. In the periodic TSP (Paletta (2002)), a set of customers each with a certain frequency is given. Each customer needs to be visited according to its frequency within a given planning period T. A solution then consists in a set of routes, one route for every day in the planning period T. This setting can be generalized to the periodic VRP, where more than one vehicle is available and several routes can be performed each day of the planning

horizon T (Mourgaya and Vanderbeck (2006)). These problems, however, do not belong to the class of latency problems.

Other applications of periodic scheduling problems can be found in scheduling of robotic cells, see e.g. Crama et al. (2000); or in digital-signal-processing, see Verhaegh et al. (2001). A signal processing algorithm consists in elementary operations that need to be carried out repeatedly on successive samples of a given digital signal. Korst et al. (1997) study the problem of nonpreemptively scheduling periodic tasks on a minimum number of identical processors. They mention an application where a number of continuous data streams must be read from a minimal number of identical disk units. Any situation where there is a repetitive execution of operations with strict timing requirements is relevant (Verhaegh et al. (2001)). In these settings, precedence constraints might also be present.

A non-periodic latency problem with profits is discussed in Chapter 2. There, a profit p_i is associated with every customer i and these profits go down linearly with time, while the customer is waiting to be served. The goal is to select customers and to find a route for the server visiting these customers such that the collected profit is maximal. A similar problem with time windows is described by Frederickson and Wittman (2007). Each service request is assigned to a time window and the goal is to find a tour that visits the maximum number of locations during their time windows. Each request that the repairman completes yields a given profit. These problems, however, are not periodic.

A problem that is probably closest to our setting is the problem described in Campbell and Hardin (2005). In their problem, the number of servers needed to serve all customers is minimized, under the assumption that each customer $i \in N$ needs to be visited *precisely* every q_i time-units. They show that a solution exists which is periodic (see Section 4.3) with length $T = lcm^1(q_1, q_2, \ldots, q_n)$.

Motivation

We see the PLPP and the PLP as basic problems with applications in

[1] least common multiplier

diverse areas. We describe here two different fields where these periodic latency problems occur. Several applications of PLPP and PLP can be distinguished. Recently, Studer (2008) described "rounding", a management process that can help to improve management and leadership skills. Studer believes that managers should make regular rounds to check on their employees. In that way, managers find out what matters to employees, and potential problems can be dealt with before they occur. This "rounding" model is based on the rounds doctors and nurses make to check on their patients in a hospital. Dimov et al. (2007) explore a method called "minirounds" that appears to improve physician-patient communication and satisfaction at a hospital. Minirounds are defined as follows: "A series of short patient encounters [each lasting about a minute] during which the physician asks patients about any changes in their condition and provide a concise daily update" (Dimov et al. (2007)). Efficiently organizing these mini-rounds is an instance of PLP. Notice that the latter application suggests a specific topology of the customers: in the introductive chapter we mention the tree network as being relevant for representing the corridor structure in hospitals and offices. We extensively study this topology in this chapter.

Another field where periodic latency problems occur is maintenance, more precisely preventive periodic maintenance. Machines, located at given positions (say different plants) need to be inspected regularly. Obviously, when a machine is viewed as a client, and when the periodicity of each machine is given, the PLP arises. Although there is quite some literature on preventive periodic maintenance, (see e.g. Dekker et al. (1997) for an overview), many contributions are stochastic, and we are not aware of deterministic situations where distances between machines are taken into account (see Anily et al. (1998) for a related problem).

We give an overview of our results in Section 4.2. We elaborate on the issue of periodicity in Section 4.3. In Section 4.4 we deal with the complexity of the single server problem, i.e., PLPP. In Section 4.5 we deal with PLP. The results for the PLP can easily be extended to the MPLPP, this is shown in Section 4.6. In these three sections we completely

classify the complexity for the following topologies: line, circle, star, tree, general; and we consider arbitrary profits p_i, versus $p_i = 1$, and arbitrary periodicities q_i, versus $q_i = Q$, for all $i \in N$. Finally, Section 4.7 discusses possible extensions.

4.2 The results

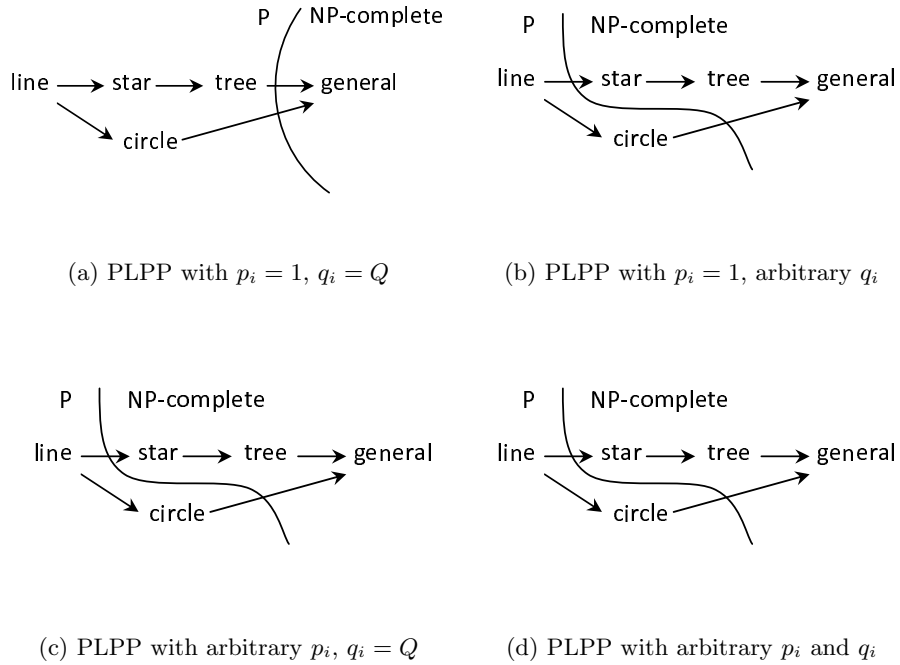
Figure 4.1 summarizes our results concerning the complexity of the PLPP for different settings of p_i and q_i, in different metric spaces. From Theorem 4.7 and Corollary 4.9 it follows that in a general case with arbitrary profits and frequencies (d), the PLPP on the line and the PLPP on the circle are solvable in polynomial time. Hence, the same results hold for the more restricted topologies in (a), (b), and (c). Figure 4.1(a) shows results for the PLPP with unit profits and a common frequency Q; this result is due to Theorem 4.14 and Corollary 4.13. Figure 4.1(b) represents the results for the PLPP with unit profits and arbitrary frequencies; this follows from Theorem 4.11. Similarly, Figure 4.1(c) holds for PLPP with arbitrary profits and a common frequency Q, see Theorem 4.10.

In the PLP, the goal is to minimize the number of servers needed to visit all customers. Profits are not applicable in this case. Results are represented in Figure 4.2 and follow directly from Theorem 4.17, Corollary 4.18, and Theorem 4.19.

4.3 Periodicity

In this section periodicity of solutions is being analyzed. The analysis in this section is restricted to the PLPP. However, everything also holds for the PLP as a solution to the PLP consists in s single server solutions, where s denotes the number of servers used.

A solution to the PLPP can be represented by an infinite sequence of customers: customer i appearing on the p-th position of the sequence indicates that the p-th customer visited is customer i. Let L be a sequence



(a) PLPP with $p_i = 1$, $q_i = Q$

(b) PLPP with $p_i = 1$, arbitrary q_i

(c) PLPP with arbitrary p_i, $q_i = Q$

(d) PLPP with arbitrary p_i and q_i

Figure 4.1: *Complexity results PLPP*

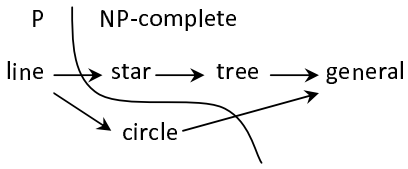


Figure 4.2: *Complexity results PLP*

representing a solution to the PLPP and S the subset of customers visited. Notice that for $|S| = 1$, periodicity is trivial.

We call any sequence of customers of finite length a *subsequence*.

Definition 4.1. A subsequence is called a *k-cycle* if

- the subsequence starts and ends with some customer $i \in N$ not appearing elsewhere in the subsequence, and
- each served customer $j \neq i$ appears with a frequency of at least 1, and at most k, in the subsequence.

Definition 4.2. We say that a solution to PLPP is *periodic* if, when viewed after some position p, the sequence representing this solution is a concatenation of identical subsequences.

Definition 4.3. We say that a solution to PLPP is a *k-cycle* if, when viewed after some position p, the sequence representing this solution is a concatenation of identical k-cycles (where it holds that when two consecutive visits are performed to a single customer this can be considered a single visit to that customer).

Clearly, from the viewpoint of implementation and optimization, periodic solutions are preferable over non-periodic ones. Fortunately, we lose nothing by restricting the search for a solution to the class of periodic solutions:

Theorem 4.4. *If there exists a feasible solution for an instance of PLPP that visits customer-set $S \subseteq N$, then there exists a periodic feasible solution visiting S.*

Proof. Let $Q = \max_{i \in S} q_i$. All customers are positioned at distinct locations and $d_{ij} \geq 1$. Consider any feasible solution to PLPP, represented by a sequence L. Suppose that the size of the set S of served customers is equal to $2 \leq |S| \leq n$. The proof relies on the following observation.

Observation 1. *L consists of infinitely many k-cycles, for some k.*

To argue that this observation holds, consider any position p in the sequence L and the customer i visited at p. Then, at position $p+1$, a customer $j \neq i$ is visited and, at position $p+2$, either again customer i is served or a "new" customer l. At a certain position $q > p$, $|S|-1$ different customers will have been served since position p. Thus, one customer, say customer l, will then necessarily be visited at a position $q' > q$ and also at a position $p' < p$; the total number of time units elapsed between these two visits can be no more than Q. It follows that between two consecutive visits to this customer l, all other customers are visited at least once. Since this holds for any position p, the observation is valid.

Now, as for a specific customer at most n^Q different k-cycles exist, the total number of different k-cycles that can appear in L cannot exceed n^{Q+1}. □

Notice that we show here that the number of k-cycles is finite given that $d_{ij} \geq 1$; however, when $d_{ij} < 1$ the number of k-cycles will be much larger but still finite. The observation above then implies that a same k-cycle will appear more than once. By repeating a subsequence that starts with this k-cycle, plus everything that followed this k-cycle up to its next appearance, we modify L into a feasible solution that is periodic.

4.4 The complexity of PLPP

In this section, we prove the results summarized in Figure 4.1. Section 4.4.1 deals with the PLPP on the line or on a circle, Section 4.4.2 deals with the PLPP on a star, Section 4.4.3 deals with the PLPP on a tree, and finally, Section 4.4.4 deals with the PLPP on a arbitrary topology.

4.4.1 PLPP on the line and the circle

Consider first an instance of the PLPP with arbitrary profits p_i, and arbitrary periodicities q_i where all customers are positioned on the line. We assume that $x_1 < x_2 < \ldots < x_n$, and we denote the distance between x_i and x_j as d_{ij} for $i, j \in N$. We first show that for this special topology,

we can restrict ourselves to solutions where the server simply oscillates between two customers. In other words, there is a solution that is a 2-cycle (see Definition 4.1).

Theorem 4.5. *If there exists a feasible solution for a set of customers $S \subseteq N$, then there is a 2-cycle serving customer-set S, with $|S| \geq 2$.*

Proof. Consider any pair of locations $x_i < x_j$, $i, j \in S$. Since $i, j \in S$, there exist two moments in time $t_i < t_j$ such that (i) the server is at x_i at time t_i, (ii) the server is at x_j at time t_j, and (iii) neither x_i nor x_j is visited at any time t with $t_i < t < t_j$. Since $i \in S$ it must be true that $q_i \geq t_j - t_i + d_{ji} \geq d_{ij} + d_{ji}$. The first inequality is true because being able to serve both customers i and j implies that the periodicity of customer i cannot be smaller than the time the server needs to travel from x_i to x_j and back (recall that we assume that the server travels at unit speed). Thus, the time needed to travel from x_i to x_j is at least equal to the distance d_{ij}, which gives us the second inequality. Similarly, we can argue that $q_j \geq d_{ji} + d_{ij}$. Thus, for all customers i and j in S it holds that

$$q_i \geq d_{ij} + d_{ji} \tag{4.1}$$

and

$$q_j \geq d_{ji} + d_{ij}. \tag{4.2}$$

We now exhibit a 2-cycle that is able to serve the customers in S. Indeed, consider a server that travels from the left-most customer in S to the right-most customer in S, and back, and repeating this pattern. Each customer $i \in S$ is served as long as

$$q_i \geq \max_{j \in S}(d_{ij} + d_{ji}) \tag{4.3}$$

This, however, is implied by (4.1) and (4.2), and hence a 2-cycle serves the customers in S. □

Using this claim it is not hard to argue that we can answer the question: can we visit all given customers? in $O(n)$ time. We refer to this problem as decision-PLPP.

Corollary 4.6. *Decision-PLPP on the line can be answered in $O(n)$.*

Proof. Given positions $x_1, \ldots, x_n$ we verify whether (4.3) holds. Given the line topology, the maximum in (4.3) can only be attained for $j = 1$ or $j = n$. It follows that we need to verify $O(n)$ inequalities. □

Theorem 4.7. *PLPP on the line with arbitrary p_i and arbitrary q_i can be solved in $O(n^2)$.*

Proof. We only need to search for a best 2-cycle (Theorem 4.5). Clearly, considering each possible combination of leftmost and rightmost customer, and then checking for all intermediate customers whether they can be served, yields an immediate $O(n^3)$ algorithm. We now proceed to describe an $O(n^2)$ algorithm.

Lemma 4.8. *Consider a server traveling on the interval $[x_i, x_i + L]$; this server serves a customer j if and only if*

(i) $x_i \leq x_j$

(ii) $x_j \leq x_i + L$

(iii) $2L + 2x_i - 2x_j \leq q_j$

(iv) $2x_j - 2x_i \leq q_j$.

Proof. Conditions (i) and (ii) state that point x_j must be contained in the interval $[x_i, x_i + L]$. Conditions (iii) and (iv) follow from the following observation. The time needed for the server to travel from x_j to the most right point of the interval and back to x_j (i.e. twice the distance between x_j and $x_j + L$), and the time needed to travel to the most left point of the interval and back (i.e. twice the distance between x_i and x_j), respectively, may not be larger than the periodicity q_j. □

According to Lemma 4.8, a customer j is thus served by a server traveling in $[x_i, x_i + L]$ if and only if $x_i \leq x_j$ and $x_i \geq x_j - \frac{1}{2}q_i$ and if L lies in the "*activity interval*"

$$A_j := [x_j - x_i; x_j + \frac{1}{2}q_j - x_i].$$

The set of served customers $S(L)$ is then the following:

$$S(L) := \{j : x_i \leq x_j, x_i \geq x_j - \frac{1}{2}q_j, \text{ and } L \in A_j\}.$$

Our algorithm consists of a preprocessing step and a main algorithm computing the best value for L, i.e. L for which $\sum_{j \in S(L)} p_j$ is maximized, and this is done for every i.

(*Preprocessing*) For every customer i, let $l(i) = x_i$ and $r(i) = x_i + \frac{1}{2}q_i$. Sort all values of $l(i)$ and $r(i)$ in a global non-decreasing list T.

(*Main algorithm*) For each customer i the interval $[x_i, x_i + L]$ maximizing the resulting profit of the served customers can be computed as follows.

(i) For all customers in T, select the customers j not violating conditions $x_i \leq x_j$ and $x_i \geq x_j - \frac{1}{2}q_j$ and define $l'(j) = l(j) - x_i$ and $r'(j) = r(j) - x_i$; $l'(j)$ and $r'(j)$ are then the leftmost and rightmost point, respectively, of the activity interval A_j of customer j. Remark that the list containing all values $l'(j)$ and $r'(j)$ is sorted, we call this set T'.

(ii) The first entry in T' will be $l'(i) = 0$, for the choice of $L = 0$ the corresponding profit P equals p_i. Set $P_{max} = p_i$.

(iii) Work through T' and determine the profits for the corresponding values of L.

  (a) If the next element in T' is a lower bound $l'(j)$ of an activity interval A_j, set $P := P + p_j$.

  (b) If the next element in T' is an upper bound $r'(j)$ of an activity interval A_j, set $P := P - p_j$

  (c) If $P > P_{max}$, set $P_{max} := P$.

