Approximation Algorithms for Group Coverage and Vehicle Routing Problems

by

Christopher Stuart Martin

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In this thesis, we present approximation algorithms for various NP-hard vehicle routing problems, as well as for a related maximum group coverage problem. Our main contribution is a framework to build good constant-factor approximation algorithms for variants of the *multi-depot k-travelling repairmen problem* (sometimes referred to as the *multi-depot minimum latency problem*), which has been studied previously [12, 28]. We use our framework to create approximations for several capacitated variants of the problem, for which only heuristic solutions exist currently. Our framework can also be used to approximate the (uncapacitated) multi-depot k-travelling repairmen problem; interestingly, the approximation ratio we obtain matches the current-best result for the problem.

To achieve these results, we develop a Linear Programming (LP) based approximation algorithm to the *maximum coverage with groups problem* (or MCG), which is used as a subroutine in our framework. This problem has not received much attention in the literature, as it is mostly used as a subroutine for bigger problems. We use the linear programming relaxation of an integer program, which we first solve using a suitable (approximate) oracle, and then show how to deterministically round the solution to an integer solution without losing too much over the optimal. This also shows the integrality gap of the LP we consider.

We finally consider the *orienteering problem with time-windows*, and a special case called the *deadline orienteering problem*. Both of these problems have been well-studied, and currently no constant approximation is known for either problem; finding a constant approximation for either one is a big open problem. We present a constant-approximation for a fairly general special case of the deadline orienteering problem that runs in sub-exponential time (which can also be extended to the time-windows version); this gives a strong indication that constant approximations for these problems exist. We also give approximations for various problems related to the deadline orienteering problem, in the hope that they might give more insight into the problem.

To my parents.

Acknowledgements

There are many people I want to thank for both supporting me through my degree as well as for their help with this thesis. First and foremost, I want to thank my supervisor, Mohammad R. Salavatipour. You have been a great teacher, mentor, and role-model who has pushed me to become a better student and researcher. While I have had my fair share of failures and stumbles along the way, you have shown me how much more I can achieve, and how much further I can still grow as a researcher.

I also want to thank Zachary Friggstad for not only providing insightful and constructive comments on this thesis, but for being a great colleague. You have been an inspiration to me, and I have valued your guidance, especially during the more difficult times over the past two years.

I especially want to thank Lorna Stewart for her very insightful questions and comments on this thesis. Your feedback has not only given me avenues to improve on as a researcher, but has also significantly strengthened the results of this thesis.

My involvement with the ACM International Collegiate Programming Contest in my undergrad was a very formative experience for me, and I want to thank my former coach Howard Cheng and teammates Darcy Best, Hugh Ramp, and Farshad Barahimi for making it such an amazing experience. I especially want to thank Howard Cheng and Darcy Best for inspiring me to pursue graduate studies and explore my full potential.

This research would not have been possible without the support of NSERC, Alberta Innovates Technology Futures, and the University of Alberta Department of Computing Science. I am also incredibly grateful to my friends and family for supporting me throughout my graduate degree. You have been such a blessing to me, it is beyond words.

Table of Contents

List of Figures

Chapter 1

Introduction

Many problems that arise in industrial and consumer applications can be viewed as the problem of routing a vehicle (or vehicles) along a road network to provide a service at different locations. Depending on the application, the objective and constraints can vary, so we collectively refer to these problems as *vehicle routing problems*. Some well-known vehicle routing problems include:

- The travelling salesman problem (TSP): find the shortest route that reaches every location,
- The orienteering problem: find a route of bounded length that reaches as many locations as possible,
- The travelling repairman problem (TRP): find a route that reaches every location, minimizing the average wait time,
- The school bus problem: find the smallest number of routes so that every location is visited within a given time.

For example, the travelling salesman problem appears in manufacturing applications, where we wish to minimize the amount of time a tool spends moving between its required positions. The school bus problem, as the name suggests, commonly appears when creating public transportation routes. The orienteering problem and travelling repairman problems commonly appear in scheduling package deliveries.

All of the problems given above are **NP**-hard in general: it is widely believed that there is no time-efficient[1] way for a computer to solve these problems exactly for an arbitrarily large input. In this thesis, we consider efficient algorithms that produce an *approximate* solution; that is, the returned solution is at most α times worse than the exact (optimal) value. For example, there is an efficient algorithm for the standard travelling salesman

[1] We call an algorithm "time-efficient" if it runs in *polynomial time*, or $O(n^c)$ for some fixed constant $c > 0$ and n being the size of the input instance.

problem that returns a route of length at most 1.5 times the length of the optimal route [13].

We investigate some interesting vehicle routing problems that are **NP**-hard, and give approximation algorithms for them. We also look at a maximum coverage style problem, which is used as a subroutine in our algorithms. We first begin with an introduction to the terminology and concepts used in this thesis, and then given an overview of the problems we consider. We will then discuss prior work in the literature, and the results we obtain.

1.1 Preliminaries

We begin by formalizing the terminology we will use throughout this thesis. The definitions given here are adapted from [34], [33], [35], and [14].

1.1.1 Graphs and Metrics

Graphs. We only consider simple graphs in this thesis, and use the term *graph* in this thesis to mean an undirected graph. Such a graph G is defined by its edge set $E(G) = \{e_1, e_2, \ldots, e_m\}$ and vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$, where each edge $e \in E(G)$ is an unordered pair of vertices in $V(G)$. To simplify notation, we may drop the parameters of V and E when the graph is clear from context, and instead denote G as the pair (V, E). We also consider *directed* graphs; in a directed graph G, each edge $e \in E(G)$ is an *ordered* pair of vertices. We use the same notation as for undirected graphs.

For each edge $e = uv \in E(G)$, we say u and v are *adjacent* and e is *incident* to u and v. The *neighbours* of a vertex v is the set of vertices u such that u and v are adjacent; we denote this set as $N_G(v)$, or simply $N(v)$ when G is clear from context.

A *subgraph* of a graph G is a graph H, where H is obtained from G by deleting some edges and/or some vertices (and their incident edges) from G. We notate this relation as $H \subseteq G$, and may simply say that G *contains* H or H is *in* G. A subgraph $H \subseteq G$ is *spanning* if $V(H) = V(G)$.

A *path* is a graph whose vertices can be ordered such that u and v appear consecutively in the ordering if and only if the edge uv is in the graph. A *cycle* is a graph with an equal number of edges and vertices, where the vertices can be placed around a circle and u and v appear consecutively if and only if the edge uv is in the graph. A *walk* in a graph G is a sequence of (not necessarily distinct) vertices $v_0, v_1, \ldots, v_k$ such that for all $1 \leq i \leq k$, $v_{i-1}v_i \in E(G)$.

A *complete graph* is a graph whose vertices are pairwise-adjacent. An *acyclic graph* is a graph that does not contain any cycle. A *connected graph* is a graph G where for every pair of vertices $u, v \in V(G)$, there is a path in G from u to v. A *tree* is an acyclic, connected

graph.

An *independant set* in a graph is a collection of vertices that are pairwise non-adjacent. A graph G is *bipartite* if its vertex set can be partitioned into two independent sets. Other interesting graph types that appear in this thesis include *spanning trees* (a subgraph of some graph G that is both a tree and spanning in G) and *directed acyclic graphs* (DAGs).

Weighted graphs and metrics. A *weighted graph* is a graph with numerical edge labels. We will assume throughout this thesis that these labels are non-negative, and refer to them as edge costs, denoted c_e or c_{uv}.

Given edge costs, we define the *cost* of a graph G as $\sum_{e \in E(G)} c_e$. A *minimum spanning tree* (MST) is a spanning tree of a weighted graph that has minimum cost. A *k-minimum spanning tree* (k-MST) of G is the cheapest MST over all subgraphs of G that contain exactly k vertices. We define the *distance* $d_G(u, v)$ between two vertices u and v in a graph G as the minimum cost of a $u-v$ path in G; if no such path exists, the distance is undefined.

A *metric* (V, d) over a set of vertices V gives a distance $d(u, v) \geq 0$ for each vertex pair $u, v \in V$ such that the following properties hold: (1) for any $u, v \in V$, $d(u, v) = d(v, u)$, and (2) for any $u, v, w \in V$, $d(u, v) \leq d(u, w) + d(w, v)$.[2] This latter property is called the *triangle inequality*. We refer to such a metric as a *symmetric metric*; if our distance function violates property (1), then we refer to the metric as an *asymmetric metric*. In the case where the vertex set is clear from context, we will denote a metric simply by its distance function d.

Metrics can equivalently be defined as a complete weighted graph whose edge costs satisfy the triangle inequality. Any metric (V, d) can be converted into such a graph G by letting $V(G) = V$, and $c_{uv} = d(u, v)$. Further, given any simple connected weighted graph G, we can define the *shortest-path metric* corresponding to G as $(V(G), d_G)$. We use these conversion methods at various points in this thesis.

1.1.2 Optimization Problems and Approximation Algorithms

Decision problems and NP. A *decision problem* is a problem that can be answered with either "yes" or "no". We view decision problems as languages over the binary alphabet $\{0, 1\}^*$; the language L corresponding to some decision problem is the set of all strings that encode "yes" instances to the problem.

A language $L \in \mathbf{NP}$ if there are polynomials p, q and a Turing machine M (called a *verifier*) such that for each string $x \in \{0, 1\}^*$, the following holds. If $x \in L$, then a *certificate* string y of length at most $p(|x|)$ must exist such that $M(x, y)$ accepts in at most

[2] This is also sometimes called a *semimetric*, to differentiate it from a metric where the following additional property holds: for any $u, v \in V$, $d(u, v) = 0$ if and only if $u = v$.

$q(|x|)$ steps. Otherwise, for all strings y of length at most $p(|x|)$, $M(x, y)$ rejects in at most $q(|x|)$ steps. **NP** is therefore the class of all languages for which there are short and quickly verifiable yes-certificates.

Let L_1 and L_2 be two languages in **NP**. A language L_1 *reduces* to L_2 if there is a Turing machine that, given the string $x \in \{0,1\}^*$, outputs a string y such that $y \in L_2$ if and only if $x \in L_1$, and does so in $\mathbf{poly}(|x|)$ steps. A language L is **NP**-*hard* if for every language $L' \in \mathbf{NP}$, L' reduces to L. A language L is **NP**-*complete* if it is both **NP**-hard and $L \in \mathbf{NP}$.

Optimization problems. An **NP**-*optimization problem* Π consists of:

- A set of valid *instances* D_Π, where we can determine if some instance $I \in D_\Pi$ in time polynomial in $|I|$. We assume all instances $I \in D_\Pi$ can be expressed as finite binary strings; this implies that all numeric values must be integer or rational. The *size* of an instance I, written $|I|$, is the number of bits needed to describe it.
- A set of *feasible solutions* $S_\Pi(I)$ for each instance $I \in D_\Pi$, where we can determine if $s \in S_\Pi(I)$ in time polynomial in $|I|$. The length of each solution must be polynomially bounded in the size of I.
- An *objective function* obj_Π that assigns each instance-solution pair (I, s) a non-negative value, computable in time polynomial in $|I|$.

We also specify whether Π is a *minimization problem* or a *maximization problem*. For a minimization/maximization problem Π and instance $I \in D_\Pi$, an *optimal solution* is a feasible solution $s \in S_\Pi(I)$ that minimizes/maximizes the value of obj_Π; that is, $\operatorname{argmin}_{s \in S_\Pi} obj_\Pi(I, s)$ or $\operatorname{argmax}_{s \in S_\Pi} obj_\Pi(I, s)$, respectively. We denote such a solution as $OPT_\Pi(I)$, or simply OPT if the problem and instance are clear from context. We slightly abuse this notation by using OPT to also refer to the objective *value* of the optimum solution, when the type of OPT is clear from context.

An **NP** optimization problem Π gives rise to a class of **NP** decision problems, by asking if a feasible solution of at most/at least some objective value exists (for minimization/maximization problems, respectively). A polynomial time algorithm that solves Π can thus be used to answer the decision problem, while proving a hardness for a decision version shows that Π is at least as hard.

For the optimization problems we consider in this thesis, the decision versions have been shown to be **NP**-hard, and so we say the optimization problems are also **NP**-hard. Ideally we would like to find exact, polynomial time algorithms for these problems. However, unless $\mathbf{P} = \mathbf{NP}$, there is no time-efficient algorithm that can solve an **NP**-hard problem exactly, so we need a different strategy. One approach is to find an exact algorithm for the problem that runs fast enough for most inputs, but may run in exponential time in the worst case.

Another approach is to find a time-efficient algorithm that returns a solution that is very close to the exact solution for most inputs, but could be very far off in the worst case; these are called *heuristic algorithms*. The third approach, which we focus on in this thesis, is to find a time-efficient algorithm that returns a solution that is never more than a given factor worse than the exact solution. We call these *approximation algorithms*.

Approximation algorithms. Let Π be a minimization (maximization) problem, and let $\alpha : \mathbb{Z}^+ \to \mathbb{Q}^+$ be a function such that $\alpha \geq 1$ for all inputs. An algorithm $\mathcal{A}$ is an α-*approximation* for Π if, for all instances I, $\mathcal{A}$ returns a feasible solution $s \in S_\Pi(I)$ such that $obj_\Pi(I, s) \leq \alpha(|I|) \cdot OPT_\Pi(I)$ ($obj_\Pi(I, s) \geq \frac{OPT_\Pi(I)}{\alpha(|I|)}$),[3] and the running time is bounded by $\mathbf{poly}(|I|)$. The function α is called the *approximation ratio* of $\mathcal{A}$.

It is sometimes difficult to obtain an algorithm that meets this definition exactly. We might need to relax the running time constraint, for example to a quasi-polynomial factor. Or, the algorithm makes random choices, and so the approximation ratio only holds in expectation over all random choices. We still loosely refer to these as approximation algorithms, although we will state such relaxations explicitly.

An algorithm $\mathcal{A}$ is an *approximation scheme* for the minimization (maximization) problem Π if for the valid instance I and error parameter $\epsilon > 0$, it returns a feasible solution s such that $obj_\Pi(I, s) \leq (1 + \epsilon) \cdot OPT_\Pi(I)$ ($obj_\Pi(I, s) \geq (1 - \epsilon) \cdot OPT_\Pi(I)$). We call $\mathcal{A}$ a *polynomial time approximation scheme* (PTAS) if its running time is $\mathbf{poly}(|I|)$ for each fixed ϵ. We call $\mathcal{A}$ a *fully polynomial time approximation scheme* (FPTAS) if its running time is $\mathbf{poly}(|I|, 1/\epsilon)$ for each fixed ϵ.