Select the highest P_{max} over all customers i.

The preprocessing step takes $O(nlogn)$, the main algorithm has $O(n)$ iterations and each iteration takes $O(n)$, yielding a total complexity of $O(n^2)$.

□

Notice that Theorem 4.7 is also valid for a server whose speed differs between traveling to the right and traveling to the left.

Consider now an instance of PLP where all customers are positioned on a circle: we can simply extend Theorem 4.7 to this case:

Corollary 4.9. *PLPP on the circle with arbitrary p_i, and arbitrary q_i can be solved in $O(n^2)$.*

Proof. A solution to PLPP on the circle is either a 2-cycle or the full circle (which is actually a 1-cycle). The best 2-cycle can be found using the same algorithm as described for the line. When fixing a leftmost customer, the rightmost customer is found going clockwise through the circle. Thus, the optimal solution to an instance of PLPP on the circle can be found in $O(n^2)$. □

4.4.2 PLPP on a star

Let us consider a star graph. The customers are positioned in the end nodes and, in addition to a profit p_i and a periodicity q_i, there is a distance d_i given which denotes the distance between the position of customer i and the center of the star, $i \in N$.

It is easy to see that the PLPP where all $p_i = 1$ and $q_i = Q$ for all $i \in N$ is solvable in polynomial time. Indeed, by selecting customers with the smallest d_i until the tour length exceeds Q, an optimal solution is found. In fact, a more general result is shown in Section 4.4.3. In Section 4.5 we show that adding a second server already makes the problem NP-hard.

When profits p_i are arbitrary, and $q_i = Q$, PLPP on a star can be shown to be equivalent to the knapsack problem.

Theorem 4.10. *PLPP on a star with arbitrary profits p_i, and with all $q_i = Q$, is NP-hard.*

Proof. We reduce from knapsack, which is a weakly NP-complete problem Garey and Johnson (1979). Consider an instance of the knapsack problem

with n items where each item i has a certain value w_i and a requirement a_i. The knapsack has size B and the question is whether we can fit a subset of the items with a total value of W in this knapsack.

Now construct an instance of PLPP on a star as follows. There are n customers. Each customer $i \in N$ has an associated weight $p_i := w_i$, a periodicity $Q := B$, and a distance to the center of the star $d_i := \frac{1}{2}a_i$. Does there exist a subset of the customers with total profit equal to W such that each customer is visited at least once within each time period Q?

In case the instance of knapsack is a yes-instance, we can copy that solution to the instance of PLPP: selected items correspond to selected customers. Starting in the center, it is clear that by visiting the selected customers in any order, and repeating that pattern gives a feasible solution. When the instance of PLPP admits a yes, there is a set of customers that we can apparently serve. A served customer i implies a travel time of $2d_i = a_i$. Feasibility of the PLPP-instance ensures that the corresponding set of items fits in the knapsack. □

When we consider PLPP on a star with $p_i = 1$, and arbitrary periodicities q_i, the problem is also NP-hard.

Theorem 4.11. *PLPP on a star with all $p_i = 1$, and arbitrary periodicities q_i, is NP-hard.*

Proof. We show that 3-Partition, which is a NP-complete problem (Garey and Johnson (1979)), can be reduced to PLPP on a star with $p_i = 1$, and arbitrary periodicities q_i. This proof is based on the proof of Korst et al. (1997) for NP-hardness of Scheduling Periodic Tasks with Slack (PSSP).

An instance of 3-Partition consists of a set A with $3m$ items and a positive integer B representing the size of the m bins. Each item $a_i \in A$ has an associated size z_i for which it holds that $\frac{B}{4} < z_i < \frac{B}{2}$ and $\sum_{a_i \in A} z_i = mB$. Can A be packed into m bins, each containing three items?

We construct an instance of our special case of PLPP on a star such that the items can be packed in m bins if the customers in the corresponding PLPP, each with their respective periodicity, can be served by a single

server. We are given a set of $n := 3m + 1$ customers, each with periodicity $q_i := m(B + 2)$ and distance $d_i := \frac{1}{2}z_i$ to the center of the star, for $i = 1, \ldots, 3m$. Further, customer $3m + 1$ has $q_{3m+1} := B + 2$ and $d_{3m+1} := 1$. The question is whether there is a solution serving all $3m + 1$ customers.

If 3-Partition has a solution, it is clear how to copy that solution to the PLPP instance, and get a solution serving all $3m+1$ customers. If the PLPP instance has a solution in which all $3m + 1$ customers are served, then, between two consecutive visits to customer $3m+1$, there are exactly B time units left that can be used to visit other customers. Each of the $3m$ customers left must be visited at least once in time period $m(B + 2)$. In that time period there are m available time slots of B time units. Thus, the $3m$ customers can be assigned to the m different time slots if and only if the corresponding items can be packed in m bins. □

The following holds when periods are arbitrary and distances unitary:

Theorem 4.12. *PLPP on a star with unitary distances, unitary profits, and arbitrary periodicities is solvable in polynomial time.*

Proof. Consider an optimal solution visiting a maximal subset S of customers. Set T is the set of unvisited customers, $N = S + T$. It holds that for any two customers $i \in T$ and $j \in S$ with $q_i > q_j$, i can be moved to subset T and j to subset S. A new optimal solution is obtained. Thus, an optimal solution can easily be found by selecting customers in decreasing order of q until a feasible solution is no longer obtained. □

4.4.3 PLPP on a tree

We argue here that PLPP on a tree with $p_i = 1$ and $q_i = Q$ is nothing but an orienteering problem (OP). In Chapter 3 we describe how to modify an algorithm from Johnson and Niemi (1983) in order to solve the orienteering problem restricted to a tree. An instance of OP on a tree consists of a set of vertices N where each vertex $i \in N$ has an associated profit p_i and each edge between two vertices i and j in the tree has a cost c_{ij}. A maximum on the cost C is given. The goal is to find a route visiting a subset of

the vertices with cost no more than C and collecting a maximal amount of profit. An instance of PLPP with $p_i = 1$ and $q_i = Q$ is then an orienteering problem with $p_i = 1$ ($\forall i \in N$), $c_{ij} := d_{ij}$, and with $C := Q$. The algorithm in Chapter 3 solves OP on a tree in $O(nP_{tot})$, with P_{tot} the sum of all given profits; since $p_i = 1$, $\forall i \in N$, total running time for PLPP is only $O(n^2)$.

Corollary 4.13. *PLPP on a tree with $p_i = 1$ and $q_i = Q$ is solvable in $O(n^2)$.*

4.4.4 PLPP on an arbitrary topology

Theorem 4.14. *PLPP with $p_i = 1$, and $q_i = Q$, is NP-hard.*

Proof. We prove NP-completeness of this variant of PLPP by a reduction from Hamiltonian cycle. An instance of the Hamiltonian cycle problem is specified as follows: given a graph $G = (V, E)$, does there exist a Hamiltonian cycle in G?

Now consider the following instance of PLPP. A node in V corresponds to a customer, and we set $n := |V|$. For each pair of customers i, j that is connected via an edge in E we set $d_{ij} := 1$, else we set $d_{ij} := 2$. The periodicity is $q_i := |V|$ for each $i \in N$. Now, does there exist a solution to PLPP with value n?

If a Hamiltonian cycle exists in G, there exists a tour in the PLPP instance with length n visiting all the locations. This solution is periodic. Vice versa, if a solution serving all customers in the PLPP instance exists, G must contain a Hamiltonian cycle. □

4.5 The complexity of PLP

In this section we study the PLP and we prove the results summarized by Figure 4.2. In the PLP our goal is to minimize the number of servers needed to visit all customers. We assume that every customer must be assigned to and served by a single server; this is a crucial assumption as can be seen from the example in Figure 4.3. Observe that when each

customer must be served by one server, three servers are needed to serve the customers in the example. If, however, that assumption is dropped, the three customers from the example can be served using only 2 servers, each server alternatingly serving the middle customer.

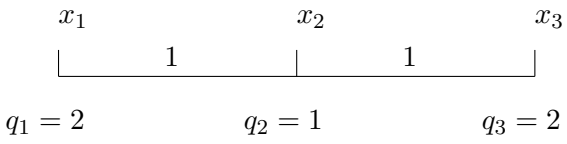


Figure 4.3: *PLP*

Determining for an arbitrary topology, whether a single server suffices to serve all customers, is NP-hard; this follows directly from Theorem 4.14. In this section we show that a dynamic programming approach solves the case of the line in polynomial time, whereas PLP on a star remains NP-hard.

4.5.1 PLP on the line

We develop a dynamic programming algorithm for the PLP on the line. Given are n customers with their positions $x_1 < x_2 < \ldots < x_n$ on the line; and a set of n identical servers. The goal is to minimize the number of servers necessary to visit all customers periodically, i.e. without violating any q_i.

There exists an optimal schedule to an instance of PLP on the line of the following shape:

- Every server s commutes between the left and the right endpoint of its interval I_s.
- The intervals I_s and I_t of any two different servers s and t are either disjoint, or one of them does contain the other one.
- The left endpoint of interval I_s is only assigned to server s.

We say that in some schedule a server *covers* customer i if the time elapsed between two consecutive visits of this server to the customers is at most q_i. The following lemma clearly holds:

Lemma 4.15. *Let $a \leq b \leq c \leq d$ be four request points. Assume that server s commutes on the outer interval $[a, d]$, and that server t commutes on the inner interval $[b, c]$. Then every point in interval $[b, c]$ that is covered by the outer server s will also be covered by the inner server t.*

Indeed, the frequency with which any point x_i within $[b, c]$ is visited by the server commuting in $[b, c]$ is higher than for the server commuting in $[a, d]$.

A dynamic programming algorithm

The dynamic program is built around the following definition.

Definition 4.16. Let i and j be integers with $1 \leq i, j \leq n$. Let i' and j' be integers such that either (i) $1 \leq i' \leq j' \leq n$ and $x_1 \leq x_{i'} \leq x_{j'} \leq x_n$ where x_i and x_j are contained in the interval $[x_{i'}, x_{j'}]$, or such that (ii) $i' = j' = 0$.

Then $F[i; j; i'; j']$ denotes the smallest number of servers that can serve all customers $k = i, i + 1, \ldots, j$ with the exception of the customers that are already covered by an external server that commutes on the interval $[x_{i'}, x_{j'}]$. (In the case $i' = j' = 0$, there is no such external server.)

Notice that in this definition we explicitly allow the situation $j < i$, which yields an empty interval and an empty set of requests. We will now compute all the values $F[i; j; i'; j']$ step by step and in increasing order of $j - i$ (the number of customers in the underlying interval).

All cases with $j < i$ have empty intervals, and hence need $F[i; j; i'; j'] = 0$ servers. In the remaining cases we have $j \geq i$. If the leftmost point is already served by the external server, we may simply set $F[i; j; i'; j'] := F[i + 1; j; i'; j']$. Otherwise the leftmost point i must be served by some new server s. We branch into several cases that depend on the behavior of s.

Assume that the server s commutes on the interval $I_s = [x_i, x_l]$ with $i \leq l \leq j$. Then we need at least

$$\alpha(l) \;:=\; F[i+1; l; i; l]$$

additional servers working and serving in interval I_s. Because of Lemma 4.15, the influence of the external server commuting on the outer interval $[x_{i'}, x_{j'}]$ has now become irrelevant; everything this server has covered will also be covered by the new server s. Furthermore, we need at least

$$\beta(l) \;:=\; F[l+1; j; i'; j']$$

servers working in the remaining uncovered interval $[x_{l+1}, x_j]$. Therefore the smallest possible number of servers in this situation is

$$F[i; j; i'; j'] = \min\left\{1 + \alpha(l) + \beta(l) \mid i \leq l \leq j\right\}.$$

In the end, the answer to the global problem can be found in $F[1; n; 0; 0]$. We have to compute $O(n^4)$ values $F[i; j; i'; j']$, and the computation of each value takes linear time. Hence the overall running time is $O(n^5)$.

Theorem 4.17. *PLP on the line can be solved in $O(n^5)$.*

4.5.2 PLP on a circle

The algorithm for the problem on the line can be extended to the circle, adding a factor n to the running time of the line algorithm.

Corollary 4.18. *PLP on the circle can be solved in $O(n^6)$.*

Proof. PLP on a circle can be broken down to n instances of the problem on the line. Solving all these line instances and selecting the best from these n solutions, yields an optimal solution to the PLP on the circle. Thus, the optimal solution to an instance of PLP on the circle can be found in $O(n^6)$. □

4.5.3 PLP on a tree

We prove that the PLP on a star with $q_i = Q$ is NP-hard; NP-hardness of the PLP on a tree with arbitrary q_i follows immediately from this result. In Section 4.4.2 it was shown that the periodic latency problem on a star graph with all customers having the same profit and requiring equal frequency is easy to solve; adding a second server though makes the problem much harder to solve.

Theorem 4.19. *PLP on star is NP-hard, even if for all $i \in N$: $q_i = Q$.*

Proof. We reduce from partition. An instance of the partition problem has a set $X = \{x_1, \ldots, x_n\}$ with $\sum_{i=1}^{n} s(x_i) = 2k$, where $s(x_i)$ denotes the size of x_i. Can the set X be partitioned into two sets X_1 and X_2 such that $\sum_{x \in X_1} s(x) = \sum_{x \in X_2} s(x) = k$ and each element occurs exactly once?
An instance of PLP on a star is constructed as follows. There is a star with n spokes, and a customer located at each spoke with distance $\frac{1}{2}x_i$ from the center of the spoke, for $i = 1, \ldots, n$. Each customer (spoke) has a periodicity $Q := k$, meaning that the maximal time that can pass between two consecutive visits to a customer equals k. There are two servers, traveling at unit speed, positioned in the center of the graph. Now, does there exist a route for each of the servers such that all customers can be served periodically?

If a solution to partition exists, the set of customers in X_1 can be assigned to one server and the customers in X_2 to the other server and each server can visit these customers in Q time units. If a solution to PLP on the star exists, each customer is visited once within every Q time units. Since total travel time to visit all customers equals $2Q$, it must be the case that every server travels exactly Q time units. Customers visited by server 1 can be assigned to one set and customers visited by server 2 to the second set, and a solution for partition is obtained. □

As a result, PLP on a tree is NP-hard.

4.6 Extension: the complexity of MPLPP

The complexity results from the previous section can be extended to a periodic latency problem with profits and multiple servers, denoted by MPLPP. As before, we are given a set of n customers with their positions $x_1, x_2, \ldots, x_n$; and each customer i has an associated profit p_i and periodicity q_i. Further, S identical servers are given, with $S < n$. The goal is then to find a repeatable route for each server collecting a maximal amount of profit without violating the q_i's. We show that, when all costumers are positioned on the line/circle the dynamic programming algorithm from the previous section can be transformed to solve the MPLPP in polynomial time. On a tree or a arbitrary graph the problem is NP-hard. One can easily check that Lemma 4.15 still holds for MPLPP on the line. Then a state in the dynamic programming algorithm is defined as follows.

Definition 4.20. Let i and j be integers with $1 \leq i, j \leq n$. Let i' and j' be integers such that either (i) $1 \leq i' \leq j' \leq n$ and $x_1 \leq x_{i'} \leq x_{j'} \leq x_n$ where x_i and x_j are contained in the interval $[x_{i'}, x_{j'}]$, or such that (ii) $i' = j' = 0$.

Then $P[i; j; i'; j'; s]$ denotes the maximum profit that can be collected serving customers in the interval $[x_i, x_j]$ using $s \leq S$ servers, excluding the customers that are already covered by an external server that commutes on the interval $[x_{i'}, x_{j'}]$. (In the case $i' = j' = 0$, there is no such external server.)

Now, we compute all the values of $P[i; j; i'; j'; s]$. All cases with $j < i$ have empty intervals, and hence yield a profit $P[i; j; i'; j'; s]$=0. When $j \geq i$ we can distinct several cases. If the leftmost point i is covered by an external server or is not served at all, it holds that $P[i; j; i'; j'; s]$=$P[i + 1; j; i'; j'; s]$. Otherwise i is served by a server from s, say s_1, traveling on the interval $[x_i, x_l]$ with $i \leq l \leq j$. Total profit for this state then consists of three parts: (i) $p(l) = \sum_{i \in S(s_1)} p_i$, the profit of the set of customers $S(s_1)$ covered by s_1; (ii) $\alpha(l, s') = P[i+1, l, i, l, s']$, the profit of the servers traveling within the interval covered by s_1; and (iii) $\beta(l, s-s') =$

$P[l+1, j, i', j', s-s'-1]$, the profit that can be collected in the remaining interval by the remaining servers. The optimal value is then:

$$P[i; j; i'; j'; s] = \max_{l,s'}\{p(l) + \alpha(l, s') + \beta(l, s') | i \leq l \leq j, 0 \leq s' \leq s\}.$$

The state $P[i; j; 0; 0; S]$ yielding maximal profit gives the optimal solution. There are $O(n^4 S)$ states to be computed and each state requires $O(n^2 S)$ time. Overall running time is then $O(n^6 S^2)$.

Theorem 4.21. *MPLPP on the line can be solved in $O(n^6 S^2)$.*

As in the previous section, the algorithm for MPLPP on the line can be extended adding a factor n to the running time.

Corollary 4.22. *MPLPP on the circle can be solved in $O(n^7 S^2)$.*

NP-hardness of MPLPP on a tree follows from the fact that MPLPP on a star is NP-hard.

Theorem 4.23. *MPLPP on a star is NP-hard, even if for all $i \in N$: $q_i = Q$.*

Proof. It is easy to see that the proof of Theorem 4.19 is directly applicable to this problem. □

It follows that the problem is NP-hard on arbitrary graphs.

4.7 Conclusion

We were able to settle complexity of a number of variants of the PLPP and the PLP. Our results still hold when customers are weighted, as a customer will always be served when a server passes by. However, some interesting questions remain. It would, for instance, be interesting to study the influence of repair times on complexity. This seems to make the problems much harder. Indeed, when repair times are added for the customers, Lemma 4.15, which is crucial for the dynamic programming algorithm, no longer holds. Further, it is also not clear what happens when servers have

restricted capacity. And what if servers are not identical, meaning that they travel at different speeds or with different operating costs?

Epilogue

Almost 30 years after Charlemagne's death, his empire was divided into three parts (the treaty of Verdun); apparently, by that time, three servers were needed to guarantee loyalty of all counts...

Chapter 5

The periodic vehicle routing problem: a case study

This chapter deals with a case study which is a variant of the Periodic Vehicle Routing Problem (PVRP). As in the traditional Vehicle Routing Problem (VRP), customer locations each with a certain daily demand are given, as well as a set of capacitated vehicles. In addition, the PVRP has a horizon, say T days, and there is a frequency for each customer stating how often within this T-day period this customer must be visited. A solution to the PVRP consists of T sets of routes that jointly satisfy the demand constraints and the frequency constraints. The objective is to minimize the sum of the costs of all routes over the planning horizon. We develop different algorithms solving the instances of the case studied. Using these algorithms we are able to realize considerable cost reductions compared to the current situation.

5.1 Introduction

In this chapter we study a routing problem of a Belgian company collecting waste at slaughterhouses, butchers, and supermarkets. Planning of the routes occurs over a time period of several days (time horizon) in which customers are visited with different frequencies. For instance, supermarkets might request service every day, while for a small butcher one collection a week suffices. The resulting problem is a variant of the Periodic Vehicle Routing Problem (PVRP). As in the traditional Vehicle Routing Problem (VRP), customer locations each with a certain demand function are given, as well as a set of capacitated vehicles. In addition, the PVRP has a horizon, say T days, and there is a frequency for each customer stating how often within this T-day period this customer must be visited. A solution to the PVRP consists of T sets of routes that jointly satisfy the demand constraints and the frequency constraints. The objective is to minimize the sum of the costs of all routes over the planning horizon. Obviously, this problem is at least as hard as the VRP.

PVRP and variants

Several variants of the PVRP are described in literature. A classification of the different variants of the PVRP can be found in a survey by Mourgaya and Vanderbeck (2006). Different objective functions are distinguished, such as minimizing the distance traveled, the driving time, or total transportation cost; however also regionalization of routes, an even spread of workload over the vehicles, the number of vehicles, and service quality can be part of an optimization function. Differences also occur in the constraints which can be divided in three categories: constraints concerning (i) the planning of visits (different frequencies, restrictions on certain days, etc.), (ii) the type of demand (constant or variable; we return to this issue later), and (iii) the vehicles. Where the PVRP is mostly situated on tactical and operational level, Francis et al. (2006) include strategic decisions in their model: frequencies of service are variables within the model, and not given parameters. Another variant is the PVRP with intermediate

facilities which is described by Angelelli and Speranza (2002), Kim et al. (2006), and Alonso et al. (2008). Intermediate facilities are locations where vehicles can unload (or reload) and thus renew capacity during a route; this happens in our case, see Section 5.2.

Case studies

The PVRP is a relevant problem; it occurs for companies that have to carry out periodic repair and maintenance activities or that collect/deliver goods periodically. Blakely et al. (2003) describe a case for periodic maintenance of elevators at different customer locations. Further case studies concerning waste collection and road sweeping can be found in Beltrami and Bodin (1974) and Eglese and Murdock (1991). Claassen and Hendriks (2007) describe a milk collection problem where it is important that the goods are collected when fresh. For the collection of raw materials for a manufacturer of auto parts, on the contrary, a very long time horizon is considered, see Alegre et al. (2007). Hemmelmayr et al. (2008) investigate the periodic delivery of blood products to hospitals by the Austrian Red Cross. In this case, the regularity of deliveries is of utmost importance. Many other case studies are described in Francis et al. (2008) and the references contained therein.

Solution methods

The PVRP is situated on the border between tactical and operational planning, combining the classical VRP with planning over a time horizon. That is why solution methods often consist of two phases. Beltrami and Bodin (1974) consider two approaches. In a first approach, routes are developed and then assigned to days of the week; in a second approach customers are assigned to days in a first phase and in a second phase the routing problem for every single day is solved using classical techniques for solving VRPs. This second approach is used in many papers, such as Tan and Beasley (1984), Christofides and Beasley (1984), and Baptiste et al. (2002). Tan and Beasley (1984) first solve an assignment problem to assign customers to days such that total demand in each day does not exceed demand capacity while taking pairwise distances between customers

into account. After that stage they solve a VRP for each day in the planning horizon. Thus, they approach this problem as an extension of the assignment problem with a routing component. Christofides and Beasley (1984) on the other hand formulate the PVRP as a routing problem with a selection decision. Customers are ordered in descending order of "importance", depending on the demands, and then selected for a route on a certain day depending on the increase in total cost for the whole period. These approaches are the more classical solution strategies. Recent PVRP literature has focused on metaheuristic methods and mathematics- based approaches to solve the problem; we refer to Francis et al. (2008) for an overview.

The rest of this chapter is structured as follows. In section 5.2 we describe the case under consideration in further detail. In section 5.3 we give a mathematical formulation. In section 5.4 we propose a solution method and in section 5.5 we report some computational results and formulate a conclusion.

5.2 The case

5.2.1 A general description

As mentioned, we study a problem faced by a Belgium transportation company, which is responsible for the collection of waste at slaughterhouses, butchers, and supermarkets. This company, which we call company A for confidentiality reasons, has customers all over Belgium and in some areas of northern France.

Legislation that originated from the BSE-epidemic (Bovine Spongiform Encephalopathy, commonly known as mad-cow disease) in the nineties, stipulates that (i) there are 3 categories of animal waste, depending on the risk of containing BSE; (ii) waste from different categories has to be collected separately. Company A only collects waste from two categories: category 1 (high-risk waste) and category 3 (low-risk waste). All high-risk waste is collected in order to be destroyed, while low-risk waste can be

further processed into e.g. pet food. Vehicles assigned to collect high-risk waste cannot be used to collect low-risk waste and vice versa. In fact, this implies that company A has to solve two different instances; one instance for the periodic collection of high-risk waste and a second instance for the periodic collection of low-risk waste.
In the current planning process of company A, routes are constructed manually on a regular basis, e.g. every month, and, during that period minor modifications to the routes can be made depending on changes in the set of customers. Company A wishes to decrease the dependence upon human expertise and wants to automatize the planning procedure. Also, the management of company A wishes to plan the routes more efficiently in order to reduce travel time and travel distance. Several opportunities need to be explored: (i) can total driving time be decreased? (ii) can the routing be done using smaller vehicles? and (iii) is it possible to decrease the vehicle fleet? All of this could be possible through more efficient planning, but obviously, the same level of service towards the customers should be retained. All this also has environmental consequences due to a reduction in petrol and in vehicle use. It is clear that it would not be possible to change the vehicle fleet whenever the routing plan changes. Our routing plan, however, should allow the management to "assemble" a good vehicle fleet in the long term. In the short term, the current vehicle fleet must be respected when constructing a routing plan.

5.2.2 The low-risk waste instance: details

Here we describe some properties of the instance corresponding to the low-risk waste. First notice that the company could only provide us the addresses of the customers; using the Shortrec software (Ortec (2007)) the travel distances between the customers were computed and Figures 5.1 and 5.2 were generated. For the low-risk waste instance there are 48 customers, spread out over Belgium and northern France (see Figure 5.1). The planning period is one week (actually 6 days), and each customer requires a certain frequency of visit over the planning period. Table 5.1 gives an

overview of how often each frequency occurs. How these frequencies can be obtained is explained in Section 5.3. There are 5 different frequencies and a frequency of 4 days does not occur. Company A has 3 trucks available

Frequency (nr of visits required)	1	2	3	5	6
nr of customers	21	15	5	5	2

Table 5.1: *Frequencies low-risk waste*

for collecting low risk waste; with capacities 12, 22, and 26 tons respectively. Some customers are located in the center of a city and cannot be reached by a truck of 22 tons or bigger. There is a central depot where each route starts in the morning and ends in the evening. When a truck is fully loaded, the driver can unload at a disposal facility and continue his route. Notice that these disposal facilities can be seen as intermediate facilities, see Angelelli and Speranza (2002). Trucks do not need to return empty in the evening, they can also dispose of their load during the tour of the following day. Only when a truck does not drive on the following day it has to be emptied before returning to the depot. An affiliated company processes the waste and has one disposal facility where the trucks can unload 24 hours a day. Loading and unloading times depend on the volume. Legally, the maximum driving time for a driver is restricted to 90 hours within two weeks and the company restricts the daily driving hours to 10 hours. Notice that these are only driving hours, they do not include loading and unloading times.

5.2.3 The high-risk waste instance: details

The instance corresponding to the collection of high-risk waste contains 262 customers, distributed all within Belgium (see Figure 5.2). The planning period is 2 weeks (10 days), and again customers have certain frequencies of visit within that period. In Table 5.2 we give an overview of the different frequencies for the high-risk waste instance. The capacities of the three trucks available are 9, 12, and 12 tons respectively. The collected waste

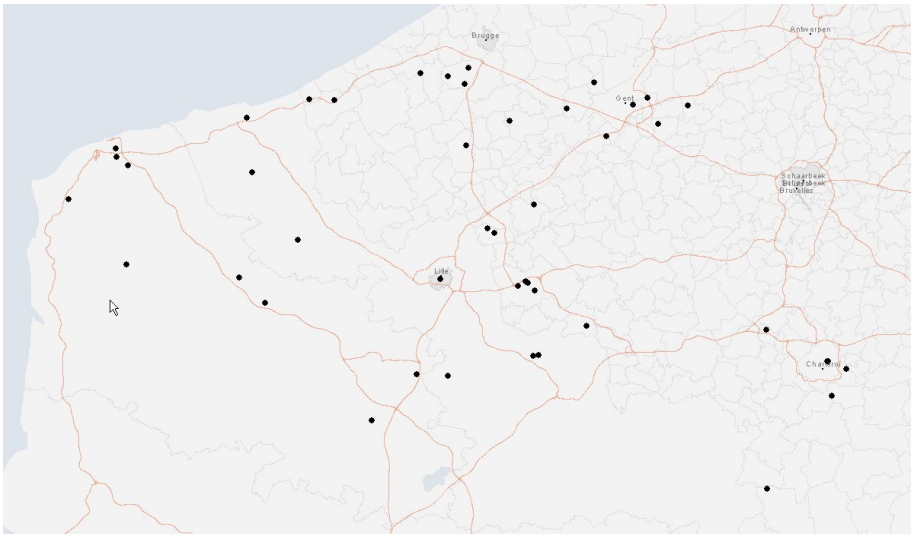


Figure 5.1: *Customers of category* 3*: instance* 1

Frequency (nr of visits required)	1	2	4
nr of customers	62	186	14

Table 5.2: *Frequencies high-risk waste*

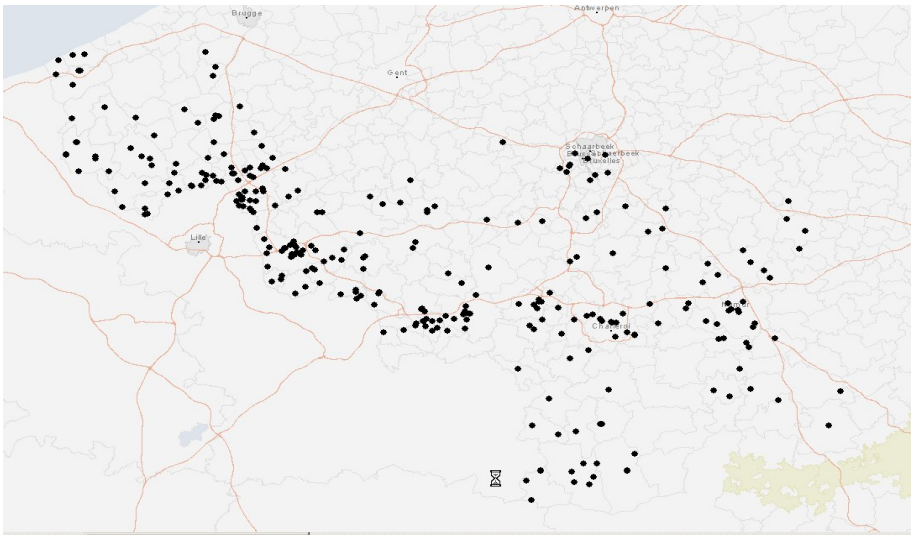


Figure 5.2: *Customers of category* 1*: instance* 2

must be delivered at two disposal facilities of an external company. Notice that time windows apply to these facilities. Loading and unloading times are constant, 10 resp. 30 minutes. Restrictions on driving hours and depot locations are the same as for low-risk waste.

5.3 Model

An important aspect of our model are the "visit-frequencies". Based on historical data, it can be accurately predicted how much waste a customer produces per day, and how much waste can be stored at a customer's site. On the basis of these data and in consultation with the customer, a *visit-frequency* is computed for each customer: the number of times a collection is done during the planning period. We also compute a demand (q_i) depending upon this frequency. As an example, consider some customer i that generates 10 tons of waste per day, and has a storage capacity of 30 tons, while the planning period is $T = 5$. We compute the corresponding visit-frequency f_i for this customer i as follows: $f_i = \lceil \frac{10T}{30} \rceil = 2$. More

in particular, the visits must be spread well-balanced over the planning period. Hence, this customer can be visited on days 1 and 3, but also on days 1 and 4, 2 and 4, 2 and 5, and 3 and 5; but not on any two consecutive days. The demand of this customer equals $q_i = \frac{10T}{f_i} = 25$. Notice that the actual amount of waste collected at customer i may differ from the predicted amount q_i.

For every frequency, we define all possible combinations of days in the planning period, and we call them scenarios. Then, for every customer we need a constraint that selects exactly one scenario from the set of feasible scenarios. How exactly the scenarios are assigned to customers will become clear further in this section. We first give some notation.