The problem Π is said to be in the class **PTAS** or **FPTAS** if it admits the respective approximation scheme. It is said to be in the class **APX** if it admits *any* constant approximation.

Let Π and Π' be two optimization problems. Π *PTAS-reduces* to Π' if there exists an algorithm $\mathcal{A}$ and function $c : \mathbb{R}^+ \to \mathbb{R}^+$, where for each valid instance I of Π and each fixed $\epsilon > 0$,

- Algorithm $\mathcal{A}$ returns an instance $I' = \mathcal{A}(I, \epsilon)$ of Π' in time $\mathbf{poly}(|I|)$, such that if I is feasible then I' is feasible, and
- Given any feasible solution $s' \in S_{\Pi'}(I')$, there exists a feasible solution $s \in S_\Pi(I)$ such that if $obj_{\Pi'}(I', s') \leq (1 + c(\epsilon)) \cdot OPT_{\Pi'}(I')$, then $obj_\Pi(I, s) \leq (1 + \epsilon) \cdot OPT_\Pi(I)$.[4]

An optimization problem Π is said to be **APX**-*hard* if for every other problem $\Pi' \in$ **APX**, Π' PTAS-reduces to Π. If in addition $\Pi \in$ **APX**, then Π is said to be **APX**-*complete*.

[3] The function α for maximization problems is sometimes defined as the reciprocal instead, *i.e.* $\alpha \in (0, 1]$ for all inputs and $obj_\Pi(I, s) \geq \alpha(|I|) \cdot OPT_\Pi(I)$.

[4] If either problem is a maximization problem, substitute $\geq (1 - c(\epsilon))$ and $\geq (1 - \epsilon)$ as appropriate.

Let L be a language in **NP**, and Π be a minimization (maximization) problem. Let $g : \{0,1\}^* \to D_\Pi$ be a function computable in polynomial time that maps yes- and no-instances of L into instances of Π. We say g is a *gap-introducing reduction* from L to Π if a polynomial-time computable function $h : D_\Pi \to \mathbb{R}^+$ and constant α exist where

- If x is a yes-instance of L, then $OPT_\Pi(g(x)) \leq h(g(x))$ ($OPT_\Pi(g(x)) \geq h(g(x))$), and
- If x is a no-instance of L, then $OPT_\Pi(g(x)) > \alpha h(g(x))$ ($OPT_\Pi(g(x)) < h(g(x))/\alpha$).

The constant α is called the *gap size*.

Hardness of approximation. A *hardness proof* shows that a certain optimization problem cannot be approximated better than some threshold assuming certain complexity assumptions. As an extreme example, it was shown in [36] that the maximum independent set problem cannot be approximated better than $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$ assuming $\mathbf{P} \neq \mathbf{NP}$, ruling out all but the most trivial approximations. A less extreme example, implied by the PCP theorem, is that approximating MAX-3SAT better than $(1+\epsilon)$ for some $\epsilon > 0$ is **NP**-hard, ruling out a PTAS assuming $\mathbf{P} \neq \mathbf{NP}$ [33]. Since this problem is also **APX**-complete, a consequence of this hardness is that for any **APX**-hard optimization problem Π, $\Pi \notin \mathbf{PTAS}$ unless $\mathbf{P} = \mathbf{NP}$.

In this thesis, we give a hardness result that relies on a complexity assumption called the *Exponential Time Hypothesis* (ETH). Consider the k-SAT decision problem:

Definition 1.1 (k-SAT)**.** *Given a boolean formula f in conjunctive normal form, where each clause contains at most k literals, is there a satisfying truth assignment for f?*

Definition 1.2 (Exponential Time Hypothesis [22])**.** *There is no algorithm to solve k-SAT for $k \geq 3$ in time $2^{o(n)}$ unless $\mathbf{P} = \mathbf{NP}$.*

We use this hypothesis in Chapter 4 to show that a restricted version of deadline orienteering is not super-constant hard.

1.1.3 Linear Programming

Many problems in **NP** can be formulated as an *integer program* that describes the problem. Let $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$ be vectors, and $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$ be a matrix. Let $\cdot$ denote the dot-product of two vectors. The integer programming problem is to find a binary vector $x \in \{0,1\}^n$ minimizing the value $c \cdot x$, satisfying:

$$Ax \geq b.$$

Note that we can use this definition to define maximization problems as well (*i.e.* by minimizing $-c \cdot x$), and allow for $\leq$ and $=$ constraints.

Finding such a binary vector, or determining if such a vector even exists, is itself an **NP**-hard problem in general (otherwise, we could use integer programming to solve other **NP**-hard problems). Instead, suppose we relax this problem: instead of trying to find a binary vector x, we try to find a satisfying $x \in \mathbb{Q}^n$. This yields a *linear program*:

$$\begin{aligned} &\text{minimize} \quad c \cdot x \\ &\text{subject to} \quad Ax \geq b, \\ &\qquad\qquad x \geq 0. \end{aligned} \tag{LP}$$

It is usually more convenient to explicitly write out the constraints and the objective function rather than specifying A, b, c directly, as in the following (equivalent) LP:

$$\begin{aligned} &\text{minimize} \quad \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad i = 1, \ldots, m, \\ &\qquad\qquad x_j \geq 0, \qquad j = 1, \ldots, n. \end{aligned} \tag{LP}$$

We say that we "solve" a linear program if we either determine no solution x exists, the value $c \cdot x$ is unbounded, or return a solution minimizing the objective $c \cdot x$. Unlike integer programs, linear programs can be solved in time polynomial in n, m, and the number of bits Δ required to write the rational entries of A, b, and c; one such approach is the *interior point method* (see, for example, [23]). We can remove the running-time dependence on m if we are instead given an *oracle* function, which determines if a given vector x satisfies all constraints, or if not will return a violated constraint. The *ellipsoid method* can be used to solve linear programs of this form [20].

Duality. We say the following linear program is the *dual* of (LP) (which we call the *primal*):

$$\begin{aligned} &\text{maximize} \quad b \cdot y \\ &\text{subject to} \quad A^T y \leq c, \\ &\qquad\qquad y \geq 0. \end{aligned} \tag{DP}$$

There are some fundamental relations between a primal linear program and its dual. Note that since we can rewrite a maximization linear program in a minimization form and vise versa, (DP) also has a dual, namely (LP). Some other useful properties include:

Theorem 1.3 (Weak Duality)**.** *If x is a feasible solution to the primal* (LP) *and y is a feasible solution to the dual* (DP)*, then $c \cdot x \geq b \cdot y$.*

In fact, we can strengthen this when x and y are optimal solutions:

Theorem 1.4 (Strong Duality)**.** *If x^* is an optimal solution to the primal* (LP) *and y^* is an optimal solution to the dual* (DP)*, then $c \cdot x^* = b \cdot y^*$.*

Let x, y be feasible solutions to (LP) and (DP) respectively. We say x and y obey the *complementary slackness conditions* if for each $x_j > 0$, $\sum_{i=1}^{m} a_{ij} y_i = c_j$ and for each $y_i > 0$, $\sum_{j=1}^{n} a_{ij} x_j = b_i$. An interesting consequence of Strong Duality is the following:

Theorem 1.5 (Complementary Slackness)**.** *The complementary slackness conditions for the feasible solutions x, y hold if and only if x and y are both optimal solutions to their respective linear programs.*

Using duality, we can sometimes solve linear programs that have exponentially many variables, which would otherwise be unsolvable. The dual for such a program has exponentially many constraints, which can be solved using the ellipsoid method given a suitable separation oracle. By complementary slackness, all variables x_j in the primal must be 0 if their corresponding dual constraint are not tight. We can further form a basis of the tight dual constraints; the primal variables corresponding to any remaining tight constraints can also be set to 0. These zeroed variables can be deleted from the primal, leaving a polynomial size linear program that can be solved.

Usefulness in approximations. Linear programming is a useful tool to build approximation algorithms with. The general procedure is to write the (relaxed) integer program, solve it, and try to round the fractional result to an integer solution in polynomial time, without either violating constraints or increasing the objective value significantly. If we can do this while only increasing the objective value by a factor of $f(n)$, where $n = |x|$, then we will have an $f(n)$-approximation to the original integer program.

We say a linear program has an *integrality gap* of $f(n)$ if for an optimum solution x^* and optimum solution $\bar{x}$ for the corresponding integer program, $\frac{c \cdot x^*}{c \cdot \bar{x}} \leq f(n)$. Proving such a gap gives a strong indication that an efficient approximation algorithm with a similar ratio should exist, and the proof typically does yield an efficient algorithm. However, this is not always the case.

1.1.4 Pipage Rounding

There are many ways to round a solution to a linear program; *pipage rounding* is one technique first introduced by Ageev and Sviridenko [1]. This is a general rounding scheme that works with linear programs of a specific form. We will use this approach in Chapter 2 to round a linear program.

In the pipage rounding scheme, we wish to approximately solve a generalized version of the following bipartite matching problem. We are given a bipartite graph $H = (U \cup W, E)$, vertex capacities p_v, and a poly-time computable function $F(x)$ defined over the vectors

$x = (x_e : e \in E)$, $x_e \in [0, 1]$. We wish to pick a collection of edges from E such that each vertex v has at most p_v edges incident with it, maximizing the value of $F(x)$; if we pick edge e for our collection, then we set $x_e = 1$, and $x_e = 0$ otherwise. Note that with $p_v = 1$ for all v and $F(x) = \sum_{e \in E} x_e$, this becomes the maximum bipartite matching problem.

This general problem can be expressed as an integer program. The relaxed version, where a solution may be a rational vector, is given below:

$$\begin{aligned} &\max \ F(x) && \text{(PIPE-LP)} \\ &\text{s.t.} \quad \sum_{e \in N(v)} x_e \leq p_v \quad \forall v \in (U \cup W) && (1.1) \\ &\qquad x \in [0, 1]. \end{aligned}$$

Note that $F(x)$ may not be a linear function, so as written this may not be a linear program and so may not be solvable using standard techniques. For our purposes however, will assume that some fractional solution x has been provided.

Let F^* be the value of an optimal *integer* solution to (PIPE-LP), and x a fractional solution to (PIPE-LP). The pipage rounding algorithm transforms the solution x into an integral solution $\bar{x}$ which, given some conditions on F, will have the property that $F(\bar{x}) \geq F(x)$. If, for an optimal fractional solution $\check{x}$, we also had $F(\check{x}) \geq F^*/\alpha$, then we would have an α-approximate solution to the original problem.

The algorithm. The pipage rounding algorithm is an iterative procedure, where in each step we convert a fractional solution x to a new solution x' with at least one less fractional component. The algorithm terminates when $x = \bar{x}$ is integral.

In each step, if we do not terminate, then x has some non-integral entry. Consider the bipartite subgraph H_x of H, where edge $e \in E$ is in H_x if and only if x_e is non-integral. Let R be a cycle in H_x, or, if no cycle exists, a path whose endpoints have degree 1 in H_x. In either case, since H_x is bipartite the cycle/path R can be represented as the union of two matchings M_1 and M_2. We will compute a new solution $x(\epsilon, R)$ using these matchings; if $e \in M_1$, then $x_e(\epsilon, R) = x_e + \epsilon$; otherwise if $e \in M_2$, then $x_e(\epsilon, R) = x_e - \epsilon$; otherwise $x_e(\epsilon, R) = x_e$.

Let ϵ_1 be the smallest ϵ we can subtract such that some $e \in M_1$ becomes 0 or $e \in M_2$ becomes 1, and let ϵ_2 be the smallest ϵ we can add such that some $e \in M_1$ becomes 1 or $e \in M_2$ becomes 0. Let $x_1 = x(-\epsilon_1, R)$, and $x_2 = x(\epsilon_2, R)$. Set $x' = x_1$ if $F(x_1) > F(x_2)$, and $x' = x_2$ otherwise. This concludes one iteration of the algorithm.

In order for a solution returned by this algorithm to have the property that $F(\bar{x}) \geq F(x)$, we require that in each step $F(x') \geq F(x)$. This latter inequality holds when $F(x(\epsilon, R))$, for $\epsilon \in [-\epsilon_1, \epsilon_2]$, is maximized at either endpoint of the interval. We call this the *ϵ-convexity*

condition.[5]

Definition 1.6. *The function F is* ϵ-convex *if, for any step of the pipage rounding algorithm and for $\epsilon \in [-\epsilon_1, \epsilon_2]$, $F(x(\epsilon, R))$ is maximized at either $-\epsilon_1$ or ϵ_2.*

Bounding the integrality gap. To bound the integrality gap of (PIPE-LP) for an arbitrary F, we can use the following technique. Suppose we are given a second poly-time computable function $L(x)$, defined over the same set of vectors x as F, and where the following conditions hold:

Condition 1.7. *For binary x, $L(x) = F(x)$.*

Condition 1.8. *For any optimal fractional solution $\check{x}$, $F(\check{x}) \geq L(\check{x})/\alpha$.*

These are called the *F/L lower bound conditions*. If (PIPE-LP) is poly-time solvable when the objective is to maximize $L(x)$ instead of $F(x)$, then since $L(\check{x}) \geq F^*$ (by condition 1.7), by condition 1.8 we would then have $F(\check{x}) \geq F^*/\alpha$, as desired. The trick is then to choose functions F and L satisfying these conditions, in addition to the ϵ-convexity condition on F.

1.2 Problems Considered

The problems we consider in this thesis are the following.

Capacitated travelling repairmen problems. We consider extensions of the *multi-depot k-travelling repairmen problem* (MD-kTRP). In this problem, we are given a collection of clients $C = \{c_1, c_2, \ldots, c_n\}$ that need to be served by a vehicle, and a collection of k identical vehicles stationed at given depots (roots) $R = \{r_1, r_2, \ldots, r_k\}$. The objective of the problem is to find a routing of the vehicles over the symmetric metric $(R \cup C, d)$, such that every client's demand is satisfied, minimizing the average time required to service all clients.

We extend this problem to include capacities in the following sense: suppose each client additionally has a positive demand which needs to be satisfied, given as the function $w : C \to \mathbb{Z}^+$, and the vehicles have a fixed positive capacity Q. The routing we compute must now obey the following additional constraints:

1. All clients must be completely served in a single trip (called *unsplit delivery*), and
2. A vehicle can serve a total of at most Q demand between visits to its depot (the visits to the depot are called *resupplies*).

[5] Note that any F that is a linear function of x satisfies this condition.