5.3.1 Notation

We define the network $G = (V, A)$. The customers to be visited are represented by vertices 1 to N, the depot is represented by node 0 and the disposal facilities by nodes $N + 1$ to $N + M$. Thus vertex set $V = \{0, 1, 2, \ldots, N, N + 1, \ldots, N + M\}$ and A is the arc set with for each arc (i, j) a travel distance d_{ij} and a travel time c_{ij} $(i, j \in V)$. The planning period has a length of T days and a customer is visited within this period according to a certain scenario $c \in C$. Essentially, a scenario is a set of days within $\mathcal{T} = \{1, \ldots, T\}$; choosing a scenario for a customer means that the customer is visited during these days. We let f^c equal the number of visiting days in scenario c. We are given a number of vehicles K, each with a certain capacity Q_k. For each customer $i \in \mathcal{N} = \{1, \ldots, N\}$, a frequency f_i, a quantity q_i and a loading time l_i are given. Quantity q_i is the quantity to be collected at each visit; this number is based on the average amount of waste that is collected at each visit (see our discussion earlier). We can use this average amount to approximate reality because visits are spread evenly over the planning period. Further, a driver may not drive longer than D_d hours a day, and no longer than D_T hours in the total planning period of T days. Disposal facilities can only be visited within their time windows $[r_i, s_i]$, $i \in \mathcal{M} = \{N + 1, \ldots, N + M\}$. Finally, a_{ct} is equal to 1 if

day $t \in \mathcal{T}$ is visited within scenario $c \in C$ and 0 otherwise. We then define the following five sets of variables: x_{ijkt} is a binary variable which is equal to 1 if customer $i \in V$ is visited after customer $j \in V$ by vehicle $k \in \mathcal{K}$ at day $t \in \mathcal{T}$, 0 otherwise; y_{ic} is a binary variable which is 1 if customer $i \in V$ is visited according to scenario $c \in C$ and 0 otherwise; let L_{ikt} and T_{ikt} be the total load and the total travel time, respectively, of vehicle $k \in \mathcal{K}$ at day $t \in \mathcal{T}$ after having visited customer $i \in V$; and finally, the load of a vehicle $k \in \mathcal{K} = \{1, 2, \ldots, K\}$ at the beginning of day $t \in \mathcal{T}$ is denoted by S_{1kt} and the load at the end of the day by S_{2kt}.

5.3.2 The model

$$\text{(PVRP) Minimize} \sum_{i \in V} \sum_{j \in V} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} d_{ij} x_{ijkt} \tag{5.1}$$

subject to

$$\sum_{c \in C: f^c = f_i} y_{ic} = 1 \qquad \forall i \in \mathcal{N}, \quad (5.2)$$

$$\sum_{j \in V} \sum_{k \in \mathcal{K}} x_{ijkt} - \sum_{c \in C} a_{ct} y_{ic} = 0 \qquad \forall i \in \mathcal{N}; \forall t \in \mathcal{T}, \quad (5.3)$$

$$\sum_{i \in V} x_{ihkt} - \sum_{j \in V} x_{hjkt} = 0 \qquad \forall h \in V; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.4)$$

$$\sum_{j \in \mathcal{N}} x_{0jkt} \leq 1 \qquad \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.5)$$

$$\sum_{i \in V} \sum_{j \in V} \sum_{t \in \mathcal{T}} d_{ij} x_{ijkt} \leq D_T \qquad \forall k \in \mathcal{K}, \quad (5.6)$$

$$\sum_{i \in V} \sum_{j \in V} d_{ij} x_{ijkt} \leq D_d \qquad \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.7)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijkt} \leq |S| - 1 \qquad \forall k \in \mathcal{K}; \forall t \in \mathcal{T}; S \subseteq \mathcal{N}; |S| \geq 2, \quad (5.8)$$

$$(L_{ikt} + q_j - L_{jkt}) \leq (1 - x_{ijkt}) Q_k \qquad \forall k \in \mathcal{K}; \forall t \in \mathcal{T}; \forall i, j \in \mathcal{N}, \quad (5.9)$$

$$L_{ikt} \leq Q_k \qquad \forall i \in \mathcal{N}; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.10)$$

$$L_{ikt} = 0 \qquad \forall i \in \mathcal{M}; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.11)$$

$$S_{2kt} = S_{1k(t+1)} \qquad \forall k \in \mathcal{K}; \forall t \in \{1, \ldots, T-1\}, \quad (5.12)$$

$$L_{ikt} - S_{2kt} \leq (1 - x_{i0kt}) Q_k \qquad \forall i \in V; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.13)$$

$$S_{1kt} - L_{jkt} \leq (1 - x_{0jkt}) Q_k \qquad \forall j \in V; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.14)$$

$$S_{2kt} = 0 \qquad \forall t \in \{fridays\}; \forall k \in \mathcal{K}, \quad (5.15)$$

$$S_{2kt} \leq Q_k z_{kt} \qquad \forall k \in \mathcal{K}; \forall t \in \{1, \ldots, T-1\}, \quad (5.16)$$

$$1 - \sum_{j \in V} x_{0jk(t+1)} \leq 1 - z_{kt} \qquad \forall k \in \mathcal{K}; \forall t \in \{1, \ldots, T-1\}, \quad (5.17)$$

$$T_{ikt} + l_i + c_{ij} - T_{jkt} \leq (1 - x_{ijkt}) M \qquad \forall k \in \mathcal{K}; \forall t \in \mathcal{T}; \forall i, j \in V, \quad (5.18)$$

$$r_i x_{ijt} \leq T_{ikt} \leq s_i x_{ijkt} \qquad \forall k \in \mathcal{K}; \forall t \in \mathcal{T}; \forall i, j \in \mathcal{N}, \quad (5.19)$$

$$x_{ijkt}, z_{kt} \in \{0, 1\} \qquad \forall i, j \in V; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}, \quad (5.20)$$

$$y_{ic} \in \{0, 1\} \qquad \forall i \in \mathcal{N}; \forall c \in C, \quad (5.21)$$

$$L_{ikt}, T_{ikt}, S_{1kt}, S_{2kt} \geq 0 \qquad \forall i \in V; \forall k \in \mathcal{K}; \forall t \in \mathcal{T}. \quad (5.22)$$

This model finds a scenario for every customer and a set of routes for each day of the planning period such that total travel distance is minimized. The first constraints (5.2) make sure that exactly one scenario is selected for every customer and in such a way that within this scenario the customer is visited according to its frequency. A customer then will be visited on the days of the selected scenario; this is ensured by constraints (5.3). Constraints (5.4) make sure that when a vehicle arrives at a customer, it also leaves from that customer. Constraints (5.5) impose that each vehicle can be used at most once every day. Constraints (5.6) and (5.7) keep the number of driving hours for every vehicle within the restrictions for the whole planning period, and within the daily restrictions, respectively. Constraints (5.8) are subtour elimination constraints. Correct counting of vehicle loads is ensured by constraints (5.9), and constraints (5.10) keep the vehicle load within the capacity. Constraints (5.11) impose that vehicles are empty when they have visited a disposal facility. A vehicle does not need to dispose of all of its load at the end of the day. It follows that the load of a vehicle at the end of a day needs to be equal to the load of that vehicle at the start of the following day, this is ensured by constraints (5.12) to (5.14). At certain moments in the week though vehicles do need to unload at the end of the day, e.g at the end of the week (5.15) and when a vehicle is not used on the next day, see constraints (5.16) and (5.17). Equations (5.18) and (5.19) make sure that time windows for the disposal facilities are not violated. Finally, constraints (5.20) through (5.22) impose binary conditions and nonnegativity conditions on the variable set.

Solving this model in order to obtain an optimal solution for a small data set of less than 10 customers already takes a very long time. That is why in the following section we develop a heuristic solution approach.

5.4 Solution approach

We study methods that consist of two phases: first, customers are assigned to days, and second, a VRP-instance is solved. In subsection 5.4.1 we investigate methods that first assign customers to days, and next solve

a VRP-instance. We consider two ways of assigning customers to days, namely striving for an "even spread" of the number of visits on a day, or use a geographically based clustering approach. Finally, we describe how we use ILOG Dispatcher to solve a VRP-instance. In subsection 5.4.2 we describe a method that first solves a VRP-instance, and then assigns customers to days.

5.4.1 First assign customers to days, then route

Assigning customers to days: algorithm ES

Here, we focus on the spread of visits over the planning horizon. We solve the following problem (problem ES). We are given a set V of customers, and each customer i has an associated frequency f_i. Further we have a set C of scenarios, where a scenario consists of a set of f^c days, meaning that in that scenario c visits are performed on f^c different days. These scenarios need to be assigned to customers such that the frequency of each customer equals the frequency of the scenario assigned to the customer and the maximum number of customers visited on each day of the planning horizon is minimized. We refer to this problem as problem ES and we claim the following.

Fact 5.1. *Problem ES is NP-hard.*

Proof. We prove NP-hardness of problem ES by a reduction from Exact Cover by 3-sets (X3C) to the decision problem of ES. In X3C a finite set X containing $3n$ elements and a collection C of 3-element subsets of X are given. The question is whether there exists a subset C' of C such that every element of X occurs in exactly one triple of C'. This problem has been proven to be NP-complete by Garey and Johnson (1979). For every instance of X3C we can create an instance of problem ES as follows. There are n customers, each with frequency f_i equal to 3. The planning period lasts $3n$ days and each triple from C corresponds to a scenario consisting of three days corresponding to the elements of the triple. Now, if an assignment of scenarios to customers can be found such that exactly one

customer is visited on each day (evenly spread), then also a solution for X3C exists, and vice versa. □

Thus, this proves that in general problem ES is a hard problem to solve. However, for our purposes, due to the relatively small size of the problem (we have $T = 5$, $|C| = 15$, $N = 48$ for the low-risk waste instance; $T = 10$, $|C| = 20$, $N = 262$ for the high-risk waste instance) we can solve problem ES in reasonable time using the following integer program (IP). We define parameter a_{ct}, which is equal to 1 if day t is visited in scenario c, and 0 otherwise. Variable y_{ic} is equal to 1 if customer i is assigned scenario c and 0 otherwise. The variable z_t is an integer variable representing the number of customers visited on day t.

$$\text{Minimize } w \tag{5.23}$$

subject to

$$w \geq z_t \qquad \forall t \in \mathcal{T}, \tag{5.24}$$

$$\sum_{c \in C: f^c = f_i} y_{ic} = 1 \qquad \forall i \in \mathcal{N}, \tag{5.25}$$

$$\sum_{i \in V} \sum_{c \in C: f^c = f_i} a_{ct} y_{ic} = z_t \qquad \forall t \in \mathcal{T}, \tag{5.26}$$

$$y_{ic} \in \{0, 1\} \qquad \forall i \in \mathcal{N}; \forall c \in C, \tag{5.27}$$

$$z_t, w \in \mathbb{Z} \qquad \forall t \in \mathcal{T}. \tag{5.28}$$

The goal of this IP is to assign a scenario to every customer such that visits are spread evenly over the planning period. We enforce this by minimizing the maximal number of customers visited on a day during the planning period (5.24). In that way we will visit more or less the same number of customers on each day. Constraint (5.25) makes sure that every customer is visited according to exactly one scenario that matches its frequency. We can solve this IP optimally using ILOG Cplex 10.2.
Notice that the location of the customers is completely ignored in this approach. Clearly, this implies that there is a risk that customers positioned

close to each other are scheduled on different days. In the next section, we assign scenarios to customers such that this disadvantage is taken into account.

Assigning customers to days: algorithm CL

In a second approach, algorithm CL (CLuster), we partition the customers using the "k-medoids clustering" algorithm before assigning scenarios to customers. This clustering algorithm partitions the locations into k clusters and attempts to minimize the squared error, i.e. the squared distances between points labeled to be in a cluster and a point designated as the center of that cluster. The k-medoids approach chooses data points as centers. We prefer to use this particular clustering method because we start from a given distance matrix rather than from the locations. We use the k-medoid method as it is defined in the C Clustering Library by de de Hoon et al. (2008). The algorithm starts with an arbitrary selection of k data points that will act as centers (medoids) and then tries to improve by swapping the medoids with other points.

We apply this method for several values of k, and then we solve (5.29)-(5.32) (which is a modification of (5.23)-(5.28)), which takes clusters into account when assigning customers to days. Define parameter o_{il}, which is equal to 1 if customer $i \in \mathcal{N}$ is in cluster $l \in 1, \ldots, k$, with k the number of clusters, and 0 otherwise. z'_{tl} is a variable which is equal to 1 if and only if a customer of cluster l is visited on day t. Now, our goal is to minimize the total sum of the number of clusters visited each day. A cluster is visited if a customer of that cluster is visited. Customers that belong to the same cluster will thus be assigned to the same day as much as possible.

$$\text{Minimize} \sum_{t \in \mathcal{T}} \sum_{l \in \{1,\ldots,k\}} z'_{tl} \tag{5.29}$$

subject to

$$\sum_{c \in C: f^c = f_i} y_{ic} = 1 \qquad \forall i \in \mathcal{N}, \qquad (5.30)$$

$$a_{ct} y_{ic} o_{il} \leq z'_{tl} \qquad \forall i \in \mathcal{N}; \forall c \in C; \forall t \in \mathcal{T}; \forall l \in \{1, \ldots, k\}, \qquad (5.31)$$

$$y_{ic}, z'_{tl} \in \{0, 1\} \qquad \forall i \in \mathcal{N}; \forall c \in C; \forall t \in \mathcal{T}; \forall l \in \{1, \ldots, k\}. \qquad (5.32)$$

Notice that a potential risk associated to the outcome of model (5.29)-(5.32) is that the driving time needed to visit all the customers assigned to a certain day exceeds D_d. To prevent this, a constraint on the number of customers visited on each day is added; we return to this issue in section 5.2.

Routing using ILOG Dispatcher

Having assigned all customers to scenarios, either using ES or CL, we know which customers need to be visited every day. Thus, for each day in the planning horizon we now need to solve a, more or less standard, VRP. Some specific constraints do need to be taken into account, such as the facts that vehicles have different capacities, vehicles can dispose of their load during the day and then continue the route (they can collect more than their capacity on one day), some customers can only be reached by small vehicles, vehicles do not need to unload at the end of the day, and the number of driving hours per vehicle per day is limited. A solution algorithm for this VRP is implemented using ILOG Dispatcher 4.4, which is a C++ library based on ILOG Solver and which offers features specifically adapted to solving problems in vehicle routing. We implement all the standard VRP constraints, as well as the problem-specific constraints mentioned above.

In the resulting routing algorithm, an initial solution to the VRP is constructed using a standard savings heuristic, which makes a trade off between more vehicles with shorter routes and fewer vehicles with longer routes. Then, ILOG Dispatcher performs neighborhood search to improve this solution. Both intra-route and inter-route neighborhoods are considered. As intra-route neighborhoods, 2-Opt and Or-Opt are implemented.

ILOG Dispatcher also interchanges between routes: the "relocate" neighborhood (inserting a customer in another route); the "exchange" neighborhood (swapping two customers from different routes); and the "cross" neighborhood (exchanging the ends of two routes). We refer to ILOG Dispatcher user's manual (ILOG (2005)) for a more elaborate description. We have restricted ourselves here to these methods to improve the routes.

5.4.2 First route, then assign customers to days: algorithm MR

In this approach, called algorithm MR (Mega Route), we first construct large routes visiting all the customers and the disposal facilities. To accomplish this, we use the same VRP heuristic as described in the previous section. Then, on the basis of these routes, customers are assigned to days using model (5.29)-(5.32). Customers belonging to the same route are then visited on the same day as much as possible. Then, we resolve a VRP for each day in the planning horizon in order to obtain better routes. Algorithm MR is similar to algorithm CL, where we first cluster the customers, since geographically close customers tend to end up being visited on a same day. Routing can yield a very different grouping of the customers though.

5.5 Computational results

The different algorithms were implemented in Microsoft Visual C++ 2005, in combination with ILOG CPLEX 10.2 and ILOG Dispatcher 4.4. All algorithms were run on a personal computer with a 2.80 GHz Intel Pentium IV processor and 504 MB of RAM. Notice that we have not been concerned with running times of our approach. This is mostly because the application did not enforce strong limitations on the amount of computing time used. In subsections 5.5.1 and 5.5.2 computational results are given for the low-risk waste instance and for the high-risk waste instance, respectively. In subsection 5.5.3 we discuss the performance of the different methods for the two instances.

5.5.1 The low-risk waste instance: results

In Table 5.3 some computational results are shown. Applying algorithm ES, we find a set of routes requiring 3 vehicles. On several days only 2 of them are being used. Notice that visits are well spread over the week, as each day a comparable distance is traveled. Between vehicles though, there is a rather uneven workload. Total traveling time is 5859 minutes in 6 days.

As can be seen in Table 5.3, a solution by algorithm CL is obtained using all three vehicles. Comparing this solution with the previous one we can see that the variation in route length over the days is higher and that total traveling time decreased to only 5551.
The last columns in Table 5.3 show the results of algorithm MR. Here also three vehicles are used and the total traveling distance is rather high (5934). Thus, routing the customers before assigning them to the different days in the planning horizon does not yield a good solution for this instance.
Concluding, this suggests that using the geographic structure yields a better routing plan. Running times vary from about 1 hour for ES, to a few hours for CL and up to 48 hours for MR. We experience a large variation in time needed by the VRP solver. Finally notice that the current routes used in practice have a total traveling time of 6565; using algorithm CL we could improve them with 15.5%.

5.5.2 The high-risk waste instance: results

Let us now apply the same algorithms to the instance for collection of high-risk waste. The results are summarized in Tables 5.4 and 5.5. Using the ES algorithm, every day the same number of customers is visited, unloading only occurs once in two days and on Fridays and only two vehicles are required. Total travel time is then 10272 for two weeks.

Applying algorithm CL, two large clusters are obtained. Based on these clusters we assign all the customers to the days of the planning horizon. This results in a very uneven spread of the customers; on some

	ES				CL				MR			
Vehicle Day	12T	22T	26T	TOT	12T	22T	26T	TOT	12T	22T	26T	TOT
1	452	391	240	1083	495	413	476	1384	339	235	567	1141
2	509		551	1060	409		279	688	339	252	220	811
3	511	378	240	1129	416	439	237	1092	439	288	588	1315
4	427	280	482	1189	512	374	214	1100	354	364	276	994
5	561		482	1140	460	163	406	1029	584	510	321	1415
6			258	258			258	258			258	258
				5859				5551				5934

Table 5.3: *Results low-risk waste (traveling time in minutes)*

Algorithm	ES			CL			
Vehicle Day	9T	12T	TOT	9T	9T	12T	TOT
1	558	350	908	311		385	696
2	662	495	1157	387		495	882
3	596	342	938				
4	587	570	1157	579	583	398	1560
5	450	549	999	551			551
6	308	599	907	445		423	868
7	641	434	1075	494	619	115	1228
8	599	334	933				
9	592	607	1199	585	577	356	1518
10	450	549	999	311		116	427
			10727				7730

Table 5.4: *Results high-risk waste (traveling time in minutes)*

Algorithm	CL-limited customers per day				MR-limited customers per day		
Vehicle Day	9T	9T	12T	TOT	9T	12T	TOT
1	540		279	819	527		527
2	510		655	1165	452	673	1125
3	506		225	731	504		504
4					538	675	1213
5	544		675	1219			
6	541		289	830	577		577
7	551	203	519	1273	455	673	1128
8	542		228	770	546		546
9					514	686	1200
10	591		615	1206	257	578	835
				8013			7655

Table 5.5: *Results high-risk waste (traveling time in minutes)(contd.)*

days 100 customers are visited, and as a result 3 vehicles are required, and on other days no customers are visited. Total traveling time is then 7730, which is clearly much better than ES; however, this leads to the usage of 3 vehicles. In order to see whether the instance could be solved using two vehicles only, we add a constraint to (5.29)-(5.32) bounding the number of customers assigned to one day. Solving this model, we obtain routes with a total travel time of 8013 and using only 2 vehicles on most days.
Finally, we also apply algorithm MR to this instance. We create 2 giant routes, visiting all the customers and the disposal facilities. Again, when dividing the customers over the days, we obtain a highly uneven spread of customers. Visits are only performed on 4 of the 10 days, with up to 171 customers on one day. In fact, as mentioned as a potential risk in section 5.4.1, this is not feasible given the available driving time; if we relax these constraints total distance traveled is 6591. Again, we need to limit the number of customers visited per day, and we set this limit equal to 75, which is more or less the amount of customers that can be served on one day by 2 vehicles. This yields a total traveling time of 7655 and utilizes 2 vehicles. More details are shown in Table 5.5.
Concluding, for this instance it is important to take into account geography. Running times are limited to seconds for all three algorithms. Compared to the original routes which take 8434 minutes, algorithm ES performs badly; MR improves the currently used routes with 9% and uses one vehicle less.

5.5.3 Discussion

Depending on the instance, the methods used above perform differently. Clustering the customers yields a better result for both instances compared to algorithm ES, but the effect is larger for the high-risk waste instance. Algorithm MR performs comparably as CL for this instance but a lot worse than ES and CL for the low-risk waste instance. The reason for this phenomenon is that there are customers in the low-risk instance with a very high frequency (see Table 5.1) and these customers are rather spread over the country (see Figure 5.1). Ignoring the geographic locations has

then only a small impact on the solutions, as on each day the vehicles have to travel in different directions anyway. There is a large difference in the frequencies of the high-risk waste instance compared to the low-risk waste instance (see Tables 5.1 and 5.2). Many customers require the same frequency, and no customer has a very high frequency (maximally 4 visits in 10 days). Together with the fact that the instance is larger, this explains why clustering has a vast effect on the solutions.
As mentioned before, we obtain routes that might yield a rather uneven workload for the different vehicles. Company A does not consider this as being a problem. We propose routes using three vehicles for the small instance. Not every day, though, all vehicles are needed, thus it might be interesting to consider renting a vehicle and driver at an external company for the days necessary. Changing the volume of the vehicles in order to diminish the number of vehicles does not seem really useful. For the low-risk waste instance a vehicle with capacity of 12 tons must be available for customers not reachable with a larger vehicle. For the high-risk waste instance, the volumes to be picked up are rather small and unloading is only necessary after two days. It is not the volume but the time restrictions that have the highest impact on travel times. That is also the reason why running times of the algorithms differ a lot between the low-risk waste instance and high-risk waste instance. In the low-risk waste instance the volumes to be picked up are higher such that vehicles unload several times during the day. This complicates the VRP model and increases running times.

5.6 Conclusion

We consider a problem occurring in practice, and we modeled it as a PVRP. Using different approaches (including ILOG's dispatcher), we were able to improve the current routes of Company A, using 1 vehicle less for one of the instances. This not only means a reduction in cost due to the gain in travel time, but also a reduction in wage costs and material costs. We use rather simple algorithms to assign the customers to the days and to

solve the VRP's. The use of clustering customers depends highly on the dispersion of the customers and on their frequencies. We deal with this instance of the PVRP by considering the problems of assigning customers to days and routing the customers independently.
For those interested to further study the problem, the instances can be found on the following website:
http://www.econ.kuleuven.ac.be/public/N05012/.

Chapter 6

On a motion control problem for a placement machine

Assembling printed circuit boards efficiently using automated placement machines is a challenging task. Here, we focus on a motion control problem for a specific type of placement machines. More specifically, the problem is to establish movement patterns for the robot arm, the feeder rack, and - when appropriate - the worktable, of a sequential pick-and-place machine. In this chapter we show that a (popular) greedy strategy may not always yield an optimum solution. However, under the relevant Tchebychev metric, as well as under the Manhattan metric, we can model the problem as a linear program, thereby establishing the existence of a polynomial time algorithm for this motion control problem. Finally, we give experimental evidence that computing optimal solutions to this motion control problem can yield significantly better solutions than those found by a greedy method.

6.1 Introduction

Assembling printed circuit boards efficiently using automated placement machines is a challenging task. The ever increasing need for competitiveness means that improving the throughput of production lines is an important topic in this industry. It follows that investigating optimization problems for whole production lines as well as for individual machines remains a relevant task.

There are many types of different placement machines; for a more extensive discussion of different types of machines we refer to Grunow et al. (2004), and Egbelu et al. (1996), where a classification is proposed depending upon which parts of the machine can move. One possible categorization is to divide placement machines into two categories: sequential machines (machines in which each component is handled sequentially) and concurrent machines (machines in which components can be handled concurrently). For instance, machines featuring a rotating turret or carousel fall under the latter type. We restrict our attention here to sequential placement machines. This type of placement machines can be described as follows. It consists of 3 basic parts:

- a worktable. The worktable carries the printed circuit board and is able, in its most general form, to move in the x-direction and in the y-direction.
- a feeder rack. The feeder rack is a bar that contains feeders in which the components are stored. Notice that a feeder stores components of a single type. The feeder rack can move in the x-direction only.
- a robot arm. This is a device that transports the components from the feeder rack to the appropriate location above the board; it is able to move in the x-direction and in the y-direction.

Such a placement machine is described in, for instance, Ayob and Kendall (2005) (where the worktable can only move in the x-direction) and in Altinkemer et al. (2000) (where the worktable is stationary).

Now, in order to operate any placement machine, several decisions must be made. There are various hierarchies of decision making, see Crama et al. (2002) for a discussion of this subject. However, given a single machine and a single board, three basic problems need at least be addressed:

- the component sequencing problem: determine a sequence of the given locations on the board where the components will be placed,
- the feeder assignment problem: determine where the feeders are located in the feeder rack, and
- the component retrieval problem: determine for each component to be placed, from which feeder it will be retrieved.

Each of these problems has been studied extensively in the literature: early references to each of these three problems include Ball and Magazine (1988), Leipälä and Nevalainen (1989), Crama et al. (1996); we refer to Crama et al. (2002) and the references contained therein for more information concerning these problems.

In this chapter we focus on a motion control problem for a sequential placement machine as described earlier. Thus, we will assume that each of the problems mentioned above has been solved. In addition, we assume (unless explicitly stated otherwise) that the robot arm can carry at most one component at any given moment in time. At first sight one may then wonder what is left to decide. However, before the machine starts actually inserting components, we need to establish the movement patterns of the 3 parts that are capable of moving: the robot arm, the feeder rack, and the worktable of the machine. This should be done in such a way that the machine finishes its last operation as soon as possible. We call this problem the motion control problem. Thus, the motion control problem for a single sequential pick-and-place machine that we address in this chapter can be described as follows. Given the locations on a board and a corresponding placement sequence, and given the location of each component in the feeder rack, the problem is to determine pick positions and place positions so that

the last placement operation is executed as soon as possible. Our goal is to minimize the total assembly time for a single board.

Of course, for some sequential machines this problem is nonexistent. Indeed, if both the feeder rack and the worktable cannot move, movement of the robot arm is completely dictated by the solution to the three problems described above. Also, if the machine's technology is such that it features a fixed pick position and a fixed place position (i.e., each component is picked (placed) at the same prescribed position) the movement of the robot arm easily follows, as well as the movement of the worktable and the feeder rack. However, the motion control problem becomes interesting when there are no fixed pick and place positions, and at least two of the three parts are capable of moving. Indeed, in Su et al. (1995), a so-called *dynamic* pick and place model is introduced in which the possibility of dynamic pick and place positions is investigated.

Notice that we do not address a specific placement machine; to achieve competitive advantages, the technologies for pick-and-place operations are subject to constant change and refinement. Instead, our results apply to any (hypothetical) sequential placement machine featuring multiple moving parts. More generally, any situation where a transporting device needs to bring a set of items able to follow some movement pattern to a given set of locations also able to follow some movement pattern, falls under our scope.

Related literature

As mentioned, the problem of finding good operational solutions for a single placement machine has been actively investigated in literature. The motion control problem is first described in Su et al. (1995) who take into account the possibility of not restricting the pick positions and the place positions to given locations. They propose a greedy strategy to solve the resulting motion control problem and give computational evidence for the gain of this dynamic pick and place model compared to the setting with fixed pick and place positions. Further studies, that also involve the computation of a feeder assignment and a component placement sequence,

are presented in Su and Fu (1998), Su et al. (1998), Wang et al. (1998), and van Hop and Tabucanon (2001b). van Hop and Tabucanon (2001a) and Ayob and Kendall (2005) each further develop a method for the motion control problem based on dynamic pick and place positions.

Motion planning has also received significant attention from the field of robotics, see e.g. Latombe (1991) for an overview. Here the emphasis is often on finding a motion plan (or a *path*) for a robot in some environment. Also, there is some literature that deals with classical routing problems such as the TSP, where the customers to be visited are known to move, and the salesman needs to take this into account, see for instance Helvig et al. (2003), and the references contained therein. Asahiro et al. (2005), inspired by an application in robot navigation, deal with a similar problem, a variant of the *Vehicle Routing Problem* where moving elements need to be grasped one by one before they move out of the reachable region of the robot arm. The goal is to pick as many elements as possible. As far as we are aware however, the complexity of the specific motion control problem discussed here has not been answered before.

Our contribution

(i) We provide an example in which it is beneficial for a (moving) feeder rack to wait, thereby postponing the next picking moment. This example shows that GREEDY methods (see Section 6.2) do not always yield an optimal solution to the motion control problem (even in the case of a stationary worktable). The example is valid for each distance metric $(dx^p + dy^p)^{1/p}$ with $p > 0$ (where dx (dy) is the distance traveled in the x (y) direction), more specifically, the example is valid for $p = 1$ (the Manhattan norm), for $p = 2$ (the Euclidean norm), and for $p = \infty$ (the Tchebychev norm). Notice that the latter norm is quite common for placement machines.

(ii) We show that the motion control problem is solvable in polynomial time for the relevant Tchebychev norm by formulating the problem as

a linear program. We also exhibit a special case of the motion control problem where a GREEDY method delivers an optimal solution.

(iii) We demonstrate that for randomly generated instances there is a significant difference between optimal solutions and solutions found by GREEDY methods. This difference partly depends on the ratio of the speed of the robot arm and the feeder rack. For some of the instances we considered the quality of a solution found by a GREEDY method may be significantly worse than the value of the optimum.

Remark

In our attempt to model the moving parts of a placement machine, we make assumptions that are not precisely fulfilled in practice. For instance, we assume a constant speed for each moving part; hence, we do not account for effects resulting from acceleration, and deceleration. For a description of the technical issues related to operating a placement machine we refer to van Gastel et al. (2004).

6.2 A problem description, a method, and an instance

In this section we further describe the problem, and we sketch a class of solution methods for the motion control problem that we call GREEDY methods. Recall that we assume that the component sequencing problem, the feeder assignment problem, and the component retrieval problem have been solved. In other words, the input to the motion control problem consists of (i) a sequence of n locations on the board, (ii) the position of the corresponding components in the feeder rack, and (iii) the starting configuration as well as the speeds of the robot arm, feeder rack, and worktable.

To facilitate the problem description we assume in this section that times needed for picking a component and times needed for placing a component can be ignored (notice that it is not difficult to include nonnegative picking and placing times in our methods and models, see sections 6.3

and 6.4). We assume that all movements occur in two-dimensional space, and hence, a position is completely specified by its x-coordinate and its y-coordinate. Also, we assume that the feeder rack coincides with the x-axis, i.e., all y-coordinates of picking positions equal zero. Finally, in order to facilitate the description of a GREEDY method, we first assume here that there are no physical obstructions for the movement patterns of robot arm, feeder rack, and worktable; we will come back to this issue later. We use the following notation:

for $i = 1, 2, ..., n$:

- (xp_i, yp_i): i-th placement position, i.e., the position where component i is placed by the robot arm onto the board,
- $(xs_i, 0)$: i-th pick position, i.e., the position where component i is picked by the robot arm from the feeder rack, and
- t_i^{place} (t_i^{pick}): moment in time when component i is placed (picked).

A feasible solution to the motion control problem amounts to finding values for these variables that correspond to achievable movement patterns. Further we use the following notation:

for $i = 1, 2, ..., n$:

- $(xb_i(t), yb_i(t))$: the position of the location on the board where component i needs to be placed at time t,
- $(xf_i(t), yf_i(t))$: the position of the location where component i is stored in the feeder rack at time t, and
- V_a (V_f, V_b): speed of the robot arm (feeder rack, board).

We call a method for the motion control problem a GREEDY method when, given the moments in time when the previous events occurred, the next event occurs as soon as possible. There are $2n + 1$ ordered events in the motion control problem: picking component i, placing component i ($i = 1, ..., n$), and returning to the starting position for the robot arm.

To describe a GREEDY method, let us for the moment assume that, for some i, $1 \leq i < n$, we know the i-th placement position (xp_i, yp_i), the corresponding time t_i^{place}, and that we also know the $(i+1)$-st pick position $(xs_{i+1}, 0)$, and its corresponding time t_{i+1}^{pick}. Observe that we then also know $(xb_{i+1}(t_i^{place}), yb_{i+1}(t_i^{place}))$ and $(xf_{i+2}(t_{i+1}^{pick}), yf_{i+2}(t_{i+1}^{pick}))$, i.e., the position of the location where the $(i+1)$-st component needs to be placed at time $t = t_i^{place}$, and the location in the feeder rack of component $(i+2)$ at time $t = t_{i+1}^{pick}$.