We call this problem the *multi-depot capacitated k-travelling repairmen problem* (MD-CkTRP). This generalization of the TRP nicely models many scenarios in package delivery management, where clients require packages of certain sizes to be delivered by vehicles with limited carrying capacity. We can further generalize the model to the cases where vehicles have different carrying capacities, or where clients have a service delay $\delta(c)$ (the delay is added to the latency of client c and all clients visited by the vehicle after c), or both. Note that if vehicles had infinite capacity ($Q = \infty$), then the problem is equivalent to the MD-kTRP. In Chapter 3, we will give approximation algorithms for all of these problems.

The deadline orienteering problem. In this problem, we are given a symmetric metric $d(u, v)$, non-negative deadlines $D(v)$ for each vertex v, and a source-sink pair r, s. The goal is to find a path P from r to s of length at most $D(s)$, containing as many vertices as possible, such that for each vertex v along the path, the distance from r to v along the path P is at most $D(v)$ (notated as $d_P(r, v) \leq D(v)$). A more general but more difficult version of the problem is called *orienteering with time-windows*; in this version, each vertex also has a release time $R(v)$, and a vertex v may only be included in a path P if $R(v) \leq d_P(r, v) \leq D(v)$.

Both of these problems are difficult to approximate well. We primarily focus our efforts on approximating restricted versions of the deadline orienteering problem, although our results carry over to the corresponding time-window version without much extra effort. We consider these problems in Chapter 4.

Regret orienteering problems. We consider a few orienteering style problems that are closely related to deadline orienteering: instead of vertices having a fixed deadline $D(v)$, each vertex v experiences *regret* along path P (denoted $Reg_P(v)$) if it is visited by P at time $d(r, v) + Reg_P(v)$, where $d(r, v)$ is the metric distance from r to v. Our objective is to find a path P visiting as many vertices as possible, with a constraint on the regret experienced by a vertex.

We consider two different constraints on the regret: (1) we are given a parameter γ, and wish to ensure that the average regret for each visited vertex is at most γ, and (2) we are given per-vertex regret bounds $B(v)$, and must ensure that $Reg_P(v) \leq B(v)$. We call these problems Max-Average-RVRP and Max-Bounded-RVRP, respectively. Deadline orienteering is captured by Max-Bounded-RVRP, by letting $B(v) = D(v) - d(r, v)$, but our approximation algorithm will violate the regret bounds by a small constant. Since the regret bounds are typically much smaller than the corresponding deadline, this might be a more interesting result. We consider these problems in Chapter 4.

The maximum coverage problem with groups (MCG). This is not a vehicle routing problem, but it arises as a sub-problem in our approximation algorithms. We are given a

set of items $I = \{e_1, e_2, \ldots, e_n\}$, a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of I, and a partition $G_1, G_2, \ldots, G_k$ of $\mathcal{S}$ (we call the sets G_i *groups*). Our goal is to choose a single subset from each G_i to maximize the total number of items covered. For our applications m may be exponentially larger than n - in this case the set $\mathcal{S}$ is not given explicitly, but implicitly with an *oracle* function (that may itself be an **NP**-hard problem to solve), such that the instance size becomes polynomial in n. We consider this in Chapter 2.

1.3 Prior Work

We now review the prior work done on the problems we consider in this thesis.

Capacitated travelling repairmen problems. The original travelling repairman problem (TRP) is the following: we are given a collection of clients and a root node r distributed over a metric and are asked to find an r-rooted tour covering all clients, such that the *average distance* along the tour from r to any client c is minimized. This problem was first considered in more restricted settings (esp. when the metric is a tree) by the Operations Research and Network Research communities [26, 24, 4], before the first true approximation for the general case was found by Blum *et al.* [5].

Currently, the best approximation known for the TRP on an arbitrary, symmetric metric is the 3.59-approximation by Chaudhuri *et al.* [10], although a PTAS was recently found by Sitters for the case where the metric is the shortest-path completion metric of an edge-weighted tree [31]. In the case of an asymmetric metric, a $O(n^{1/2+\epsilon})$-approximation was first found by Nagarajan and Ravi [27], which was later improved to $O(\log n)$ by Friggstad, Salavatipour and Svitkina [18]. All of these problems are known to be **NP**-hard [30], and on a general metric the TRP is known to also be **APX**-hard [5].

A well-studied generalization of the problem is the case where we must find k rooted tours instead of just one; we require that all clients are covered by some tour, and try to minimize the average distance among all tours from the roots to the clients. For the case where all tours must be rooted at a single root, an 8.497-approximation was first given by Fakcharoenphol *et al.* [16] for symmetric metrics, and this was recently improved to 7.183 by Post and Swamy [28].

For the case where the roots may differ (referred to as the multi-depot k-travelling repairmen problem), a 24-approximation[6] was given by Chekuri and Kumar [12]; this was also recently improved by Post and Swamy to 8.497 [28]. We study the capacitated version of this problem in Chapter 3, where we add client demands and each vehicle must (1) satisfy a client's entire demand in a single trip, and (2) serve at most Q client demand between visits to its depot.

[6]The authors initially stated their algorithm was a 12-approximation, but a minor bug was later found in their analysis.

Our problem has been studied previously in the Operations Research community, where it is also referred to as the *multiple depot cumulative capacitated vehicle routing problem with multiple trips*. Heuristic approaches to solve the problem have been examined recently, for example by Lysgaard and Wohlk [25] and Rivera *et al.* [29]. Since our problem is a superset of the TRP problem, we can trivially see that MD-CkTRP is both **NP**-hard and **APX**-hard on general metrics. However, no approximation guarantee is currently known.

The deadline orienteering problem. Both the deadline orienteering problem and orienteering problem with time-window have been studied extensively in the Operations Research community (see [15] for a survey), and many different approaches have been studied for solving these problem exactly.

Both problems have also been studied from an approximation perspective. Bansal *et al.* [3] presented the first true, polynomial time $O(\log n)$-approximation to the deadline orienteering problem on a symmetric metric, and they then showed a simple technique that allows certain α-approximations to deadline orienteering to be converted into an α^2-approximation to the time-windows version, yielding approximations to both problems. They also gave a bi-criteria approximation for the deadline problem that violates deadlines by a $(1+\epsilon)$-factor to obtain a better approximation ratio of $O(\log \frac{1}{\epsilon})$. Chekuri and Kumar [12] gave simpler, constant approximations to both problems that run in time polynomial in $(n\Delta)^k$, where k is the number of distinct deadlines and $\Delta = \max_{u,v} d(u,v)$. Thus, their approximations are not polynomial time in general settings.

More recently, an $O(\max\{\log n, \log \frac{L_{max}}{L_{min}}\})$-approximation was discovered for the time-window problem (where L_{max}/L_{min} are the longest/shortest time windows, respectively); this was due to Chekuri, Korula, and Pál [11]. They also gave approximations for the asymmetric versions of the orienteering, deadline orienteering and orienteering-with-time-window problems, losing a $O(\log^2 n)$ factor over the symmetric versions. All of these orienteering problems, both symmetric and asymmetric, are known to be **APX**-hard [6].

Regret orienteering problems. The school-bus problem has been well-studied; Bock *et al.* [7] gave an $O(\log n)$ approximation for the problem, which was later improved by Friggstad and Swamy [19] to a constant approximation. The versions we consider have not, to the best of our knowledge, been studied previously.

The maximum coverage problem with groups. This problem (and its variations) was first studied by Chekuri and Kumar [12], who used it as a subroutine to obtain the first constant-factor approximation for the multiple depot k-travelling repairmen problem. Their initial approximations to the MCG problem have not been improved since their original paper.

1.4 New Results

The main contributions of this thesis are the following.

The maximum coverage problem with groups. In Chapter 2, we use LP techniques to improve the approximation of Chekuri and Kumar [12] to the MCG. Given a ρ-approximate oracle, we show how to improve their $(\rho + 1)$ approximation to a $\left(\frac{e^{1/\rho}}{e^{1/\rho}-1}\right)$ approximation. Our approximation additionally bounds the integrality gap of the LP we use, and the approximation ratio we obtain matches that achievable with pipage rounding when the oracle is exactly solvable.

Capacitated travelling repairmen problems. In Chapter 3, we present an algorithm framework for capacitated travelling repairmen problems, and use it to show the following:

Theorem 1.9. *There is a 25.49-approximation to the unit-demand MD-Ck TRP (*i.e. $w(c) = 1$ *for all $c \in C$).*

Theorem 1.10. *There is a 42.49-approximation to the unsplit-delivery MD-Ck TRP.*

Our algorithm can be extended to solve the problem when we have non-uniform vehicle capacities with no additional loss. We can further extend the problem to include client service delays $\delta(c)$, with an additional $+0.5$ factor loss in approximation. We also obtain an 8.497-approximation when vehicles have infinite capacity, matching the current-best result by Post and Swamy [28].

Deadline orienteering and regret orienteering problems. In Chapter 4, we present a few different results that provide further insight into the deadline and time-window orienteering problems.

Recall that in the Max-Average-RVRP, we wish to cover as many vertices as possible ensuring that the average regret for each visited vertex is bounded by a constant γ. In the Max-Bounded-RVRP we also wish to cover as many vertices as possible, but must ensure each visited vertex v does not have regret more than $B(v)$. We obtain the following results for these problems:

Theorem 1.11. *There is an $(8 + \epsilon)$-approximation to the* Max-Average-RVRP*, which finds a path covering at least a $\frac{1}{8+\epsilon}$ fraction of the optimal number of vertices.*

Theorem 1.12. *There is a $(2.542, 4)$-approximation to the* Max-Bounded-RVRP *running in quasi-polynomial time when regrets $B(v)$ are poly-bounded, which finds a path covering at least a $\frac{1}{2.542+\epsilon}$ fraction of the optimal number of vertices and violating regret bounds $B(v)$ by at most a factor of 4.*

For the deadline orienteering problem, we first show the following result:

Theorem 1.13. *There is a polynomial-time algorithm to solve deadline orienteering and orienteering with time-windows when the input graph is a DAG.*

We then restrict our focus to a slightly more restricted version of the problem: we assume that the input graph/metric has poly-bounded edge lengths. This allows us to obtain the following:

Theorem 1.14. *There is an $O(1)$-approximation to deadline orienteering on a general metric with poly-bounded edge lengths running in sub-exponential time.*

Our final corollary follows from the previous theorem:

Corollary 1.15. *No $\omega(1)$-hardness result for the deadline orienteering problem with poly-bounded edge lengths is possible under the assumption $\mathbf{P} \neq \mathbf{NP}$, unless the Exponential Time Hypothesis is false.*

This gives a strong (although not definitive) indication that a sub-logarithmic approximation can be found for the original deadline orienteering and orienteering with time-window problems.[7]

We conclude this thesis by discussing future research directions in Chapter 5.

[7] It is still possible that a weaker hardness result exists if, for example, we assume $\mathbf{NP} \neq \mathbf{QP}$. This is not excluded by our result, but it seems unlikely.

Chapter 2

Approximating Maximum Coverage with Group Budget Constraints

We first consider the *maximum coverage problem with group budget constraints* (shortened to max coverage with groups, or simply the MCG problem). This problem appears as a key subroutine in our latency approximations, which we will develop in a later chapter. To improve those results, we focus on developing an approximation algorithm that uses linear programming techniques.

2.1 Problem Overview

The version of the MCG problem we are interested in can be described as follows. We are given a set of items $I = \{e_1, e_2, \ldots, e_n\}$, a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of I, and a partition $G_1, G_2, \ldots, G_k$ of $\mathcal{S}$ (we call the sets G_i *groups*). Our goal is to choose a single subset from each G_i to maximize the total number of items covered. An α-approximation to this problem will find such a collection of subsets that covers at least a $1/\alpha$-fraction of the optimal number of elements.

This problem is in fact a special case of submodular function maximization subject to a matroid constraint: the instance can be represented by the combination of a monotone submodular function $f(S)$ denoting the number of elements covered by the set S, and a partition matroid $\mathcal{M}$ over the sets in $\mathcal{S}$ that defines the groups. Călinescu *et al.* [8] showed how to obtain a $\left(\frac{e}{e-1}\right)$-approximation for this problem with running time polynomial in $|\mathcal{S}|$ and $|I|$.

More direct techniques can also be used if $|\mathcal{S}|$ is polynomially bounded. For example, the pipage rounding technique of Ageev and Sviridenko [1] can be directly applied, yielding a deterministic $\left(\frac{e}{e-1}\right)$-approximation [12]. A randomized algorithm with the same ratio can be achieved using the probabilistic rounding techniques of Srinivasan [32].

We are interested in the case where $\mathcal{S}$ might be exponentially large with respect to $|I|$. In this case, we cannot explicitly describe $\mathcal{S}$ or the groups G_i, since the size of the problem instance is no longer polynomially bounded in $|I|$. Suppose we were given a polynomial-time oracle $\mathcal{A}(i, w)$ that takes as input a group index i and a weight function $w : I \to \{0, 1\}$. If this oracle can return a set $S \in G_i$ such that $w(S) = \sum_{e \in S} w(e)$ is maximized, then we can approximately solve the corresponding MCG instance with a factor 2 loss [12].

Typically the oracle $\mathcal{A}$ is itself an **NP**-hard problem. Suppose we were instead given a ρ-approximation for $\mathcal{A}$; that is, $\mathcal{A}$ will always return a set S with $w(S) \geq \frac{1}{\rho} \max_{S' \in G_i} w(S')$. We call this a ρ-approximate oracle. Chekuri and Kumar showed that a simple greedy algorithm yields a $(\rho + 1)$-approximation to the MCG instance the oracle $\mathcal{A}$ describes, which we restate as Theorem 2.1.

Theorem 2.1 ([12])**.** *There is a polynomial-time $(\rho + 1)$-approximation algorithm for the MCG problem that covers a $\frac{1}{\rho+1}$-fraction of the optimal number of elements, given a ρ-approximate oracle.*

2.1.1 Our Results

Using LP techniques, we will show how to improve this $(\rho + 1)$ approximation to $\left(\frac{e^{1/\rho}}{e^{1/\rho}-1}\right)$, where e is Euler's number, given that we have a *weighted* version of $\mathcal{A}$; that is, the function w provided to $\mathcal{A}$ returns any non-negative rational value instead of the binary values $\{0, 1\}$. Most oracles can be converted to this form with a small loss in approximation - we cover the details of this transformation later in the chapter. Our approximation will additionally bound the integrality gap of the LP we use.