Given the i-th placement position, and its corresponding time, and given the $(i+1)$-st picking position, and its corresponding time, we now show how a GREEDY method computes the $(i+1)$-st placement position, the $(i+2)$-nd picking position, as well as their corresponding times. Applying this computation, starting with a given initial state, for $i = 1, 2, \ldots, n-1$ iteratively, gives us a solution to the motion control problem. This is done as follows. At $t = t_i^{place}$, we let the worktable move such that the location of the $(i+1)$-st placement location travels towards the $(i+1)$-st picking position. There are two possibilities. Either the board location arrives at the $(i+1)$-st picking position $(xs_{i+1}, 0)$ on or before $t = t_{i+1}^{pick}$, i.e., the board location arrives there before the robot arm (recall that we, for the moment, ignore potential physical obstructions). In that case, the board stops and waits for the robot arm to arrive. It follows then that $(t_{i+1}^{place}, (xp_{i+1}, yp_{i+1})) = (t_{i+1}^{pick}, (xs_{i+1}, 0))$. Or, the board and its $(i+1)$-st placement location is unable to reach the $(i+1)$-st picking position before $t = t_{i+1}^{pick}$, and given the pick occurring at $t = t_{i+1}^{pick}$, a placement position and time are computed by having the robot arm and board travel directly towards each other. This determines $(t_{i+1}^{place}, (xp_{i+1}, yp_{i+1}))$. To express this in mathematical terms, let f be a function which takes as input two states, each state corresponding to an object, where a state is specified by (time, location, speed). The function f then outputs the time and the location where the two objects meet, provided they travel directly

towards each other. Thus:

$$(t_{i+1}^{place}, (xp_{i+1}, yp_{i+1})) = f((t_{i+1}^{pick}, (xs_{i+1}, 0), V_a), \\ ((t_i^{place}, (xb_{i+1}(t_i^{place}), yb_{i+1}(t_i^{place})), V_b)). \quad (6.1)$$

Next, given $(t_{i+1}^{place}, (xp_{i+1}, yp_{i+1}))$, we compute a minimal t_{i+2}^{pick} as follows. At $t = t_{i+1}^{pick}$, the feeder rack location of component $(i+2)$ starts to move towards the position on the x-axis where the robot arm can reach component $(i+2)$ as quickly as possible after having placed the $(i+1)$-st component. We express this using a function g that takes as input two states, each state corresponding to an object, where a state is again specified by (time, location, speed). The function g then outputs the minimal time and the corresponding location where the two objects meet, given that the second object moves only in the x-direction. Thus:

$$(t_{i+2}^{pick}, (xs_{i+2}, 0)) = g((t_{i+1}^{place}, (xp_{i+1}, yp_{i+1}), V_a), \\ ((t_{i+1}^{pick}, (xf_{i+2}(t_{i+1}^{pick}), yf_{i+2}(t_{i+1}^{pick})), V_f)). \quad (6.2)$$

Equations (6.1) and (6.2) show how the $(i+1)$-st placement position, and the $(i+2)$-nd picking position, as well as their corresponding times can be computed when knowing the i-th placement position, the $(i+1)$-st picking position and the corresponding times. By viewing the starting configuration as the 0-th placement position, and by computing $(xs_1, 0)$ and a minimal t_1^{pick} given the starting configuration, we have specified a GREEDY method. We refer to the example below, and the corresponding figure for an illustration of a GREEDY method.

Notice that we have not specified the precise form of the functions f and g; they depend on the particular distance metric used. We use f to find the meeting place, and meeting time for two objects that each can move in both the x and y-direction, and we use g when one of the two objects can only move horizontally. Thus, in case the board is restricted to move only in the x-direction (see Ayob and Kendall (2005)), this is easily accommodated by replacing f by g in (6.1). Notice further that we

assumed in the description above that there are no physical constraints for any of the moving parts. These constraints, however, are present in existing placement machines (although one could envision a technology where the feeder racks are above the board, and the board could travel beneath the rack without any physical constraints). Then, since the board should not collide with the rack, the board will not be able to reach a picking location. In the case these physical constraints play a role, we let a moving part (e.g., the work table) travel to the location that is *closest* to the location (under the appropriate norm) that was aimed for in case of the absence of these constraints. Observe also that, in case of the Tchebychev norm, there may be multiple locations each of which achieves a minimal time. We come back to this issue in Section 6.4. Finally, notice that GREEDY has the property that at any moment in time the robot arm moves (apart from the time spent in picking and placing the components).

Let us now proceed by sketching an example that shows that a GREEDY method may not always give an optimal solution.

Example

Let us consider the following instance where we first assume a Tchebychev metric. The speed of the robot arm (denoted by V_a) equals 4 (measured in distance-units per time-unit), the speed of the feeder rack (denoted by V_f) equals 1, and the speed of the board (denoted by V_b) equals 0 (i.e., the board is stationary in this example). Let us assume that picking times and placing times can be ignored. Suppose further that at time $t = 0$ the robot arm is positioned at $(0, 0)$ and that it has to place two identical components that are stored in the feeder rack, currently positioned at $(20, 0)$. The first component has to be placed at $(20, 1)$ and the second at $(20, 2)$. The example instance is depicted in Figure 6.1. The starting configuration of the instance is as follows: $t_0^{place} = t_0^{pick} = 0, (xs_0, ys_0) = (xp_0, yp_0) = (0, 0), V_a = 4, V_f = 1$ and $V_b = 0$. Applying GREEDY to this instance yields the following:

$t = 4$: The robot arm meets the first component at $(16, 0)$, and picks it up. Using terminology introduced above, this is computed as follows:
$(t_1^{pick}, (xs_1, 0)) =$
$g[(t_0^{place}, (xp_0, yp_0), V_a), (t_0^{pick}, (xf_1(t_0^{pick}), yf_1(t_0^{pick})), V_f)]$.
Making g explicit we compute xs_1 and t_1^{pick} as follows:
since $yp_0 < |xp_0 - xf_1(t_0^{place})|$
$xs_1 = xf_1(t_0^{place}) - (|xp_0 - xf_1(t_0^{place})|/(V_a + V_f)) \times V_f = 20 - (|0 - 20|/(1 + 4)) \times 1 = 16$
and
$t_1^{pick} = t_0^{pick} + |xf_1(t_0^{pick}) - xs_1|/V_f = 0 + |20 - 16|/1 = 4$.
Thus
$g[(0, (0, 0), 4), (0, (20, 0), 1)] = (4, (16, 0))$.

$t = 5$: The robot arm reaches the first placing location, and places the first component at $(20, 1)$:
$(t_1^{place}, (xp_1, yp_1)) =$
$f[(t_1^{pick}, (xs_1, 0), V_a), (t_0^{place}, (xb_1(t_0^{place}), yb_1(t_0^{place})), V_b)]$.
Making f explicit we compute t_1^{place} and (xp_1, yp_1) as follows:
$t_1^{place} = t_1^{pick} + max(|xs_1 - xp_1|/V_a, yp_1/V_a) = 4 + (|16 - 20|/4, 1/4) = 5$
and, as the board is stationary,
$(xp_1, yp_1) = (xb_1(t_0^{place}), yb_1(t_0^{place})) = (20, 1)$.
Thus
$f[(4, (16, 0), 4), (0, (20, 1), 0)] = (4 + (20 - 16)/4, (20, 1)) = (5, (20, 1))$.

$t = 5.6$: The robot arm meets the second component at $(17.6, 0)$, and picks it up:
$(t_2^{pick}, (xs_2, 0)) =$
$g[(t_1^{place}, (xp_1, yp_1), V_a), (t_1^{pick}, (xf_2(t_1^{pick}), yf_2(t_1^{pick})), V_f)]$
$= g[(5, (20, 1), 4), (4, (16, 0), 1)] = (5.6, (17.6, 0))$.

$t = 6.2$: The robot arm places the second component at $(20, 2)$:
$(t_2^{place}, (xp_2, yp_2)) =$
$f[(t_2^{pick}, (xs_2, 0), V_a), (t_1^{place}, (xb_2(t_1^{place}), yb_2(t_1^{place})), V_b)]$
$= f[(5.6, (17.6, 0), 4), (5, (20, 2), 0)] = (5.6 + (20 - 17.6)/4, (20, 1)) =$

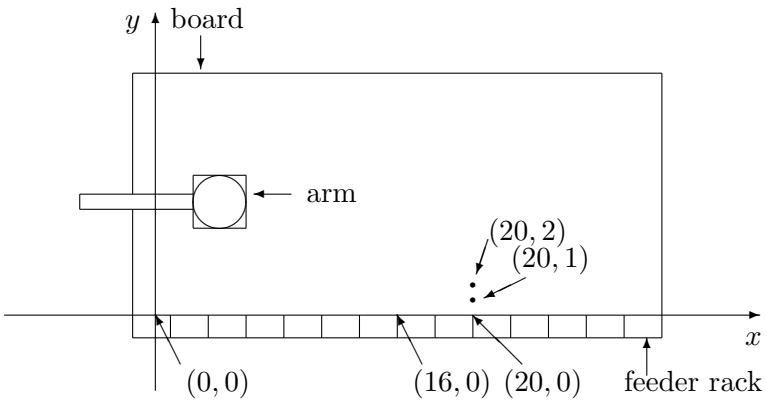


Figure 6.1: *Graphical Representation of the Example*

$(6.2, (20, 2))$.

$t = 11.2$: The robot arm arrives at $(0, 0)$.

Summarizing, robot arm and feeder meet for the first time at $t = 4$ at position $(16, 0)$ where the picking of the first component occurs. The robot arm then travels to the first placing position $(20, 1)$ and places the first component at $t = 5$. The picking of the second component takes place at time $t = 5.6$ at position $(17.6, 0)$ and the placing at $t = 6.2$ at position $(20, 2)$. Finally, the arm needs another 5 time units to return to $(0, 0)$ such that total assembly is finished at $t = 11.2$.

Notice what would happen if we, starting with the initial configuration at $t = 0$, let the feeder rack move only 1 distance unit and wait with the feeder rack in $(19, 0)$ for the robot arm to arrive: then at $t = 4.75$, the robot arm would pick its first component at $(19, 0)$, place this component at time $t = 5$, return to $(19, 0)$ to pick the second component and place it at $t = 5.75$ and finally return to arrive at $(0, 0)$ at time $t = 10.75$, which is faster than GREEDY's solution. Indeed, in this setting it is beneficial to wait with the feeder rack instead of moving it (one also can exhibit examples in which it is beneficial to wait with the robot arm instead of the feeder rack). The idea behind this example is that postponing the picking moment can actually decrease the time from place point to next

place point. This can happen when the robot arm moves faster than the feeder rack. In this case it may be advantageous to travel with the robot arm only, instead of traveling with both of them. More generally, when the speeds of two moving parts differ, GREEDY may not always find an optimal solution. Thus, intuitively, it can be better to use the "fast" moving piece and wait with the "slow" moving piece of equipment instead of moving them both. Under Manhattan metric this example yields similar results. Using GREEDY, the robot arm picks at $t = 4$ the first component at $(16, 0)$, places it at $t = 5.25$, returns to $(18, 0)$ at $t = 6$ to pick the second component and place it at $t = 7$ to finally arrive in $(0, 0)$ at $t = 12.5$. If we would allow the feeder rack to wait at $(20, 0)$, the first pick moment would only occur at $t = 5$ and the first component would be placed at $t = 5.25$ but the second component would be picked at $t = 5.5$, yielding a total assembly time $t = 11.5$. Notice that the example is valid under an Euclidean metric as well, and, in fact, for any other metric $(dx^p + dy^p)^{1/p}$ with $p > 0$.

Obviously, we do not claim that this is a realistic, or a worst-case example; the sole purpose of this example is to illustrate that GREEDY may not yield an optimum solution.

6.3 LP formulation

In this section we show that the motion control problem is solvable in polynomial time by formulating the problem as a linear program under the Tchebychev metric as well as the Manhattan metric. We assume (without loss of generality) that the rack has y-coordinate 0, and that all other y-coordinates are nonnegative; we also assume positive speeds for each of the moving elements. Further, we start from a situation (at $t = 0$) where the robot arm is located at $(0, 0)$, and we impose that the robot arm has to return to $(0, 0)$ after all components have been placed. We now state all variables and parameters we need to describe the model. We use the following variables, for $i \in N = \{1, \ldots, n\}$:

- xs_i: x-coordinate of the pick position of component i,
- xp_i: x-coordinate of the place position of component i,
- yp_i: y-coordinate of the place position of component i,
- T_i^1: time between picking component i and placing it,
- T_i^2: time between placing component i and picking component $i+1$.

(Notice that we let T_n^2 correspond to the time the robot arm needs between placing component n and returning to $(0,0)$.) Finally, let

- T_0: time needed before picking component 1.

We use the following parameters:

- V_a: speed of the robot arm,
- V_b: speed of the worktable,
- V_f: speed of the feeder rack.

Further, for each component i to be placed ($i = 1, \ldots, n$) we have:

- pk_i: time needed to pick component i,
- pc_i: time needed to place component i.

Also, for any pair of locations i and $i+1$ to be visited consecutively ($i = 1, \ldots, n-1$), let

- dx_i: be the difference in x-coordinate,
- dy_i: be the difference in y-coordinate,
- d_i: be the difference (in x-coordinate) between the feeder from which component i is retrieved and the feeder from which component $i+1$ is retrieved.

Finally, let

- d_0: the distance (at $t = 0$) between $(0, 0)$ and the feeder holding the first component,
- (x_1, y_1): the x, y-coordinates of the location of the first component (at $t = 0$), and
- $xs_{n+1} = 0$.

$$\text{Minimize } T_0 + \sum_{i=1}^{n}(T_i^1 + T_i^2) \tag{6.3}$$

$$\text{subject to } T_i^1 \geq \frac{yp_i}{V_a} \qquad \forall i \in N; \tag{6.4}$$

$$T_i^1 \geq \frac{|xs_i - xp_i|}{V_a} \qquad \forall i \in N; \tag{6.5}$$

$$T_i^2 \geq \frac{yp_i}{V_a} \qquad \forall i \in N; \tag{6.6}$$

$$T_i^2 \geq \frac{|xs_{i+1} - xp_i|}{V_a} \qquad \forall i \in N; \tag{6.7}$$

$$pc_i + T_i^1 + T_i^2 \geq \frac{|xs_i + d_i - xs_{i+1}|}{V_f} \qquad \forall i \in N \setminus \{n\}; \tag{6.8}$$

$$pk_{i+1} + T_i^2 + T_{i+1}^1 \geq \frac{|xp_i + dx_i - xp_{i+1}|}{V_b} \qquad \forall i \in N \setminus \{n\}; \tag{6.9}$$

$$pk_{i+1} + T_i^2 + T_{i+1}^1 \geq \frac{|yp_i + dy_i - yp_{i+1}|}{V_b} \qquad \forall i \in N \setminus \{n\}; \tag{6.10}$$

$$T_0 \geq \frac{|d_0 - xs_1|}{V_f} \tag{6.11}$$

$$T_0 \geq \frac{|xs_1|}{V_a} \tag{6.12}$$

$$pk_1 + T_0 + T_1^1 \geq \frac{|x_1 - xp_1|}{V_b} \tag{6.13}$$

$$pk_1 + T_0 + T_1^1 \geq \frac{|y_1 - yp_1|}{V_b} \tag{6.14}$$

$$\text{all variables} \geq 0. \tag{6.15}$$

Notice that since the sum of all picking times and all placing times is a constant, the objective is formulated with (6.3). Constraints (6.4) imply

that the time needed between picking component i and placing it (the left hand side) is at least equal to the time needed to travel with the robot arm in the y-direction to the y-coordinate of the next place position. In a similar fashion, constraints (6.5), (6.6) and (6.7) can be explained. Constraints (6.8) state that the amount of time needed between two consecutive picking operations (the left hand side) must be at least the time needed for the feeder rack to arrive at the position where the next component will be picked (notice that $xs_i + d_i$ reflects the position where component $i + 1$ is at the time when component i is picked). Constraints (6.9) and (6.10) ensure that the board has enough time between two consecutive placement operations to arrive at the next placement operation. Finally, constraints (6.11)-(6.14) deal with the time needed for the first placement.

We make the following remarks:

- Strictly speaking, the model above is not a linear program due to the occurrence of absolute values. However, standard reformulation techniques can resolve this issue.

- Notice that this formulation can easily be modified for a Manhattan metric. Constraints (6.4) and (6.5) can be replaced by a single constraint: $T_i^1 \geq \frac{|xs_i - xp_i| + yp_i}{V_a}$, and a similar operation can be applied to constraints (6.6) and (6.7), constraints (6.9) and (6.10), (6.13) and (6.14).

- This model can easily be modified for the case of a stationary worktable. Indeed, by dropping constraints (6.9) and constraints (6.10) and by turning the xp_i and yp_i from variables into parameters, we obtain a model for the case of a stationary worktable. Also, the model is easily adapted to deal with the case of a worktable being only able to move in the x-direction (see e.g. Ayob and Kendall (2005)).

- Notice that in the description of this model we assume a single feeder rack. However, one easily generalizes this model to a setting where there are two feeder racks alongside the machine (or, even more general, when each component has its own specific travel characteristics,

see Asahiro et al. (2005)).

- In case there are limits for the robot arm, feeder rack, and board on the locations they can reach, one can add linear constraints ensuring these limits.

Finally, there are two important directions in which model (MCP) can be generalized. First, when a point is characterized by d coordinates (instead of two), the model can be easily adapted to deal with this situation. Second, in a setting where the robot arm has a capacity $c \geq 1$, the formulation remains valid. Indeed, it is not unnatural to assume that the robot arm can hold more than a single component (see e.g. Altinkemer et al. (2000)), and as long as the sequence is specified with which these components need to be picked, and need to be placed, the model remains valid.

6.4 Implementation, design, and computational results

6.4.1 Implementation and design

We first describe how we implemented a GREEDY method, and next, we discuss the design of the experiments.

As described in Section 6.2, a GREEDY method for the motion control problem consists in iteratively minimizing time between picking and placing a component and between placing a component and picking the next component. Indeed, suppose that the robot arm and the feeder rack meet each other somewhere on the x-axis to pick a component. From that point on the robot arm moves towards the placing position of that component (assuming a stationary worktable). The feeder starts moving at the same time with the next component to be picked in the direction of the next picking position. After placing, the robot arm returns to the x-axis and robot arm and feeder will meet as soon as possible. Because we are using the Tchebychev metric, this meeting point is not always uniquely deter-

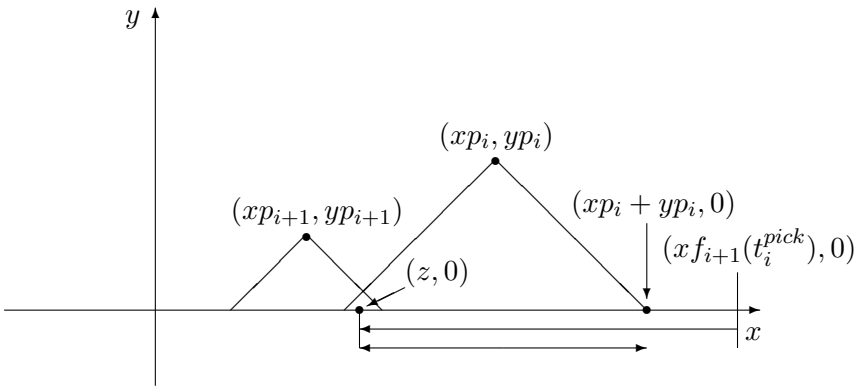


Figure 6.2: *GREEDY for the Tchebychev Metric*

mined, as is shown in Figure 6.2. The robot arm can reach every point in the interval $[(xp_i - yp_i, 0), (xp_i + yp_i, 0)]$ in the same minimal timespan.

Suppose now that at $t = t_i^{pick}$, the feeder location of component $i + 1$ is to the right of $(xp_i + yp_i, 0)$, as indicated in Figure 6.2. Suppose further that the feeder rack can reach up to $(z, 0)$ before the robot arm returns to the x-axis. It follows that every position in the interval $[(z, 0), (xp_i + yp_i, 0)]$ is a meeting point for robot arm and feeder rack achieving minimal time. In our implementation of a GREEDY method under a Tchebychev metric we choose as a meeting point the point which causes a minimal distance for the feeder rack to travel. Thus, we use as a secondary criterion the distance traveled by the feeder rack. In the example depicted in Figure 6.2 this would amount to $(xp_i + yp_i, 0)$ as a meeting point.

Obviously, using knowledge of the next placement points may result in a better solution. Indeed, referring again to Figure 6.2, given placement position (xp_{i+1}, yp_{i+1}), $(z, 0)$ may be a better meeting point than $(xp_i + yp_i, 0)$ when it comes to minimizing total time needed. However, we decided in our implementation of a GREEDY method not to use any information from upcoming placement positions, and instead use as a secondary criterion the feeder distance traveled.

A GREEDY algorithm for the motion control problem is much more straightforward in the case of a Manhattan metric. Robot arm and feeder

meet in a unique point, i.e., starting from a pick position the robot arm moves towards the placement position, returns to the x-axis and moves in the direction of the feeder; in the meantime the feeder moves with the next component in the direction of the robot arm until they meet.

The setting of our experiment is as follows: consider a board of length 1000 and width 500 with n randomly generated placing positions on this board. We generated for m different component types a position on a feeder rack of length 3000. For each of the n components we uniformly selected a component type. The robot arm can move in the x- and y-direction, the feeder rack can move in the x-direction only, and the board is stationary. To completely specify an instance of the motion control problem, we took a random sequence of locations as the solution to the component sequencing problem, and we took a random assignment of feeders to positions in the rack as a feeder rack assignment. In addition, we assumed that there is precisely one feeder for each type of components, and hence, the component retrieval problem vanishes. As pointed out by a referee, the fact that these solutions are not found by some heuristic, may adversely affect the results of a GREEDY method, when compared to an optimum solution to the motion control problem.

Different experiments were executed by changing (i) the number of components to be placed ($n = 40, 80, 160$), (ii) the number of component types ($m = 10, 20$), (iii) the relative speeds of feeder and robot arm ($V_f/V_a = 0.25, 0.5, 1, 2, 4$), and (iv) the capacity c of the head of the robot arm ($c = 1, 4$); see Table 6.1 for an overview. In our choice for some of these parameter values, we used van Gastel et al. (2004).

We implemented a GREEDY strategy in C^{++} language and we solved the LP's using ILOG CPLEX 8.1.0, OPL Studio 3.6.1. The tests were performed on a personal computer with a 2.8 GHz Intel(R)Pentium(R) IV with 504 MB of RAM. Since all computation times are within two seconds, we have not reported them.

6.4.2 Results

The results for the Tchebychev metric are summarized in Tables 6.2 and 6.3. Each number is the average over the results of 10 different randomly generated instances.

Table 6.2 gives the percentage deviation[1] of the GREEDY heuristic from an optimal solution for a machine with a robot arm carrying at most one component. It is clear that for a slow moving feeder rack (slow compared to the robot arm), the deviation of GREEDY's solutions from an optimal solution is relatively small. This is to be expected: in an extreme case of a stationary feeder rack, a solution found by a GREEDY method and an optimal solution coincide. However, GREEDY's performance deteriorates when the ratio V_f/V_a increases. Indeed, the slower the robot arm is (compared to the feeder rack) the larger the interval becomes where all meeting points have a minimal time between placing component i and picking component $i+1$. Also, the effect of the density (n/m) on GREEDY's performance seems relatively small.

In Table 6.3 the percentage deviation of GREEDY's solutions from optimal solutions is given when the head of the robot arm can carry at

[1] $(=100 \times \frac{\text{assembly time GREEDY} - \text{optimal assembly time}}{\text{optimal assembly time}})$

number of components	40/80/160
number of component types	10/20
length of the board	1000 (in distance units)
width of the board	500 (in distance units)
length of the feeder rack	3000 (in distance units)
speed feeder rack/speed robot arm	0.25/0.5/1/2/4
time needed to pick a component	0.8 (in time units)
time needed to place a component	0.8 (in time units)
capacity of head	1/4

Table 6.1: *Experimental design*

n	m	V_f/V_a				
		0.25	0.5	1	2	4
40	10	5.599	5.843	7.971	16.218	20.398
80	10	6.512	6.675	8.108	14.674	19.965
160	10	6.769	6.735	8.194	16.895	21.690
40	20	7.436	7.114	8.544	16.941	20.552
80	20	6.984	6.825	8.149	16.602	20.457
160	20	7.046	7.240	8.544	16.018	19.636

Table 6.2: *Percentage deviation of GREEDY from the optimum under a Tchebychev metric, head carries 1 component*

n	m	V_f/V_a				
		0.25	0.5	1	2	4
40	10	1.470	1.791	2.469	5.624	7.668
80	10	1.844	2.069	2.600	5.862	7.566
160	10	1.861	1.990	2.616	6.352	7.988
40	20	1.476	1.754	2.100	5.621	7.501
80	20	2.021	2.281	2.563	6.549	8.182
160	20	1.773	1.983	2.375	6.132	7.470

Table 6.3: *Percentage deviation of GREEDY from the optimum under a Tchebychev metric, head carries 4 components*

most four components, meaning that it can pick up four components before it travels to the board for placing. These results follow the same trend as the results in Table 6.2, namely a small deviation for a fast moving arm (compared to the feeder) which becomes larger as V_f/V_a increases. But, the percentage deviations in Table 6.3 are smaller than in the previous table. This can be explained by the fact that both a GREEDY solution and an optimal solution follow the same movement pattern during the time that the arm needs to pick up four components; only when the arm travels to the board to place the four components and then returns to the x-axis to pick the next component, differences may occur. And since the number of times the robot arm has to return to the feeder rack is now much smaller, the deviation of GREEDY's solutions compared to optimal solutions will be smaller.

n	m	V_f/V_a				
		0.25	0.5	1	2	4
40	10	3.669	1.319	0.000	3.426	2.084
80	10	3.900	1.776	0.000	3.396	1.791
160	10	4.183	2.098	0.000	3.515	1.580
40	20	4.322	2.507	0.000	2.982	0.809
80	20	4.206	2.273	0.000	3.205	0.923
160	20	4.085	2.183	0.000	3.062	1.060

Table 6.4: *Percentage deviation of GREEDY from the optimum under a Manhattan metric, head carries 1 component*

Similar experiments have been carried out for the Manhattan metric, and the results are summarized in Tables 6.4 and 6.5. We first consider the case where the robot arm carries at most one component (Table 6.4). Here, the picture is somewhat different compared to the Tchebychev metric: although it is still true that for small ratio's of V_f/V_a, GREEDY's deviations from the optimum are relatively small, GREEDY's results improve up to $V_f/V_a = 1$. Indeed, when $V_f = V_a$ the values of the two solutions are

identical; we claim the following.

Claim 2. *When $V_a = V_f$, $V_b = 0$, and when using a Manhattan metric, GREEDY gives an optimal solution to the motion control problem.*

Argument: We first argue that - in case $V_a = V_f, V_b = 0$ and under a Manhattan metric - there exists an optimal solution to the motion control problem that satisfies the following two properties:

Property 1: between two consecutive placements, the robot arm travels, directly after placing component i, from (xp_i, yp_i) to $(xp_i, 0)$, travels towards the appropriate location on the feeder rack until it meets this location at $(xs_{i+1}, 0)$, where it picks up component $i+1$ (notice that this location may coincide with $(xp_i, 0)$), and finally the robot arm travels via $(xp_{i+1}, 0)$ to (xp_{i+1}, yp_{i+1}) where it places component $i+1$ $(1 \leq i \leq n)$, and

Property 2: the feeder rack, after the picking of component i, moves in a way that the location of component $i+1$ travels to $(xp_i, 0)$, and the feeder rack only stops when this location reaches this position, or when it meets the robot arm $(1 \leq i \leq n)$.

To see that property 1 is valid, consider an optimal solution in which

n	m	V_f/V_a				
		0.25	0.5	1	2	4
40	10	0.300	0.089	0.000	6.517	9.690
80	10	0.763	0.209	0.000	6.992	10.099
160	10	0.775	0.201	0.000	7.212	10.218
40	20	0.472	2.066	0.000	6.979	10.282
80	20	0.871	0.142	0.000	7.577	10.830
160	20	0.870	0.101	0.000	7.315	10.323

Table 6.5: *Percentage deviation of GREEDY from the optimum under a Manhattan metric, head carries 4 components*

the robot arm deviates from the sketched procedure. More specifically, consider an optimal solution in which the robot arm after having placed component i, and after having traveled to $(xp_i, 0)$, does not travel towards the appropriate location on the feeder rack. Then an alternative optimal solution can be constructed with identical picking moments and placing moments that involves waiting time for the robot arm. We further modify this alternative optimal solution by 'collecting' the waiting time, and 'shift' it to the first picking moment that follows, say at $t = t_i^{pick}$. Thus, in this optimal solution the robot arm waits until the feeder rack arrives, and the pick occurs at $t = t_i^{pick}$. It is clear that, instead of the robot arm having to wait until the feeder rack arrives, we can also let the robot arm travel towards the feeder rack, meet the feeder rack halfway, and, after having picked this component, return, together with the feeder rack to be back at $t = t_i^{pick}$ at the original picking location. Thus, this alternative solution has at $t = t_i^{pick}$ the robot arm and the feeder rack at exactly the same location as in the original optimal solution, and, the remaining part of the solution is identical to the original optimal solution. Thus, any optimal solution not satisfying Property 1, can be modified into one that does.

Property 2 is valid for a similar reason. Suppose, after the picking of component i, the location of component $i + 1$ in the feeder rack waits at a position different from $(xp_i, 0)$ until the robot arm arrives. Again, as above, an alternative optimal solution would be to travel with the feeder rack meeting the robot arm, and return to the original position at the same time.

Properties 1 and 2 together precisely yield GREEDY. □

Notice also that for further increasing values of the ratio V_f/V_a GREEDY's results first deteriorate, however then improve again for a faster moving feeder rack. Finally, the effect of the density (n/m) on GREEDY's performance seems again relatively small.

When the head can carry up to four components, the results found by GREEDY improve (compared to the setting where the head carries a single component) as long as $V_f/V_a \leq 1$ (see Table 6.5). When $V_f > V_a$,

GREEDY's results are worse than in the setting with a single component. This is explained by the observation that the robot arm is forced - when picking up four components - to spend time traveling along the x-axis. This may be done in a suboptimal way leading to the results in Table 6.5. This contrasts with the case of the head carrying at most one component: in that case, the rack is waiting for the robot arm when it needs to pick up a component, and hence the robot arm spends its time exclusively placing the components.

Notice that there is a difference here compared to the Tchebychev metric, where results of GREEDY improve overall when going from a capacity of the arm of 1 to a capacity of 4. The effect as described above for GREEDY also holds for a Tchebychev metric but is overruled by another effect. Remember that in the implementation of GREEDY under a Tchebychev metric we used as a secondary criterion the distance traveled by the feeder rack. As a consequence, a fast feeder (compared to the arm) might spend a lot of time waiting in a suboptimal position every time the robot arm is placing components on the board. As this happens more often when the arm can carry only one component, GREEDY will perform worse than in the case that the robot arm has a capacity of 4.

6.5 Conclusion

We investigated the problem of how to determine movement patterns for the moving parts of an automated placement machine. We showed that a straightforward greedy strategy to establish these patterns may not give an optimal solution. However, at least under a Tchebychev metric and under a Manhattan metric the problem is solvable in polynomial time by formulating it as a linear program. Both for a Tchebychev metric and a Manhattan metric, we showed that a reduction in assembly times is possible by using the LP-model.

List of Figures

List of Tables

Bibliography

Afrati, F., Cosmadakis, S., Papadimitriou, C. H., Papageorgiou, G., Papakostantinou, N., 1986. The complexity of the traveling repairman problem. Informatique Théorique et Applications 20, 79–87.

Aksen, D., Aras, N., 2005. Customer selection and profit maximization in vehicle routing problems. Operations Research Proceedings 2005, Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), 37–42.

Alegre, J., Laguna, M., Pacheco, J., 2007. Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. European Journal of Operational Research 179, 736–746.

Alonso, F., Alvarez, M. J., Beasley, J. E., 2008. A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. Journal of the Operational Research Society 59, 963–976.

Altinkemer, K., Kazaz, B., Köksalan, M., Moskowitz, H., 2000. Optimization of printed circuit board manufacturing: Integrated modeling and algorithms. European Journal of Operational Research 124, 409–421.

Angelelli, E., Speranza, M. G., 2002. The periodic vehicle routing problem with intermediate facilities. European Journal of Operational Research 137, 233–247.

Angelelli, E., Speranza, M. G., Tuza, Z., 2008. Notes on the TSP with profit, manuscript.