Note that the approximation ratio we obtain will match that achievable with pipage rounding when $\mathcal{A}$ is exactly solvable. This is no accident; we will eventually show how to integrate the oracle into the pipage rounding scheme, leveraging both to achieve our final approximation. We first describe a simple randomized LP-rounding algorithm to solve the MCG problem in sections 2.2 and 2.3. We then show in section 2.4 how to use pipage rounding to derive a deterministic algorithm with the same approximation ratio.

2.2 A Linear Programming Relaxation for MCG

We first start with an integer programming formulation of the MCG problem, and show how to (approximately) solve its linear relaxation using the weighted version of $\mathcal{A}$. For item e and group G_i, let x_e^i be a binary variable that indicates whether item e is being covered by a set from group G_i or not. For a set $S \in \mathcal{S}$, let z_S be a binary variable indicating whether set S is chosen to form a part of the solution. Using the problem definition, we can express the MCG problem as the following integer program (IP):

$$\begin{aligned}
\max \quad & \sum_{e,i} x_e^i && && \text{(IP)}\\
\text{s.t.} \quad & \sum_i x_e^i \leq 1 && \forall e && (2.1)\\
& \sum_{S \in G_i} z_S \leq 1 && \forall i && (2.2)\\
& \sum_{S \in G_i : S \ni e} z_S \geq x_e^i && \forall e, i && (2.3)\\
& x, z \in \{0,1\}.
\end{aligned}$$

The constraints (2.1) prevent a solution from counting an item more than once, and similarly constraints (2.2) prevent a solution from choosing more than one set from any group. Constraints (2.3) enforce that an item can be covered only if a set containing it is chosen.

Solving an integer program is **NP**-hard; we therefore consider the linear relaxation of (IP), where the x and z variables may be any non-negative rational number at most 1. This relaxation is given as (LP). We will also require the corresponding dual program, given as (DP) (the dual variables are listed alongside the corresponding primal constraints).

$$\begin{aligned}
\max \quad & \sum_{e,i} x_e^i && && && \text{(LP)}\\
\text{s.t.} \quad & \sum_i x_e^i \leq 1 && \forall e && (\alpha_e) && (2.4)\\
& \sum_{S \in G_i} z_S \leq 1 && \forall i && (\beta_i) && (2.5)\\
& \sum_{S \in G_i : S \ni e} z_S \geq x_e^i && \forall e, i && (\theta_e^i) && (2.6)\\
& x, z \geq 0.
\end{aligned}$$

$$\begin{aligned}
\min \quad & \sum_e \alpha_e + \sum_i \beta_i && && \text{(DP)}\\
\text{s.t.} \quad & \alpha_e + \theta_e^i \geq 1 && \forall e, i && (2.7)\\
& \sum_{e \in S} \theta_e^i \leq \beta_i && \forall i, S \in G_i && (2.8)\\
& \alpha, \beta, \theta \geq 0.
\end{aligned}$$

Let OPT_{LP} be the optimal objective value of (LP); by strong duality this is also the optimal objective value of (DP). In the case where $|\mathcal{S}|$ is exponentially large, note that (LP) contains exponentially many variables, with a polynomial number of constraints. Similarly, the dual (DP) contains a polynomial number of variables with exponentially many constraints. While we cannot directly solve the primal, we *can* solve the dual using the ellipsoid method given a suitable *separation oracle*. The separation oracle must, in polynomial time, determine whether a particular dual solution (α, β, θ) violates any dual constraints - such a collection of violated constraints defines a *separating hyperplane*, which is used by the ellipsoid algorithm to constrain the possible values of the optimum dual solution.

Separating the dual. Determining whether any of constraints (2.7) are violated can be

done trivially in polynomial time. Determining if any of constraints (2.8) are violated is more difficult - we cannot check all of them. We can instead solve the following problem, using our (weighted) oracle $\mathcal{A}$: given a group G_i and item rewards θ_e^i, we must determine whether a set $S \in G_i$ that collects more than β_i reward exists. Define the weight $w(e)$ of item e as the value θ_e^i. If running $\mathcal{A}$ on inputs i and w will always return an optimal set S, then some constraint (2.8) for i is violated if and only if $w(S) > \beta_i$. Using this, we can then separate and exactly solve the dual, which yields an optimal solution to the primal by strong duality.

In the case where $\mathcal{A}$ may only return an approximate solution, we can instead apply a technique originally proposed by Carr and Vampala [9] to approximately solve (DP) and (LP). Here we adapt the more recent presentation of this technique by Friggstad and Swamy [19] to suit our more complex LP. Define the following polytope:

$$\mathcal{P}(\upsilon, a) = \{(\alpha, \beta, \theta) : (2.7), (2.8), \sum_e \alpha_e + a \sum_i \beta_i \leq \upsilon\}.$$

Note that $\mathcal{P}(OPT_{LP}, 1)$ defines the collection of optimum solutions for (DP), and so OPT_{LP} is the largest υ such that $\mathcal{P}(\upsilon, 1) \neq \emptyset$.

Using our ρ-approximate oracle $\mathcal{A}$, given some υ and point (α, β, θ), we can certify that either $(\alpha, \rho\beta, \theta) \in \mathcal{P}(\upsilon, 1)$, or give a hyperplane certifying that $(\alpha, \beta, \theta) \notin \mathcal{P}(\upsilon, \rho)$, as follows. If (α, β, θ) violates some constraint (2.7) or the constraint $\sum_e \alpha_e + \rho \sum_i \beta_i \leq \upsilon$, then we return the constraint as a separating hyperplane. Otherwise, for each i, we run $\mathcal{A}$ with element weights θ_e^i. If the returned set S has weight $w(S) > \beta_i$, then since $w(S) \geq (1/\rho) \max_{S' \in G_i} w(S')$, we can return the constraint (2.8) corresponding to i, S as the separating hyperplane. Otherwise, we must have that for all $S \in G_i$, $w(S) \leq \rho\beta_i$, and so $(\alpha, \rho\beta, \theta) \in \mathcal{P}(\upsilon, 1)$.

Using this approximate separation, the ellipsoid algorithm will certify in polynomial time that either $\mathcal{P}(\upsilon, \rho) = \emptyset$, or give a point $(\alpha, \rho\beta, \theta) \in \mathcal{P}(\upsilon, 1)$. We can then use binary search to find the smallest value υ^* such that $\mathcal{P}(\upsilon^*, 1)$ is non-empty, and so $\upsilon^* \geq OPT_{LP}$.

Obtaining a good solution to (LP)**.** Suppose we run the ellipsoid algorithm with input $\upsilon^* - \epsilon$ for any $\epsilon > 0$; this must necessarily produce a certificate showing $\mathcal{P}(\upsilon^* - \epsilon, \rho) = \emptyset$. This certificate will consist of the polynomially-many separating hyperplanes provided by the oracle during the run, including the inequality $\sum_e \alpha_e + \rho \sum_i \beta_i \leq \upsilon^* - \epsilon$. Consider the following polytope, which is the dual of $\mathcal{P}(\upsilon, a)$:

$$\mathcal{Q}(\upsilon, a) = \{(x, z) : (2.4), \sum_{S \in G_i} z_S \geq a, (2.6), \sum_{e,i} x_e^i \geq \upsilon\}.$$

By duality, the certificate we obtained corresponds to a point $(x, z) \in \mathcal{Q}(\upsilon^* - \epsilon, \rho)$ with polynomially-many non-zero variables.[1] Note that the point $(x/\rho, z/\rho)$ is a feasible solution

[1] We can obtain such a point by setting all variables x_e^i and z_S that are not in our certificate to zero

to (LP) with objective value $(v^* - \epsilon)/\rho \geq (OPT_{LP} - \epsilon)/\rho$; we thus have an approximate solution to (LP). Further, the point (x, z) is *almost* a feasible solution to (LP) that only violates (2.5); this latter solution has objective value $v^* - \epsilon \geq OPT_{LP} - \epsilon$.[2] This property will be crucial to our rounding scheme.

2.3 A Randomized $\left(\frac{e^{1/\rho}}{e^{1/\rho}-1}\right)$-Approximation

Given a feasible solution $(x/\rho, z/\rho)$ to (LP), where the solution (x, z) still satisfies constraints (2.4) and (2.6), we now show how to (randomly) round it to a feasible integer solution with expected objective value at least $\frac{e^{1/\rho}-1}{e^{1/\rho}} \cdot OPT_{LP}$. The algorithm presented here will additionally provide an integrality gap proof for (LP).

Without loss of generality, we can assume that constraints (2.5) are tight for the feasible solution - that is, $\sum_{S \in G_i} z_S/\rho = 1$ for all i. If this is not the case for some i, then since the groups G_i define a partition, we can safely raise z values within that group without violating any other constraint. Given this assumption, the rounding algorithm is straight-forward: for each group i, sample a set randomly from the distribution given by $\{z_S/\rho\}_{S \in G_i}$. Only a polynomial number of z values can be non-zero, so this sampling algorithm will run in polynomial time.

Theorem 2.2. *Randomized rounding is a $\left(\frac{e^{1/\rho}}{e^{1/\rho}-1}\right)$-approximation to the MCG problem, given a ρ-approximate oracle. Furthermore, the integrality gap of* (LP) *is at most $\frac{e}{e-1}$.*

Proof. We proceed by first bounding the probability that an arbitrary item $\bar{e}$ is *not* covered by the collection of sets we pick. We then show that the expected contribution of $\bar{e}$ to our solution is at least a constant fraction of its contribution to OPT_{LP}; since OPT_{LP} is at least the integer optimum, this implies that we cover a constant fraction of the optimal number of items in expectation.

Let $n_{\bar{e}}$ be the number of sets that contain $\bar{e}$, and let $a_e^i = \sum_{S \in G_i : S \ni e} z_S/\rho$. Note that the probability that $\bar{e}$ is not covered by any set we picked is at most $\prod_i (1 - a_{\bar{e}}^i)$. By the arithmetic-geometric mean inequality, we have

$$\prod_i (1 - a_{\bar{e}}^i) \leq \left(1 - \frac{1}{n_{\bar{e}}} \sum_i a_{\bar{e}}^i\right)^{n_{\bar{e}}} \leq \left(1 - \frac{1}{\rho n_{\bar{e}}} \min(1, \sum_{S \ni \bar{e}} z_S)\right)^{n_{\bar{e}}}.$$

Thus, the probability that $\bar{e}$ *is* covered by our solution is at least

$$1 - \left(1 - \frac{1}{\rho n_{\bar{e}}} \min(1, \sum_{S \ni \bar{e}} z_S)\right)^{n_{\bar{e}}}. \tag{2.9}$$

Let the value $x_{\bar{e}} = \sum_i x_{\bar{e}}^i$ be the total contribution of $\bar{e}$ to the LP objective value of the (infeasible) solution (x, z); we noted before that this solution has total objective value at

(or alternatively, deleting them altogether), and solving the newly obtained linear program, which is now polynomially-large.

[2] We omit the ϵ for the remainder of this discussion for clarity.

least OPT_{LP}. If $\sum_{S \ni \bar{e}} z_S \geq 1$, then we must have $x_{\bar{e}} = 1$ by constraints (2.4) and (2.6); otherwise we could raise some $x^i_{\bar{e}}$ value without violating any constraints to create a better solution. Alternatively, if $\sum_{S \ni \bar{e}} z_S < 1$, then by similar reasoning constraints (2.6) involving $\bar{e}$ must be tight, so $x_{\bar{e}} = \sum_{S \ni \bar{e}} z_S$. Thus, $x_{\bar{e}} = \min(1, \sum_{S \ni \bar{e}} z_S)$.

Substituting this into (2.9), note that the function $1-(1-x_{\bar{e}}/(\rho n_{\bar{e}}))^{n_{\bar{e}}}$ is concave for $x_{\bar{e}} \in [0,1]$ and $\rho \geq 1$; further, for $x_{\bar{e}} = 0$ and $x_{\bar{e}} = 1$ we obtain the values 0 and $1-(1-1/(\rho n_{\bar{e}}))^{n_{\bar{e}}}$, respectively. Thus, using properties of the exponential function we have

$$1 - \left(1 - \frac{x_{\bar{e}}}{\rho n_{\bar{e}}}\right)^{n_{\bar{e}}} \geq 1 - x_{\bar{e}} \left(1 - \frac{1}{\rho n_{\bar{e}}}\right)^{n_{\bar{e}}} \geq x_{\bar{e}}(1 - e^{-1/\rho}).$$

In expectation then our integer solution collects at least $(1 - e^{-1/\rho}) \cdot OPT_{LP}$ items, yielding a randomized $\left(\frac{e^{1/\rho}}{e^{1/\rho}-1}\right)$-approximation. This further shows that the integrality gap of (LP) is $\frac{e}{e-1}$. □

2.4 Derandomization

To derandomize our rounding scheme, we make use of the pipage rounding technique, as described by Ageev and Sviridenko [1]. The rounding algorithm we present will match the result we obtained in Theorem 2.2, but is completely deterministic.

Our approach to solve the MCG problem follows the ideas used by Ageev and Svirendenko [1] to prove a similar integrality gap for the standard Maximum Coverage problem. First note that (LP) is equivalent to the following program:

$$\max \quad \sum_e \min\left(1, \sum_{S \ni e} z_S\right) \tag{LP2}$$

$$\text{s.t.} \quad \sum_{S \in G_i} z_S \leq 1 \qquad \forall i \tag{2.10}$$

$$z_S \in [0,1].$$

Here, we have rewritten constraints (2.4) and (2.6) as the minimum in the objective. Since this program is equivalent to (LP), given a fractional solution $(x/\rho, z/\rho)$ to (LP), we can obtain a solution z/ρ to (LP2) of equal objective value (that is, OPT_{LP}/ρ), in polynomial time. (LP2) is now in pipage rounding form - the bipartite graph $(U \cup W, E)$ has vertices $u_i \in U$ for each group G_i, vertices $w_S \in W$ for each set $S \in \mathcal{S}$, and edges $u_i w_S \in E$ for each group G_i and set $S \in G_i$.

Since (LP2) can be approximately solved in poly-time, we would like to round it to an integral solution using the pipage rounding algorithm and the input z/ρ; note that since only a polynomial number of z values are non-zero, the rounding algorithm will terminate in a polynomial number of steps, and each step can be performed in poly-time. This would give an integer solution $\bar{z}$, which we then wish to show is at most a factor of $(1 - e^{-1/\rho})$ away from the optimal solution to (LP2).

Let $L(z) = \sum_e \min(1, \sum_{S \ni e} z_S)$. We will define a "nice" function $F(z)$ that is both ϵ-convex on the input z/ρ and satisfies a (modified) version of the F/L lower bound condition - instead of condition 1.8, we would like to show something stronger. Suppose that, for an optimal solution $\check{z}$, our sub-optimal solution z/ρ has the property that $F(z/\rho) \geq L(\check{z})/\alpha$ for some α. If L and F are coincident on binary inputs, then $L(\check{z}) \geq F^*$, and so this new condition would imply we have an α-approximation after pipage rounding.

We first claim that the function $F(z) = \sum_e (1 - \prod_{S \ni e}(1 - z_S))$ satisfies these conditions.

Claim 2.3. *$F(z)$ satisfies the modified F/L lower bound conditions.*

Proof. First note that for binary z, $L(z) = F(z)$. We now need to show some α exists such that for any optimal solution $\check{z}$, $F(z/\rho) \geq L(\check{z})/\alpha$.

Let n_e be the number of sets in $\mathcal{S}$ which contain e. We can use similar arguments as in the proof of Theorem 2.2 to show that, for our sub-optimal solution z/ρ,

$$\begin{aligned} 1 - \prod_{S \ni e}(1 - z_S/\rho) &\geq 1 - \left(1 - \frac{1}{\rho n_e} \min(1, \sum_{S \ni e} z_S)\right)^{n_e} \\ &\geq 1 - \left(1 - \frac{1}{\rho n_{\bar{e}}}\right)^{n_{\bar{e}}} \min(1, \sum_{S \ni e} z_S) \\ &\geq (1 - e^{-1/\rho}) \min(1, \sum_{S \ni e} z_S). \end{aligned}$$

Thus, $F(z/\rho) \geq (1 - e^{-1/\rho})L(z)$. But since $L(z) \geq OPT_{LP}$ (see the arguments made in Theorem 2.2), then $L(z) \geq L(\check{z})$, and so $F(z/\rho) \geq (1 - e^{-1/\rho})L(\check{z})$. □

Claim 2.4. *$F(z/\rho)$ is ϵ-convex.*

Proof. Since the groups G_i define a partition over $\mathcal{S}$, the bipartite graph used during the rounding must be a forest, with each tree having height 1. This implies that in each step of the pipage rounding algorithm, the chosen R must be a path of length at most 2. Rewriting $F((z/\rho)(\epsilon, R))$ as a function of ϵ, we then have either a linear or quadratic function. Since for all e, $z_e/\rho \in [0, 1]$, then this quadratic will have a non-negative main term, and so $F(z/\rho)$ is ϵ-convex. □

Given this, we can now apply pipage rounding on the fractional solution z/ρ and using the function F to guide the algorithm. This yields a deterministic $\left(\frac{e^{1/\rho}}{e^{1/\rho}-1}\right)$-approximation to the MCG problem.

Chapter 3

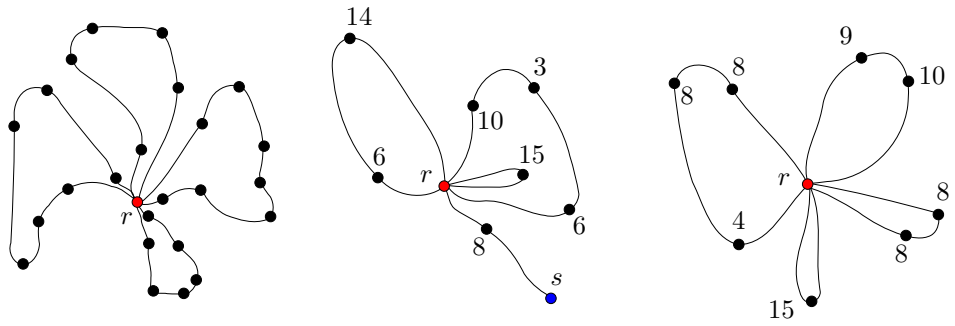
Approximating Capacitated Latency Problems

In this chapter we consider a collection of vehicle routing problems, showing how to approximate them within a constant factor of the optimum. We focus specifically on some capacitated variants of the *minimum latency problem*, where the objective is to minimize the average waiting time of a client; this is in contrast to travelling salesman-type problems, where the objective is to minimize the distance the vehicle travels. The idea of *latency* nicely models the real-world concept of response time, which in many situations is a much more important metric than the distance a vehicle travels, including emergency response management, routing delivery vehicles and school buses, routing repairmen, etc. These problems are therefore also commonly known as *travelling repairmen problems* (TRP).

We specifically consider capacitated variants of the *Multi-Depot k-Travelling Repairmen Problem* (abbreviated to MD-kTRP). While the uncapacitated problem has been studied previously, the capacitated variants we consider have only been looked at from a heuristics perspective, making the results we present here the first true approximations for these problems. The approach we take is modular, which allows us to approximate more than one problem with the same basic algorithm. This in fact allows us to match the previous best result for uncapacitated MD-kTRP, with a different technique. We make use of the MCG approximations developed previously to achieve our results.

3.1 Problem Overview

We study a generalization of the previously studied latency problems, which has been recently considered by the Operations Research community. In the *Multi-Depot Capacitated k-Travelling Repairmen Problem* (MD-CkTRP), we are given a collection of clients $C = \{c_1, c_2, \ldots, c_n\}$ that need to be served by some vehicle, and a collection of k identical vehicles stationed at given depots (roots) $R = \{r_1, r_2, \ldots, r_k\}$. Each client additionally has a positive demand which needs to be satisfied, given as the function $w : C \to \mathbb{Z}^+$, and the



(a) An r-rooted flower (b) An $r \to s$ capacitated walk (c) An r-rooted capacitated flower

Figure 3.1: Examples illustrating different walk types used. For the capacitated examples, numbers above vertices are demands, and $Q = 20$.

vehicles have a fixed positive capacity Q. The objective is to find a routing of the vehicles over the symmetric metric $(R \cup C, d)$, such that every client's demand is satisfied, minimizing the average time required to service all clients, while obeying the following constraints:

1. All clients must be completely served in a single trip (called *unsplit delivery*), and
2. A vehicle can serve a total of at most Q demand between visits to its depot (the visits to the depot are called *resupplies*).

In order to discuss the problem itself and our algorithm, we first need to clarify our terminology. We will use these terms throughout our discussion to clarify the structure of the vehicle routes we will be computing; they are more clearly illustrated in figure 3.1. Unless otherwise stated, we refer to any $v \in R \cup C$ as a *vertex*.

Definition 3.1. *A **flower** is an ordered sequence of tours, which share a single common vertex r; the tours that make up the flower are called the **petals**. See figure 3.1a.*

Definition 3.2. *A **capacitated walk** is an ordered sequence of 0 or more tours rooted at a single common vertex r, followed by an additional path starting at r, where each tour/path W_i covers clients with total demand at most Q (*i.e. $Q \geq \sum_{c \in W_i} w(c)$*). Similarly, a **capacitated flower** is a flower in which each petal covers clients with total demand at most Q. See figures 3.1b and 3.1c.*

Let W be any (potentially capacitated) walk; define $d_W(u, v)$ to be the distance between vertices $u, v \in W$ along the walk W.

Definition 3.3. *Given a walk W rooted at a vertex r, the **latency** of a vertex v along W is given by $lat_v = d_W(r, v)$. The total latency of a (potentially capacitated) walk W is given by $lat_W = \sum_{c \in W} lat_c$.*

A feasible solution to the MD-CkTRP is a collection of r_i-rooted capacitated walks F_i ($1 \leq i \leq k$), such that every client belongs to exactly one such walk. Our objective is to minimize the sum $\sum_{F_i} lat_{F_i}$, or the cumulative latency of all vertices; this is equivalent to minimizing the average service time.

3.1.1 Our Results

In order to approximate MD-CkTRP, we build on and extend the techniques used previously for approximating MD-kTRP and other capacitated vehicle routing problems. Our approach is modular in a sense; our main algorithm uses an oracle as a subroutine which must conform to a specific specification, and queries this oracle to generate pieces of the vehicle routes. The main algorithm then stitches these together carefully, producing a good solution that is feasible for the original problem. With a good collection of oracles, this allows us to obtain $O(1)$-approximation algorithms for the problems we consider. We obtain the following results:

Theorem 3.4. *There is a 25.49-approximation to the unit-demand MD-Ck TRP (*i.e. $w(c) = 1$ *for all $c \in C$).*

Theorem 3.5. *There is a 42.49-approximation to the unsplit-delivery MD-Ck TRP.*

Our algorithm can be extended to solve the problem when we have non-uniform vehicle capacities with no additional loss. We can further extend the problem to include client service delays $\delta(c)$, with an additional $+0.5$ factor loss in approximation. We also obtain an 8.497-approximation when vehicles have infinite capacity, matching the current-best result by Post and Swamy [28].

Our approach requires an oracle that can approximate the following problem. Suppose we are given a set of clients C, a budget B, and a collection of r_i-rooted walks $\mathcal{W}_i$; we say these are the feasible walks for vehicle i. The *$\mathcal{W}_i$-restricted orienteering problem* is to choose a collection of walks $W_j \in \mathcal{W}_i$ that contains as many clients in C as possible, such that the sum of the lengths of W_js is at most B. We define an approximation to this problem as follows:

Definition 3.6. *A (ρ, γ)-approximation to the $\mathcal{W}_i$-restricted orienteering problem is an algorithm that finds a collection of walks of cost at most γB covering at least a $1/\rho$-fraction of the number of clients in an optimal collection of walks.*

If we create an approximation algorithm to this problem that is guaranteed to always return a *flower*, but whose ratio is still bounded against the optimal collection of walks, then we call this a *flower-approximation*. We use the collection $\mathcal{W}_i$ to generalize the notion of capacitated walks and simplify the discussion. For our purposes, we will define the

collections $\mathcal{W}_i$ with enough structure so that we can always find a flower corresponding to any collection of walks in $\mathcal{W}_i$; this makes the notion of a flower-approximation well defined.

With such an oracle available for a given definition of $\mathcal{W}_i$, we can state our main result as follows:

Theorem 3.7. *Let $\mathcal{W}_i$ be the set of all feasible r_i-rooted walks for vehicle i, and let $\mathcal{A}$ be a (ρ, γ) flower-approximation to the $\mathcal{W}_i$-restricted orienteering problem for ρ, γ both constants. Then there is an $O(1)$-approximation to the $\mathcal{W}_i$-restricted multi-depot k-TRP.*

Choosing $\mathcal{W}_i$ to be the collection of *all* r_i-rooted paths, the problem we solve with Theorem 3.7 becomes the MD-kTRP. Similarly, by choosing $\mathcal{W}_i$ to be the collection of r_i-rooted walks/tours containing at most Q clients (or client-demand), the problem them becomes the unit-demand (or unsplit-delivery) MD-CkTRP. For different choices of $\mathcal{W}_i$, approximations for other variants of TRP can be obtained (with a corresponding oracle).

We first prove Theorem 3.7 in Section 3.2, and then proceed to define oracles that yield algorithms for our various problems. We present a simple oracle for the uncapacitated problem, yielding an 8.497-approximation for MD-kTRP, in Section 3.3.1. We then give more complex oracles to prove Theorems 3.4 and 3.5 in Section 3.3.2.

3.2 A Constant Approximation

We present a combinatorial algorithm that satisfies Theorem 3.7. This result builds on the combinatorial techniques of Chekuri and Kumar for approximating the uncapactiated MD-kTRP [12]. Our approach is in fact equivalent to the LP rounding technique of Post and Swamy [28] for the uncapacitated problem they consider; the combination of our greedy algorithm and MCG LP yields the time-indexed LP they use. Our approach in a sense unifies the results of [12] and [28], and by separating the MCG LP from our greedy algorithm, our results become more easily adaptable to problems beyond latency.

3.2.1 Algorithm

The algorithm breaks the computation into several *rounds*, where for each round, we are given a budget (*i.e.* time limit) with which to use all the vehicles to cover as many new clients as possible. We can then bound the latency of the clients we covered by the sum of the budget for the round plus the budgets for all previous rounds. If we ensure that all vehicles return to the depot after each round, then the tours found for a round will be (capacitated) flowers, which we can easily stitch together with the flowers from other rounds to form the final solution.

Recall that we are given as part of the input an oracle $\mathcal{A}$, which is a (ρ, γ) flower-approximation to the $\mathcal{W}_i$-restricted orienteering problem. Let $\tau > 1$, $U \in [0, 1)$, and $b = \tau^U$

be global constants; we will pick a good value for τ later (based on ρ and γ), and choose U uniform-randomly. Let $j \geq 1$ be the index of the current round, and let C_j^u be the set of **u**ncovered clients at the *start* of round j.

We will require a subroutine $\mathcal{C}$ that can solve the following *multi-depot group orienteering problem* (MD-GOP). Suppose we are given a subset C' of clients (that still need to be visited), vehicle depots $R = \{r_1, r_2, \ldots, r_k\}$, a hard budget B, and the collections of paths $\mathcal{W}_i$. We wish to find an r_i-rooted $\mathcal{W}_i$-restricted walk of length at most B for each vehicle i, such that the collection of clients in C' that are covered by these walks is as large as possible. We will call $\mathcal{C}$ a (ρ, γ) flower-approximation if it finds a collection of $\mathcal{W}_i$-restricted flowers, one per vehicle, that are each at most γB in length and collectively cover at least a $1/\rho$-fraction of the optimal number of clients.

We can construct such a subroutine $\mathcal{C}$ as follows. Let S_W be the set of vertices in $C' \cup R$ contained in the walk W, and let $\mathcal{W}_i^B$ be the subset of $\mathcal{W}_i$ containing only walks of length at most B. Let $G_i = \{S_W : W \in \mathcal{W}_i^B\}$ be the group of all $\mathcal{W}_i$-restricted walks of length at most B. This defines a valid MCG instance; note that the groups G_i must be disjoint since every $S_W \in G_i$ must contain the root r_i, which is a vertex unique to that group. Solving this instance optimally yields a collection of $\mathcal{W}_i$-restricted walks, one per vertex and of length at most B, that collectively cover as many vertices in C' as possible. We obtain a $(\frac{e^{1/\rho}}{e^{1/\rho}-1}, \gamma)$ flower-approximation by combining our improved MCG approximation in Chapter 2 with the oracle $\mathcal{A}$.

With our algorithm $\mathcal{C}$, we can now state the algorithm for round j:

function DO-ROUND(j)

Run $\mathcal{C}(C_j^u, b\tau^j, \mathcal{A})$ with clients C_j^u, budget $b\tau^j$, and using the oracle $\mathcal{A}$.

Traverse the returned flower for each r_i in either direction, chosen uniformly at random.