Anily, S., Glass, C. A., Hassin, R., 1998. The scheduling of maintenance service. Discrete Applied Mathematics 82, 27–42.

Archetti, C., Feillet, D., Hertz, A., Speranza, M. G., 2008. The capacitated team orienteering and profitable tour problems. Journal of the Operational Research Society, 1–12.

Arora, S., Karakostas, G., 2003. Approximation schemes for minimum latency problems. SIAM Journal on Computing 32, 1317–1337.

Artois, P., 2004. Routeplanning bij het Koninklijk Instituut Woluwe, masters thesis, KULeuven (in Dutch).

Asahiro, Y., Miyano, E., Shimoirisa, S., 2005. Pickup and delivery for moving objects on broken lines. Lecture Notes in Computer Science 3701, 36–50.

Averbakh, I., Berman, O., 1994. Routing and location-routing p-deliverymen problems on a path. Transportation Science 28, 162–166.

Averbakh, I., Berman, O., 1996. A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. Discrete Applied Mathematics 68, 17–32.

Ayob, M., Kendall, G., 2005. A triple objective function with a chebychev dynamic pick-and-place point specification approach to optimise the surface mount placement machine. European Journal of Operational Research 164, 609–626.

Balas, E., 1989. The prize collecting traveling salesman problem. Networks 19, 621–636.

Ball, M., Magazine, M., 1988. Sequencing of insertions in printed circuit board assembly. Operations Research 36, 192–201.

Baptiste, S., Oliviera, R. C., Zúquete, E., 2002. A period vehicle routing case study. European Journal of Operational Research 139, 220–229.

Beltrami, E. J., Bodin, L. D., 1974. Networks and vehicle routing for municipal waste collection. Networks 4, 65–94.

Bérubé, J., Gendreau, M., Potvin, J., 2009. An exact ε-constraint method for bi-objective combinatorial optimization problems - application to the traveling salesman problem with profits. European Journal of Operational Research 194, 39–50.

Blakely, F., Bozkaya, B., Cao, B., Hall, W., Knolmajer, J., 2003. Optimizing periodic maintenance operations for Schindler Elevator Corporation. Interfaces 33, 67–79.

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., Sudan, M., 1994. The minimum latency problem. Proceedings of the twenty-sixth annual ACM symposium on the theory of computing (STOC), 163–171.

Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for team orienteering problems. 4OR 5, 211–230.

Butt, S. E., Cavalier, T. M., 1994. A heuristic for the multiple tour maximum collection problem. Computers and Operations Research 21, 101–111.

Campbell, A. M., Hardin, J. R., 2005. Vehicle minimization for periodic deliveries. European Journal of Operational Research 165, 668–684.

Chandran, B., Raghavan, S., 2008. Modeling and solving the capacitated vehicle routing problem on trees. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), The Vehicle Routing Problem. Springer, pp. 239–261.

Chao, I. M., Golden, B., Wasil, E. A., 1996. The team orienteering problem. European Journal of Operational Research 88, 464–474.

Christofides, N., Beasley, J. E., 1984. The period routing problem. Networks 14, 237–256.

Claassen, G. D. H., Hendriks, T. H. B., 2007. An application of special ordered sets to a periodic milk collection problem. European Journal of Operational Research 180, 754–769.

Coene, S., Arnout, A., Spieksma, F. C. R., 2008a. The periodic vehicle routing problem: a case study. FEB Research Report 0828.

Coene, S., Filippi, C., Spieksma, F. C. R., Stevanato, E., 2008b. The traveling salesman problem on trees: balancing profits and costs. Submitted.

Coene, S., Spieksma, F. C. R., 2008. Profit-based latency problems on the line. Operations Research Letters 36, 333–337.

Coene, S., Spieksma, F. C. R., Woeginger, G. J., 2009. Charlemagne's challenge: the periodic latency problem, manuscript.

Coene, S., van Hop, N., van de Klundert, J., Spieksma, F. C. R., 2008c. A note on a motion control problem for a placement machine. OR Spektrum 30, 535–549.

Coja-Oghlan, A., Krumke, S. O., Nierhoff, T., 2006. A heuristic for the stacker crane problem on trees which is almost surely exact. Journal of Algorithms 61, 1–19.

Crama, Y., Flippo, O. E., van de Klundert, J. J., Spieksma, F. C. R., 1996. The component retrieval problem in printed circuit board assembly. International Journal of Flexible Manufacturing Systems 8, 287–312.

Crama, Y., Kats, V., van de Klundert, J., Levner, E., 2000. Cyclic scheduling in robotic flowshops. Annals of Operations Research 96, 97–124.

Crama, Y., van de Klundert, J., 1997. Cyclic scheduling of identical parts in a robotic cell. Operations Research 45, 952–965.

Crama, Y., van de Klundert, J. J., Spieksma, F. C. R., 2002. Production planning problems in printed circuit board assembly. Discrete Applied Mathematics 123, 339–361.

de Hoon, M., Imoto, S., Miyano, S., 2008. The C Clustering Library.

de Paepe, W. E., Lenstra, J. K., Sgall, J., Sitters, R. A., Stougie, L., 2004. Computer-aided complexity classification of dial-a-ride problems. INFORMS Journal on Computing 16, 120–132.

Dekker, R., van der Duyn Schouten, F. A., Wildeman, R. E., 1997. A review of multi-component maintenance models with economic dependence. Mathematical Methods of Operations Research 45, 411435.

Dell'Amico, M., Maffioli, F., Värbrand, P., 1995. On prize-collecting tours and the asymmetric traveling salesman problem. International Transactions in Operational Research 2, 297–308.

Deĭneko, V. G., Rudolf, R., van der Veen, J. A. A., Woeginger, G. J., 1995. Three easy special cases of the Euclidean traveling salesman problem. SFB Report 17, Institut für Mathematik, TU Graz, Austria.

Dimov, V., Kumar, A., Hebbar, R., Fares, W., Sharma, P., 2007. Mini-rounds improve physician-patient communication and satisfaction (abstract). Journal of Hospital Medicine, SHM Annual Meeting 2, 48.

Egbelu, P. J., Wu, C., Pilgaonkar, R., 1996. Robotic assembly of printed circuit boards with component feeder location consideration. Production Planning and Control 7, 162–175.

Eglese, R. W., Murdock, H., 1991. Routing road sweepers in a rural area. The Journal of the Operational Research Society 42, 169–194.

Ehrgott, M., 2000. Approximation algorithms for combinatorial multicriteria optimization problems. International Transactions in Operational Research 7, 5–31.

Ehrgott, M., 2005. Multicriteria Optimization. Springer.

Erlebach, T., Kellerer, H., Pferschy, U., 2002. Approximating multiobjective knapsack problems. Management Science 48, 1603–1612.

Fackaroenphol, J., Harrelson, C., Rao, S., 2007. The k-traveling repairman problem. ACM Transactions on Algorithms 3, Article No. 40.

Feillet, D., Dejax, P., Gendreau, M., 2005. Traveling salesman problems with profits. Transportation Science 39, 188–205.

Francis, P., Smilowitz, K., Tzur, M., 2006. The period vehicle routing problem with service choice. Transportation Science 40, 439–454.

Francis, P., Smilowitz, K., Tzur, M., 2008. The period vehicle routing problem and its extensions. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), The Vehicle Routing Problem. Springer, 239-261.

Frederickson, G. N., Wittman, B., 2007. Approximation algorithms for the traveling repairman and speeding deliveryman problems with unit-time windows. APPROX and RANDOM 2007, Lecture Notes in Computer Science (LNCS) 4627, 119–133.

Friese, P., Rambau, J., 2006. Online-optimization of multi-elevator transport systems with reoptimization algorithms based on set-partitioning models. Discrete Applied Mathematics 154, 1908–1931.

García, A., Jodrá, P., Tejel, J., 2002. A note on the traveling repairman problem. Networks 40, 27–31.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco.

Goemans, M., Kleinberg, J., 1998. An improved approximation ratio for the minimum latency problem. Mathematical Programming 82, 111–124.

Golden, B. L., Levy, L., Vohra, R., 1987. The orienteering problem. Naval Research Logistics 34, 307–318.

Golden, B. L., Raghavan, S., Wasil, E. A. (Eds.), 2008. The Vehicle Routing Problem: Latest Advances and New Challenges. Springer.

Grunow, M., Günther, H. O., Schleusener, M., Yilmaz, I. O., 2004. Operations planning for collect-and-place machines in pcb assembly. Computers and Industrial Engineering 47, 409–429.

Helvig, C. S., Robins, G., Zelikovsky, A., 2003. The moving-target traveling salesman problem. Journal of Algorithms 49, 153–174.

Hemmelmayr, V., Doerner, K. F., Hartl, R. F., Savelsbergh, M. W. P., 2008. Delivery strategies for blood products supplies. OR Spectrum available online.

Hoogeveen, J. A., 1992. Single-machine bicriteria scheduling, PhD Thesis, Eindhoven University of Technology.

Hoogeveen, J. A., 2005. Multicriteria scheduling. European Journal of Operational Research 167, 592–623.

ILOG, 2005. ILOG Dispatcher 4.1, User's manual.

Johnson, D. S., Niemi, K. A., 1983. On knapsacks, partitions, and a new dynamic programming technique for trees. Mathematics of Operations Research 8, 1–14.

Jothi, R., Raghavachari, B., 2007. Approximating the k-traveling repairman problem with repairtimes. Journal of Discrete Algorithms 5, 293–303.

Jozefowiez, N., Glover, F., Laguna, M., 2008a. Multi-objective metaheuristics for the traveling salesman problem with profits. Journal of Mathematical Modeling and Algorithms 7, 177–195.

Jozefowiez, N., Semet, F., Talbi, E., 2008b. Multi-objective vehicle routing problems. European Journal of Operational Research 189, 293–309.

Kalmanson, K., 1975. Edgeconvex circuits and the traveling salesman problem. Canadian Journal of Mathematics 27, 1000–1010.

Karuno, Y., Nagamochi, H., Ibaraki, T., 1997. Vehicle scheduling on a tree with release and handling times. Annals of Operations Research 69, 193–207.

Keller, C. P., Goodchild, M., 1988. The multiobjective vending problem: A generalization of the traveling salesman problem. Environmental Planning B: Planning Design 15, 447–460.

Kim, B., Kim, S., Sahoo, S., 2006. Waste collection vehicle routing problem with time windows. Computers & Operations Research 33, 3624–3642.

Klinz, B., Woeginger, G. J., 1999. The steiner tree problem in kalmanson matrices and in circulant matrices. Journal of Combinatorial Optimization 3, 51–58.

Korst, J., Aarts, E., Lenstra, J. K., 1997. Scheduling periodic tasks with slack. INFORMS Journal on Computing 9, 351–362.

Labbé, M., Laporte, G., Mercure, H., 1991. Capacitated vehicle routing on trees. Operations Research 39, 616–622.

Latombe, J. C., 1991. Robot Motion Planning. Kluwer, Boston.

Leipälä, T., Nevalainen, O., 1989. Optimization of the movements of a component placement machine. European Journal of Operational Research 38, 167–177.

Lim, A., Wang, F., Xu, Z., 2005. The capacitated traveling salesman problem with pickups and deliveries on a tree. Proceedings of ISAAC05, Lecture Notes in Computer Science, 1061–1070.

McMullan, R., Gilmore, A., 2008. Customer loyalty: an empirical study. European Journal of Marketing 42, 1084–1094.

Menger, K., 1932. Das botenproblem. Ergebnisse Eines Mathematischen Kolloquiums 2, 11–12.

Minieka, E., 1989. The delivery man problem on a tree network. Annals of Operations Research 18, 261–266.

Mourgaya, M., Vanderbeck, F., 2006. Problème de tournées de véhicules multipériodiques: Classification et heuristique pour la planification tactique. RAIRO Operations Research 40, 169–194.

Muslea, I., 1997. The very offline k-vehicle routing problem on trees. Proceedings of the International Conference of the Chilean Computer Science Society, 155–163.

Ortec, 2007. Shortrec. http://www.ortec.be.

Paletta, G., 2002. The period traveling salesman problem: a new heuristic algorithm. Computers and Operations Research 29 (10), 1343–1352.

Papadimitriou, C. H., Yannakakis, M., 2000. On the approximability of trade-offs and optimal access of web sources. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 86-92.

Picard, J., Queyranne, M., 1978. The time-dependent travelling salesman problem and its application to the tardiness problem in one-machine scheduling. Operations Research 26, 86–110.

Psaraftis, H. N., Solomon, M. M., Magnanti, T. L., Kim, T., 1990. Routing and scheduling on a shoreline with release times. Management Science 36, 212–223.

Salehipour, A., Sörensen, K., Goos, P., Bräysy, O., 2008. An efficient GRASP+VND metaheuristic for the traveling repairman problem. UA Research report 2008008.

Samphaiboon, N., Yamada, T., 2000. Heuristic and exact algorithms for the precedence-constrained knapsack problem. Journal of Optimization Theory and Applications 105, 659–676.

Sitters, R., 2002. The minimum latency problem is NP-hard for weighted trees. Proceedings of the ninth International Conference on Integer Programming and Combinatorial Optimization (IPCO) LNCS 2337, 230–239.

Sitters, R., 2004. Complexity and Approximation in Routing and Scheduling, PhD thesis. Technical University Eindhoven.

Studer, Q., 2008. Results that last: hardwiring behaviors that will take your company to the top. John Wiley & Sons, Inc., Hoboken, New Jersey.

Su, C., Fu, H., 1998. A simulated annealing heuristic for robotics assembly using the dynamic pick-and-place model. Production Planning and Control 9, 795–802.

Su, C., Ho, L., Fu, H., 1998. A novel tabu search approach to find the best placement sequence and magazine assignment in dynamic robotics assembly. Integrated Manufacturing Systems 9, 366–376.

Su, Y., Wang, C., Egbelu, P., Cannon, D. J., 1995. A dynamic point specification approach to sequencing robot moves for pcb assembly. International Journal on Computer Integrated Manufacturing 8, 448–456.

Tan, C. C. R., Beasley, J. E., 1984. A heuristic algorithm for the period vehicle routing problem. Omega 12, 497–504.

T'kindt, V., Bouibede-Hocine, K., Esswein, C., 2005. Counting and enumeration complexity with application to multicriteria scheduling. 4OR 3, 1–21.

Toth, P., Vigo, D. (Eds.), 2002. The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia.

Tricoire, F., Romauch, M., Doerner, K. F., Hartl, R. F., 2008. Algorithms for the multi-period orienteering problem with multiple time windows. EU/MEeting 2008: Workshop on Metaheuristics for Logistics and Vehicle Routing, paper 23.

Tsitsiklis, J. N., 1992. Special cases of traveling salesman and repairman problems with time windows. Networks 22, 263–282.

van Gastel, S., Nikeschina, M., Petit, R., 2004. Fundamentals of SMD assembly. Assembléon.

van Hop, N., Tabucanon, M. T., 2001a. Extended dynamic point specification approach to sequencing robot moves for pcb assembly. International Journal of Production Research 39, 1671–1687.

van Hop, N., Tabucanon, M. T., 2001b. Multiple criteria approach for solving feeder assignment and assembly sequence problem in pcb assembly. Production Planning and Control 12, 735–744.

Verhaegh, W. F. J., Aarts, E. H. L., van Gorp, P. C. N., Lippens, P. E. R., 2001. A two-stage solution approach to multidimensional periodic scheduling. IEEE Transactions on computer-aided design of integrated circuits and systems 20, 1185–1199.

Wang, C., Ho, L., Cannon, D. J., 1998. Heuristics for assembly sequencing and relative magazine assignment for robotic assembly. Computers and Industrial Engineering 34, 422–431.

Woeginger, G. J., 1999. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? INFORMS Journal on Computing 12, 57–74.

Wu, B. Y., 2000. Polynomial time algorithms for some minimum latency problems. Information Processing Letters 75, 225–229.

Wu, B. Y., Huang, Z., Zhan, F., 2004. Exact algorithms for the minimum latency problem. Information Processing Letters 92, 303–309.

Yu, W., Hoogeveen, H., Lenstra, J. K., 2004. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. Journal of Scheduling 7, 333–348.

Doctoral dissertations from the Faculty of Business and Economics

A list of doctoral dissertations from the Faculty of Business and Economics can be found at the following website:
`http://www.econ.kuleuven.be/phd/doclijst.htm`.