$C_{j+1}^u \leftarrow$ all remaining uncovered clients from C_j^u.

end function

The overall algorithm is then to run DO-ROUND for $j = 1, 2, 3, \ldots$ until C_j^u becomes empty for some j, stitching the returned flowers together to build the final solution. Since the budget per round increases geometrically, the number of rounds required to cover everything will be polynomial in the instance size, and the solution will be feasible since every vehicle's route will be composed of a collection of $\mathcal{W}_i$-restricted flowers, which can only be stitched together at the root r_i.

3.2.2 Analysis

We now show that this algorithm is an $O(1)$-approximation, satisfying Theorem 3.7. Let OPT be some fixed optimal solution, and let O_j be the set of clients with latency at most $b\tau^j$ in OPT. Let C_j^v be the clients our algorithm has chosen to **v**isit up to the *end* of round

j (with O_j and C_j^v being the empty sets for $j \leq 0$). Recall that C_j^u is the set of clients that are still unvisited at the *start* of round j.

Lemma 3.8. $|C_j^v - C_{j-1}^v| \geq \frac{e^{1/\rho}-1}{e^{1/\rho}}|O_j - C_{j-1}^v|$.

Proof. Let $R_j = O_j - C_{j-1}^v$; these are clients that OPT could cover with latency at most $b\tau^j$ that we have not covered yet. There must therefore be a $\mathcal{W}_i$-restricted walk for each vehicle i of length at most $b\tau^j$ collectively covering the clients in R_j. This implies that our algorithm $\mathcal{C}$ can find walks covering at least $\frac{e^{1/\rho}-1}{e^{1/\rho}}|R_j|$ new clients, giving the lemma. □

Let $n_j^{OPT} = |C - O_j|$ and $n_j = |C - C_j^v|$; these are the number of clients in OPT with latency more than $b\tau^j$ and the number of clients our algorithm covers after round j, respectively. Let B_j be the budget for round j; for $j \geq 1$ this is $b\tau^j$ and for $j \leq 0$ we define it to be 0. Let $\Delta_j = B_j - B_{j-1}$ be the increase to the budget for round j.

Lemma 3.9. *For all j, we have $n_j \leq \frac{1}{e^{1/\rho}}(n_{j-1} + (e^{1/\rho} - 1)n_j^{OPT})$.*

Proof. From Lemma 3.8, we can see that $n_j \leq n_{j-1} - \frac{e^{1/\rho}-1}{e^{1/\rho}}|O_j - C_{j-1}^v|$. We obtain the lemma by observing that $|O_j| = n - n_j^{OPT}$ and $|C_{j-1}^v| = n - n_{j-1}$. □

Lemma 3.10. *In expectation over all choices of U, the latency of our solution is at most $\frac{\gamma(\tau+1)}{2(\tau-1)} \sum_{j\geq 1} n_{j-1}\Delta_j$.*

Proof. For a client c covered in round j by vehicle i, its latency in our solution is at most the sum of the lengths of all flowers chosen for vehicle i in rounds $1 \leq j' \leq j$. Note however that the last flower is traversed in a random direction, so in expectation the additional latency c incurs in round j is $1/2$ the total length of the flower picked in round j. This implies the expected latency of c is at most

$$\frac{\gamma b\tau^j}{2} + \sum_{j'=1}^{j-1} \gamma b\tau^{j'} \leq \frac{\gamma(\tau+1)}{2(\tau-1)} b\tau^j.$$

We can now upper-bound the total expected latency of all clients in our solution by

$$\frac{\gamma(\tau+1)}{2(\tau-1)} \sum_{j\geq 1} B_j(n_{j-1} - n_j) = \frac{\gamma(\tau+1)}{2(\tau-1)} \sum_{j\geq 1} n_{j-1}\Delta_j. \tag{OUR-UB}$$

We obtain the right-hand side by re-arranging the summation, noting that $\Delta_1 = B_1$ and $n_0 = n$. □

Lemma 3.11. *In expectation over all choices of U, the latency of OPT is at least $\frac{\ln \tau}{\tau-1} \sum_{j\geq 1} n_{j-1}^{OPT}\Delta_j$.*

Proof. Since n_j^{OPT} is the number of clients with latency $> b\tau^j$, we can initially lower-bound the total latency of OPT with

$$\sum_{j\geq 0} b\tau^j(n_j^{OPT} - n_{j+1}^{OPT}) = \frac{1}{\tau}\sum_{j\geq 1} n_{j-1}^{OPT}\Delta_j. \tag{3.1}$$

The left-hand side is derived by rounding the latency of each client down to the nearest $b\tau^j$.[1] The right-hand side is obtained by rearranging the summation.

We can improve this bound using the random value $b = \tau^U$: suppose client c is visited in OPT with latency $lat_c^{OPT} = d\tau^j$, where $1 \leq d < \tau$. If we choose U such that $b \leq d$, then c's latency will be rounded on the left-hand side of (OPT-LB) down to $b\tau^j$. Otherwise, $b > d$, so the latency of c will be rounded down to $b\tau^{j-1}$. Over all uniform-random choices of U, the expected latency of c using this rounding is

$$\begin{aligned}\int_0^{\log_\tau d} b\tau^j dU + \int_{\log_\tau d}^1 b\tau^{j-1} dU &= \tau^{j-1}\left(\tau \int_0^{\log_\tau d} \tau^U dU + \int_{\log_\tau d}^1 \tau^U dU\right) \\ &= d\tau^{j-1}\left(\frac{\tau-1}{\ln \tau}\right) \\ &= lat_c^{OPT} \cdot \frac{\tau-1}{\tau \ln \tau}.\end{aligned}$$

In expectation then, our rounding in (3.1) is a factor $\frac{\tau \ln \tau}{\tau - 1}$ away from OPT. Our lower-bound can therefore be improved to

$$\frac{\ln \tau}{\tau - 1} \sum_{j \geq 1} n_{j-1}^{OPT} \Delta_j. \tag{OPT-LB}$$

□

By summing Lemma 3.9 over all j, and carefully rearranging summations, we can now see that

$$\begin{aligned}\sum_{j\geq 1} \Delta_j n_{j-1} &\leq \frac{1}{e^{1/\rho}}\left(\sum_{j\geq 1} \Delta_j n_{j-2} + (e^{1/\rho}-1)\sum_{j\geq 1} \Delta_j n_{j-1}^{OPT}\right) \\ &= \frac{\tau}{e^{1/\rho}} \sum_{j\geq 1} \Delta_j n_{j-2} + \frac{(1-e^{-1/\rho})(\tau-1)}{\ln \tau} \frac{\ln \tau}{\tau - 1} \sum_{j\geq 1} \Delta_j n_{j-1}^{OPT} \\ \implies \sum_{j\geq 1} \Delta_j n_{j-1} &\leq \frac{(\tau-1)(1-e^{-1/\rho})}{\ln(\tau)(1-\tau e^{-1/\rho})} (\text{OPT-LB}) \\ \implies (\text{OUR-UB}) &\leq \frac{\gamma(\tau+1)(1-e^{-1/\rho})}{2\ln(\tau)(1-\tau e^{-1/\rho})} (\text{OPT-LB}),\end{aligned} \tag{3.2}$$

which allows us to complete the proof of Theorem 3.7.

Proof of Theorem 3.7. By (3.2), our algorithm from Section 3.2.1 is a $\frac{\gamma(\tau+1)(1-e^{-1/\rho})}{2\ln(\tau)(1-\tau e^{-1/\rho})}$-approximation for any constant $1 < \tau < e^{1/\rho}$, satisfying the requirements of the theorem. □

3.3 Oracles

Theorem 3.7 gives us a general framework with which to solve the capacitated problems we are interested in, but we require good oracles. We present three in this section: one

[1] There is an annoying detail hidden here - this lower bound only holds if no client has latency less than b in OPT. However, by scaling distances we can ensure this is always the case.

for the uncapacitated MD-kTRP, one for the unit-demand MD-CkTRP, and one for the unsplit-delivery MD-CkTRP.

Recall that we want to find a (ρ, γ)-flower approximation to the $\mathcal{W}_i$-restricted orienteering problem. This algorithm should find an r_i-rooted flower of cost at most γB covering at least a $1/\rho$-fraction of the number of clients in an optimal walk.

3.3.1 An Oracle for MD-kTRP

To start, we give a simple $(1, 2+\epsilon)$-approximate oracle for the uncapacitated MD-kTRP problem, where $\mathcal{W}_i$ is *all* r_i-rooted walks. This oracle was originally described by Chaudhuri *et al.* [10] in the context of the single-vehicle travelling repairman problem, and was later used in the multi-vehicle and multi-depot settings [16, 12].

Consider the k-stroll problem: we are asked to find the shortest r, s walk that contains at least k vertices. If this optimal walk has length B, then Chaudhuri *et al.* [10] showed that a k-MST of cost at most $B + \epsilon$ can be found in polynomial time that contains both r and s. So, assume the optimal walk for our subproblem covers ℓ^{OPT} clients; such a walk is an ℓ^{OPT}-stroll of cost at most B. By using the algorithm from [10], we can find an r_i-rooted ℓ^{OPT}-MST that has cost at most $B + \epsilon$. By doubling and short-cutting this tree, we obtain a flower of cost at most $2B + \epsilon'$ covering ℓ^{OPT} clients, yielding a $(1, 2+\epsilon')$-approximation.

Combining this oracle with Theorem 3.7, we obtain the following, which matches the current-best result obtained by Post and Swamy [28].

Corollary 3.12. *There is an* 8.497*-approximation to the uncapacitated MD-kTRP, for* $\tau \approx 1.616$.

3.3.2 Oracles for Unit-Demand and Unsplit-Delivery Versions

We now look at the capacitated problems; our algorithms for each will be similar, so we describe both simultaneously. For these problems, $\mathcal{W}_i$ is the set of all r_i-rooted capacitated walks, where vehicle i is only allowed to sequentially serve at most Q client demand before it must return to r_i to resupply. It must further serve a client's entire demand at once; a client may not be served over multiple trips or by more than one vehicle. We wish to find a flower that has this property; if we find one then it must also be in $\mathcal{W}_i$.

In this section we will show how to build a $(1, 10+\epsilon)$-flower approximation algorithm for the unsplit-delivery version, and a $(1, 6+\epsilon)$-flower approximation algorithm for the unit-demand version. If the optimal capacitated walk covers ℓ^{OPT} clients, then our algorithms will find a flower covering ℓ^{OPT} clients, respecting the capacity constraint, with cost at most $(10+\epsilon)B$ and $(6+\epsilon)B$, respectively.

Preliminaries. Note that any capacitated walk of cost B can be converted into a flower

of cost at most $2B$ by doubling each edge in the final path section of the walk and short-cutting, so with a factor-2 loss we simply focus on finding such a flower. We utilize two key inequalities, which are classic results in capacitated vehicle routing.

Let W^{OPT} be some optimal r_i-rooted capacitated walk for the relevant problem, spanning clients C^{OPT} (where $|C^{OPT}| = \ell^{OPT}$), of cost at most B. Let F^{OPT} be the corresponding flower of cost at most $2B$. Let TSP^{OPT} be an optimal r_i-rooted TSP tour covering the clients in C^{OPT}. For convenience, define $d(W)$ to be the metric cost of the walk/tour/flower W.

Lemma 3.13. *The following inequalities hold for $w(c) \in \mathbb{Z}^+$:*

$$d(TSP^{OPT}) \leq d(F^{OPT}) \leq 2B \tag{3.3}$$

$$\frac{2}{Q} \sum_{c \in C^{OPT}} d(c, r_i) w(c) \leq d(F^{OPT}) \leq 2B. \tag{3.4}$$

Proof. (3.3) holds trivially, since any optimal capacitated walk can be no cheaper than an optimal uncapacitated walk that visits the same clients.

To show (3.4), we first prove the following weaker version holds, and then refine the argument:

$$\frac{2}{Q} \sum_{c \in C^{OPT}} d(c, r_i) \leq d(F^{OPT}) \leq 2B \tag{3.5}$$

This inequality was first shown by Haimovich and Rinnooy Kan [21]. Let C_j denote the clients covered by the j^{th} petal in F^{OPT}. It follows that

$$d(C_j) \geq 2 \max_{c \in C_j} d(c, r_i) \geq 2 \frac{\sum_{c \in C_j} d(c, r_i)}{|C_j|} \geq \frac{2}{Q} \sum_{c \in C_j} d(c, r_i).$$

This implies the inequality, since

$$d(F^{OPT}) = \sum_j d(C_j) \geq \frac{2}{Q} \sum_{c \in C^{OPT}} d(c, r_i).$$

We can now show (3.4) holds, by mimicking the arguments of Altinkemer and Gavish [2]. Consider a new unit-demand instance containing the vertices r_i and $w(c)$ copies of each client $c \in C^{OPT}$. Let Ω_c be the set of the $w(c)$ copies of c in G'. We will define a new metric d' as follows: for any $v \in \Omega_c$, $d'(r_i, v) = d(r_i, c)$; for any $u, v \in \Omega_c$, $d'(u, v) = 0$; and for any $u \in \Omega_c$, $v \in \Omega_{c'}$, $d'(u, v) = d(c, c')$.

We can covert F^{OPT} into a feasible flower of no greater cost in this new unit-demand instance that covers $\sum_{c \in C^{OPT}} w(c)$ vertices, by simply following the original flower's route and visiting all copies of client c when that client is visited in the original flower. Applying inequality (3.5) and collapsing the summation yields (3.4). □

Algorithm. Our algorithm in essence finds an ℓ^{OPT}-MST, converts it into a tour, and then sub-divides this tour carefully to create the petals of a flower; we then return this flower. To

determine the approximation ratio however, we first need to transform the optimal solution into a "nice" form, which then allows us to bound the cost of the solution we compute.

Consider the input metric as an undirected graph G. For any given set of clients C', define $LB(C') = \frac{1}{Q}\sum_{c\in C'} d(r_i, c)w(c)$; then by inequality (3.4), $LB(C^{OPT}) \leq B$. For all clients c, add a new client c_t to the graph (we call these *terminals*), and an edge (c, c_t) of cost $\frac{1}{Q}d(r_i, c)w(c)$ for the unit-demand problem, or $\frac{2}{Q}d(r_i, c)w(c)$ for the unsplit-delivery problem. We call this modified graph G'.

Note that the walk W^{OPT} can be converted into a walk on G' that visits both C^{OPT} and the corresponding terminal clients; the cost of such a walk is at most $B+2LB(C^{OPT}) \leq 3B$ for the unit-demand problem, or $B + 4LB(C^{OPT}) \leq 5B$ for the unsplit-delivery problem. Further, by skipping over the clients in C^{OPT} we can obtain a walk of no greater cost that only covers ℓ^{OPT} terminals. Take the shortest-path metric completion of G', and remove the non-terminal clients - we can now use the ℓ-MST algorithm of Chaudhuri et at [10] to find a tree covering ℓ^{OPT} terminals with cost at most $3B + \epsilon$ for the unit-demand problem, or $5B + \epsilon$ for the unsplit-delivery problem.[2]

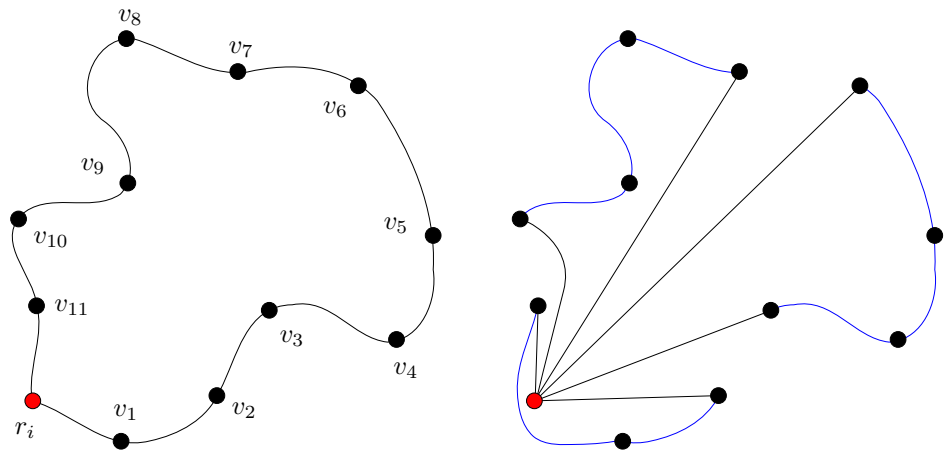
Double this tree and shortcut to produce a tour T. Note that this tour is also a valid tour in G' that still visits ℓ^{OPT} clients, and further note that if we shortcut past all terminals, then this tour will have cost at most $6B-2LB(T)$ for the unit-demand problem, or $10B-4LB(T)$ for the unsplit-delivery problem. Thus, we have an upper bound on the cost of this tour in the original metric d.

We now sub-divide this tour and build a capacitated flower, without increasing the cost too much. For the unit-demand problem, number the vertices of T in the order they are visited. Pick a random offset $R \in \mathbb{Z}_Q$. Walk along the tour starting at vertex R, shortcutting past r_i, and cut away a strip of the tour every Q vertices. For each strip, add an edge back to r_i at each end - these form the petals of our flower. The expected cost of these extra r_i-edges is $2LB(T)$, so some offset R exists such that we pay at most this extra amount. We can therefore always divide T into petals, where each petal covers at most Q demand, with total cost at most $6B$. This flower is a feasible solution to the unit-demand problem, giving a $(1, 6 + \epsilon)$-approximation. See figure 3.2a and 3.2b for an illustration of this procedure.

For the unsplit-delivery problem, we need to perform some extra steps. Like in the proof of inequality (3.4), consider building the unit-demand graph G'', where a client c is duplicated $w(c)$ times. The tour T can be mapped to an equivalent tour in G'' that covers $\sum c \in Tw(c)$ clients. As before, number these vertices, pick R randomly, and split this tour into petals. Each petal covers at most Q demand, however a client may be served over more than one petal - we need to convert this solution back into an unsplit-delivery solution.

Since we focus on unsplit-delivery, we can assume that $w(c) \leq Q$ - this implies a client

[2] We temporarily ignore the ϵ for clarity.



(a) Initial tour, before splitting into petals

(b) After subdivision and conversion into a flower, strips are in blue ($R = 3$)

Figure 3.2: Unit-demand example ($Q = 4$) of converting a tour into a flower, by breaking the tour into strips, shortcutting past r_i, and adding edges back to r_i.

c can only be served in at most 2 consecutive petals. If this is the case, we can remove c from both petals and cover it separately in its own petal. The cost of these additional petals is bounded by $2LB(T)$, so the total cost of the flower is at most $10B$. This flower is now feasible in the original metric for the problem, giving a $(1, 10+\epsilon)$-approximation. Note that there is a simple poly-time algorithm for splitting T up in this way without explicitly building G'' or trying more than ℓ^{OPT} values of R, so our oracles run in poly-time. **Results.**

Combining the above oracles with Theorem 3.7, we obtain the following proofs to Theorems 3.4 and 3.5.

Proof of Theorem 3.4. Choosing $\tau \approx 1.616$ and using the $(1, 6+\epsilon)$-approximate oracle for the unit-demand problem, we obtain a 25.49-approximation to the unit-demand MC-CkTRP using Theorem 3.7. □

Proof of Theorem 3.5. Choosing $\tau \approx 1.616$ and using the $(1, 10+\epsilon)$-approximate oracle for the unsplit-delivery problem, we obtain a 42.49-approximation to the unsplit-delivery MD-CkTRP using Theorem 3.7. □

3.4 Extensions

We can also consider two extensions to our problems - non-uniform vehicle capacities and service delays. In the first case, suppose instead of vehicle i having capacity Q, it had capacity Q_i. We then adjust the definition of $\mathcal{W}_i$ to be all walks requiring capacity at most Q_i. The oracles we presented have the same approximation ratio regardless of the value of

Q_i, and Theorem 3.7 does not deal with Q_i, so we obtain identical results in the non-uniform capacity case.

For the service-delay problem, suppose each client c has a service time $\delta(c) \geq 0$ (assume for the roots, $\delta(r_i) = 0$), which adds to the time a vehicle must spend as it delivers its load to c. The latency of c and all subsequent clients served by the vehicle then increases by $\delta(c)$; we still wish to minimize the cumulative latency of all clients. We modify this problem into an instance without service delays as follows. Define a new metric $d'(u, v) = d(u, v) + \frac{\delta(u)+\delta(v)}{2}$, and use this metric in place of d. Note that an optimum solution to this instance will have cumulative latency no more than the optimum solution to the original instance.

Approximately solve this instance, using whichever algorithm is appropriate, and call the computed flower for vehicle i F_i. Suppose in this solution, client c has latency $lat'_c = d'_{F_i}(r_i, c)$. We can map this solution back onto the original metric; ignoring service delays, client c will then have latency $lat_c = d_{F_i}(r_i, c) = d'_{F_i}(r_i, c) - \sum_{c' \text{ visited before } c} \delta(c') - \delta(c)/2$. Thus, when service delays are included, c will have latency $lat_c = d'_{F_i}(r_i, c) + \delta(c)/2$. Since $\sum_c \delta(c)$ is a lower bound on the optimum solution cost, with an α-approximat ion to the problem without service delays, we obtain an $(\alpha + 0.5)$-approximation when service delays are added.

Chapter 4

Towards a Better Approximation for Deadline Orienteering

In this chapter we focus on a different vehicle routing problem known as *deadline-TSP*, or more recently, as *deadline orienteering*. In this problem, we are given two locations r, s that we wish to travel between, and deadlines $D(v)$ on all vertices. We wish to find an $r - s$ path that reaches as many vertices as possible at a time before their deadlines.

This problem is a generalization of the standard orienteering problem; in orienteering, each vertex has a deadline $D(v)$ equal to the overall budget B. The deadline orienteering problem has been well-studied in the Operations Research community, which has produced many good heuristic and exact solutions for the problem. However, while we have good approximation algorithms for orienteering, only $O(\log n)$-approximations are known for the deadline version [3, 11].

The lack of a sub-logarithmic approximation is surprising, as the problem does not appear to be significantly more difficult than other undirected vehicle routing problems. Although we do not find such an approximation in this chapter, we will show more concretely why a better approximation likely exists, and give approximations for special cases of the problem, including relevant versions of the related and well-studied *school-bus problem*.

4.1 Problem Overview

The deadline orienteering problem, more formally stated, is the following. We are given a symmetric metric $d(u, v)$, non-negative deadlines $D(v)$ for each vertex v, and a source-sink pair r, s in the metric. Without loss of generality, we assume distances and deadlines are integral. The goal is to find a simple path P from r to s of length at most $D(s)$, containing as many vertices as possible, such that for each vertex v along the path, $d_P(r, v) \leq D(v)$ (recall that $d_P(u, v)$ is the distance along the path P from u to v). Another interesting and

related variation of the problem is called *orienteering with time-windows*; in this version, each vertex also has an (integral) release time $R(v)$, and a vertex v may only be included in a path P if $R(v) \leq d_P(r, v) \leq D(v)$.

A similar problem to deadline orienteering which we consider is the *regret vehicle routing problem* (RVRP) or *school-bus problem*: instead of a deadline, a vertex v has *regret* $Reg_P(v)$ along path P if it is visited by P at time $d(r, v) + Reg_P(v)$. The objective is (typically) to find a minimal collection of r-rooted paths that visit all vertices such that each vertex has regret at most a given bound B. We consider maximization versions of the problem that relate to deadline orienteering: we wish to find a single path that covers as many vertices as possible, subject to a constraint on the regrets (note that deadline orienteering is a version of this problem; vertex v must be visited with regret at most $D(v) - d(r, v)$).

The exact problems we consider can be formally stated as follows. Let $|P|$ be the number of vertices reached by path P. The first problem, which we call the Max-Average-RVRP, is to find a path P rooted at r such that $|P|$ is maximized while ensuring $\sum_{v \in P} Reg_P(v) \leq \gamma |P|$; *i.e.* ensure the average regret is within some bound γ. The second problem, which we call the Max-Bounded-RVRP, is to similarly find a path P rooted at r such that $|P|$ is maximized, but instead ensure that for each vertex $v \in P$, $Reg_P(v) \leq B(v)$ ($B(v)$ being the regret bound for v). While this second problem is essentially deadline orienteering, we give an approximation that violates $B(v)$ by a constant factor, which is typically much smaller than $D(v)$, and so may be a more useful result.

4.1.1 Our Results

We present a few different results in this chapter, that provide further insight into the deadline and time-window problems. We first consider the Max-Average-RVRP and Max-Bounded-RVRP, obtaining the following results:

Theorem 4.1. *There is an* $(8 + \epsilon)$*-approximation to the* Max-Average-RVRP*, which finds a path covering at least a* $\frac{1}{8+\epsilon}$ *fraction of the optimal number of vertices.*

Theorem 4.2. *There is a* $(2.542, 4)$*-approximation to the* Max-Bounded-RVRP *running in quasi-polynomial time when regrets* $B(v)$ *are poly-bounded, which finds a path covering at least a* $\frac{1}{2.542+\epsilon}$ *fraction of the optimal number of vertices and violating regret bounds* $B(v)$ *by at most a factor of* 4*.*

We then consider the deadline orienteering problem itself. We first show the following result:

Theorem 4.3. *There is a polynomial-time algorithm to solve deadline orienteering and orienteering with time-windows when the input graph is a DAG.*

For our other results, we work with a slightly more restricted version of the deadline orienteering problem: we assume that the input graph/metric has poly-bounded edge lengths (and thus, without loss of generality, has poly-bounded deadlines). This restriction is irritating, and it is not obvious how to work around it, but the results we obtain are interesting nonetheless:

Theorem 4.4. *There is an $O(1)$-approximation to deadline orienteering on a general metric with poly-bounded edge lengths running in sub-exponential time.*

The algorithm presented to prove Theorem 4.4 is impractical, but its existence does imply the following hardness result:

Corollary 4.5. *No $\omega(1)$-hardness result for the deadline orienteering problem with poly-bounded edge lengths is possible under the assumption $\mathbf{P} \neq \mathbf{NP}$, unless the Exponential Time Hypothesis is false.*

This gives a strong indication that a constant approximation can be found for the original deadline orienteering problem. We give proofs for Theorems 4.1, 4.2, and 4.3 in sections 4.2, 4.3, and 4.4 respectively. In section 4.5, we prove Theorem 4.4 and Corollary 4.5.

4.2 An $(8+\epsilon)$-Approximation for Max-Average-RVRP

In this section, we present an $(8+\epsilon)$-approximation for the Max-Average-RVRP. Recall that in this problem, we are given a parameter γ, and must find an r-rooted path P where $\sum_{v \in P} Reg_P(v) \leq \gamma |P|$. We wish to maximize $|P|$; our approximation will find a path that covers at least a $\frac{1}{8+\epsilon}$ fraction of the optimal number of vertices.

A subroutine. Before we can discuss the algorithm, however, we need to first describe a key subroutine, which we call the Max-Uniform-RVRP. Suppose that we are given a single regret bound B for all vertices, and our objective is to find an r-rooted path P where $Reg_P(v) \leq B$ for all $v \in P$. We again wish to maximize $|P|$. We claim the following:

Lemma 4.6. *An α-approximation to the point-to-point orienteering problem yields an α-approximation to* Max-Uniform-RVRP*.*

To show this, we consider some fundamental facts about regret. Define the *regret metric* d^{Reg} as $d^{Reg}(u,v) = d(r,u) + d(u,v) - d(r,v)$. For notational convenience, let $d(e)$ be the cost of edge e, and $d(P) = \sum_{e \in P} d(e)$. The following facts were shown by Friggstad and Swamy [19]:

Fact 4.7. *The value $d^{Reg}(u,v)$ is non-negative for any u, v, and d^{Reg} satisfies the triangle inequality: for any u, v, w, $d^{Reg}(u,w) \leq d^{Reg}(u,v) + d^{Reg}(v,w)$.*

Fact 4.8. *For any u, v path P, $d^{Reg}(P) = \sum_{e \in P} d^{Reg}(e) = d(r,u) + d(P) - d(r,v)$.*

Fact 4.9. *For any r-rooted path P, regrets do not decrease when moving away from r.*

Using these facts, we can prove Lemma 4.6.

Proof of Lemma 4.6. Assume we know the optimal endpoint s (or guess it). Fact 4.9 implies that any $r - s$ path we choose that reaches s at time at most $d(r,s) + B$ will reach all other vertices v along the path at time at most $d(r,v) + B$. Thus, any $r - s$ orienteering path of length at most $d(r,s) + B$ is a feasible solution, and since the optimal solution is an orienteering path, an α-approximate orienteering solution is an α-approximate solution to Max-Uniform-RVRP. □

Note that the point-to-point version of Max-Uniform-RVRP can be solved in the same way, without the guessing of the endpoint s.

The algorithm. Given this subroutine, the remainder of the algorithm is straight-forward. Let OPT be an optimal path. There can be at most $|OPT|/2$ vertices in this path with regret more than 2γ by an averaging argument, so suppose we shortcut past these vertices - this does not increase the regret of the remaining vertices. Of what remains, at least half of the vertices either have regret at most γ, or have regret between γ and 2γ.

In the latter case, since regret is monotonic, let v be the first vertex with regret at least γ. Note that by shortcutting past all vertices before v, the regret of v will become 0, and the regret of all vertices following v will thus be reduced by at least γ. In both cases then, we can use our approximation for the Max-Uniform-RVRP to approximate the optimal path restricted to these vertices. This implies that any approximate path we find with Lemma 4.6 will cover at least a 4α-fraction of the optimal number of vertices. There is currently a $(2+\epsilon)$-approximation for orienteering due to Chekuri, Korula and Pál [11], yielding an $(8+\epsilon)$-approximation for Max-Average-RVRP. This proves Theorem 4.1.

4.3 A $(2.542, 4)$-Approximation for Max-Bounded-RVRP

In this section we consider the Max-Bounded-RVRP. Recall that we are given vertex regret bounds $B(v)$, and wish to find an r-rooted path P where for all vertices $v \in P$, $Reg_P(v) \leq B(v)$. We wish to maximize $|P|$. Note that a deadline orienteering instance can be cast as an instance of this problem (*i.e.* with $B(v) = D(v) - d(r,v)$). Let $B = \max_v B(v)$; we will give an algorithm with running time of $n^{O(\log B)}$ that is an (α, β) bi-criteria approximation. That is, our algorithm will find a solution that reaches at least an α fraction of the optimal number of vertices, and each reached vertex will have regret at most $\beta B(v)$. The running time of this algorithm will be quasi-polynomial if B is poly-bounded.

Let OPT be an optimal path, and let v_i be the last vertex in OPT with regret at most 2^i. Suppose we insert a delay into OPT so that vertex v_i is visited *exactly* at time 2^i; this increases the regret of v_i and all subsequent vertices in OPT by at most 2^{i-1}. If we do this for all i, then the vertices between v_i and v_{i+1} will have their regret increased by at most 2^i - since regret is monotonic (fact 4.9), then this implies the regrets of all vertices will increase by at most a factor of 2.

Using this idea, suppose we knew the vertices v_i in OPT; we will assume v_i is visited exactly at time 2^i. Note that the vertices covered by OPT between v_i and v_{i+1} have regret bounds at least 2^i, and are visited with regret at most 2^{i+1}. Suppose we doubled the regret bounds on these vertices; this subpath of OPT is now a feasible solution to the following Max-Uniform-RVRP instance: find a $v_i \to v_{i+1}$ path over vertices with regret bound at least 2^{i+1}, where all vertices must have regret at most 2^i.

To solve the entire Max-Bounded-RVRP problem, assuming we know the vertices v_i in OPT, we can construct the following MCG instance, which we can approximate using our algorithm from Chapter 2. Let P_i be a path from v_i to v_{i+1} of length at most $d(r, v_{i+1}) - d(r, v_i) + 2^i$ that visits vertices with $2B(v) \geq 2^i$, and let S_{W^i} be the set of vertices in this path. Let G_i be the group of all S_{W^i}'s; note that since every W_i must contain both v_i and v_{i+1}, these groups will be disjoint. This is therefore a valid MCG instance, and can be approximated using our algorithm from Chapter 2 for MCG and our algorithm for Max-Uniform-RVRP as an oracle.

Our algorithm for Max-Uniform-RVRP will cover at least a $\frac{1}{2+\epsilon}$ fraction of the optimal number of vertices for a given sub-instance; combining this with our approximation in Chapter 2 for MCG yields an algorithm that covers at least a $\frac{e^{1/(2+\epsilon)}-1}{e^{1/(2+\epsilon)}} \geq \frac{1}{2.542}$ fraction of the optimal number of vertices for the complete MCG instance. The MCG instance we constructed may violate the original regret bounds of vertices by a factor of 4, since we ensure vertices are visited at time exactly 2^i, and by our definition of the groups. We thus have a $(2.542, 4)$-approximation, that runs in polytime.

Since we do not know what the vertices v_i in OPT are, we can try all possible combinations. We assume the regret bounds $B(v)$ are poly-bounded, so there can be at most $\log_2 B(v) \leq O(\log n)$ such vertices v_i in OPT, and so trying all possible $O(n^{\log n})$ combinations of $O(\log n)$ vertices will for sure find the optimal set. Running our above algorithm with each v_i combination must then find a solution that is a $(2.542, 4)$-approximation of OPT. The running time of the entire algorithm is then $O(n^{\log n})$, which is quasi-polynomial. Improving the running time would require avoiding this guessing step, but it is not obvious how to do this.

4.4 Solving Deadline Orienteering on a DAG

In this section, we present a simple dynamic programming algorithm to solve the deadline orienteering problem when the input graph is a DAG. Recall that in the r, s deadline orienteering problem, we must find a path P from r to s maximizing the number of vertices we visit by their deadlines (*i.e.* $v \in P$ is only counted as visited if $d_P(r, v) \leq D(v)$).

Our dynamic program is as follows. Define $DP(v, i)$ to be the minimum cost of an $r \to v$ path if the path must reach at least i clients by their deadlines. Since we are not working on a metric, this path may visit a client after its deadline (including v); in this case we do not count that client as being visited by this path. The fact that the graph is a DAG is important - any r, s path cannot reach the same vertex twice, and so can never be double-counted. Thus, our DP is valid.

Let $N^{in}(v)$ be the set of vertices v' where $v' \to v$ is an edge in the DAG, and let $d(v', v)$ be the cost of the edge $v'v$. Define $DP(v, i)$ as follows:

$$DP(v,i) = \begin{cases} 0 & \text{if } v = r \text{ and } i \leq 1, \\ \infty & \text{if } (v = r \text{ and } i > 1) \text{ or } N^{in}(v) = \emptyset, \\ \Delta(v, i-1) & \text{if } \Delta(v, i-1) \leq D(v), \\ \Delta(v,i) & \text{otherwise,} \end{cases}$$

where $\Delta(v, i) = \min_{v' \in N^{in}(v)} DP(v', i) + d(v', v)$. This DP can be extended to handle the time-windows problem as well, by altering the definition of Δ to allow for waiting at a vertex until its release time: $\Delta^{tw}(v, i) = \max\{R(v), \Delta(v, i)\}$.

The maximum number of vertices that can be visited by their deadlines on a path from r to s is the largest value of i for which $DP(s, i) \leq \infty$; the corresponding path can be recovered using standard dynamic programming techniques. Computing all entries in the dynamic programming table requires $O(n^3)$-time, so this algorithm runs in polynomial time.

4.5 An $O(1)$-Approximation for Deadline Orienteering

We now present a constant approximation to the deadline orienteering problem on a general metric, which runs in sub-exponential time when the metric distances are poly-bounded. The algorithm presented here builds upon the deadline orienteering approximations of Bansal *et al.* [3] and Chekuri and Kumar [12].

Although impractical, the existence of this approximation yields Corollary 4.5, which is an interesting hardness result - we show this at the end of this section. A similar result holds for the time-windows variant, using the reduction of Bansal *et al.* [3] from orienteering with time-windows to deadline orienteering. While this does not give similar hardness results for the general problem, it strongly suggests a better approximation to both problems exist.

4.5.1 Preliminaries

Let OPT be some optimal solution to the deadline orienteering instance. We define $t_{OPT}(v)$ to be the time at which vertex v is visited by OPT. The following concepts were first defined in [3], which we reuse and extend here. These concepts are more natural if you consider OPT plotted on a deadline vs. time graph as in figures 4.1a and 4.1b, where each vertex $v \in OPT$ is plotted at the point $(D(v), t_{OPT}(v))$.

Definition 4.10. *A **minimal vertex** with respect to OPT is a vertex v where for any other vertex u, $t_{OPT}(u) \leq t_{OPT}(v)$ or $D(u) \geq D(v)$.*

That is, any vertex that appears in OPT after v must have a deadline at least that of v (or in figure 4.1a, there can be no vertex above and to the left of v). Let M_{OPT} be the collection of minimal vertices in OPT.

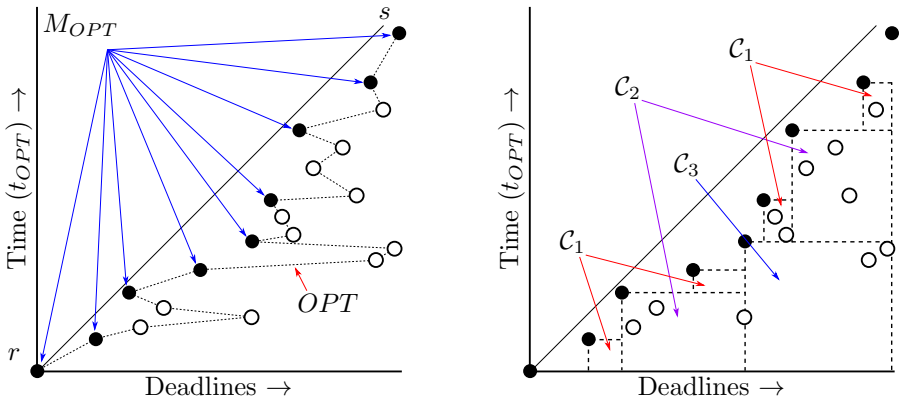
Definition 4.11. *Let u, v, w be minimal vertices. The **rectangle** $R(u, v, w)$ is the collection of vertices $\{v' \in OPT : t_{OPT}(u) \leq t_{OPT}(v') \leq t_{OPT}(v)$ and $D(v) \leq D(v') \leq D(w)\}$. We define v to be the **corner** of this rectangle.*

A side-effect of this definition is that every vertex in $R(u, v, w)$ is visited before v and has deadline after v (in figure 4.1b, every vertex in $R(u, v, w)$ is below and to the right of v); this implies that if we restrict and shortcut OPT to cover only vertices in $R(u, v, w)$, we obtain a valid orienteering tour with budget $D(v)$. This property will be very useful later on.

Definition 4.12. *$R(u, v, w)$ and $R(u', v', w')$ are **disjoint** if $t_{OPT}(u') \geq t_{OPT}(v)$, and $t_{OPT}(v') \geq t_{OPT}(w)$ (or vise versa).*

Note that if the intersection of two disjoint rectangles is non-empty, any vertex v'' in the intersection will have the property that $t_{OPT}(v'') = t_{OPT}(v') = t_{OPT}(v)$. Thus, one rectangle will be a subset of the other. A consequence of this is that if two rectangles $R(u, v, w)$ and $R(u', v', w')$ are disjoint, then a feasible solution can never visit a vertex in $R(u, v, w)$, then one in $R(u', v', w')$, and then one in $R(u, v, w)$ again (or vise versa). This property means that we can consider the two rectangles as separate instances of the problem, and the solutions can never overlap.

Suppose we knew the collection of minimal vertices M_{OPT}, and further suppose we could find a collection of disjoint rectangles $\mathcal{C} = \{R_1, R_2, \dots\}$, where the union of these rectangles contains a good fraction of the vertices in OPT. We could then find point-to-point orienteering paths in each rectangle and join them together, and achieve a good approximation to the deadline orienteering problem. This is the idea of the original Bansal *et al.* algorithm [3]; they first show how to find such a collection $\mathcal{C}$ given M_{OPT}, and then use a dynamic programming routine to try all reasonable sets M_{OPT}. They achieve an



(a) The path OPT and corresponding M_{OPT}, with vertex v placed at point $(D(v), t_{OPT}(v))$

(b) The family $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ constructed using (4.1)

Figure 4.1: Plots of OPT on the deadline-time plane illustrate key concepts used.

$O(\log n)$-approximation in this way, and we will describe this algorithm in more detail in Section 4.5.2.

We will take things a step further in Section 4.5.3, by relaxing our requirements on $\mathcal{C}$. Suppose that $\mathcal{C}$ can be partitioned into *groups* of rectangles, where two rectangles may be in different groups only if they are disjoint from each other. Using the properties of rectangles, we can show that OPT restricted to any group is also a valid deadline orienteering path that has a fixed number of distinct deadlines (bounded by the size of the group).

Suppose that the size of each group in $\mathcal{C}$ is bounded by $f(n)$. Using the deadline orienteering algorithm of Chekuri and Kumar [12], we can obtain a constant approximation of the optimal orienteering path in each group in time $(n\Delta)^{f(n)}$, where Δ is the maximum edge length in the graph (we will assume this is poly-bounded). We will show how to ensure that $f(n)$ is small enough such that the running time for each group becomes sub-exponential, while ensuring $\mathcal{C}$ contains a constant fraction of the vertices in OPT. Applying the remainder of the Bansal *et al.* algorithm will then yield a complete constant approximation in sub-exponential time.

4.5.2 A Simple $O(\log n)$-Approximation

We now present an $O(\log n)$-approximation that is equivalent to the algorithm of Bansal *et al.* [3]. Assume for now that we know the set of minimal vertices M_{OPT}. Sort these vertices by increasing t_{OPT}, and let $M_{OPT}(i)$ denote the ith vertex in this sorted order. Without loss of generality, we assume that r and s are the first (0-th) and last vertices in this sorted order, that $D(r) = 0$, and that all vertices have deadline at most $D(s)$. We will abuse our notation slightly to simplify things; when clear from the context, we will use an index i to refer to

the minimal vertex $M_{OPT}(i)$, and denote the rectangle $R(M_{OPT}(i), M_{OPT}(j), M_{OPT}(k))$ by $R(i, j, k)$. We will always be using minimal vertices to define our rectangles, so this latter simplification is not ambiguous.

We first define a family of collections of rectangles $\mathcal{F} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{\log n}\}$. Let

$$\mathcal{C}_i = \{R(j \cdot 2^i, j \cdot 2^i + 2^{i-1}, (j+1) \cdot 2^i) : 0 \leq j < \frac{|M_{OPT}|}{2^i}\}. \tag{4.1}$$

We claim the groups in each $\mathcal{C}_i$ have size 1 (i.e. all rectangles in each collection are pairwise disjoint), and where all vertices in OPT are contained in some rectangle in some $\mathcal{C}_i$. These properties can be seen more clearly in figure 4.1b, but we formally prove them here.

Claim 4.13. *All rectangles within the collection $\mathcal{C}_i$ are pairwise disjoint.*

Proof. Suppose two rectangles $R_j = R(j \cdot 2^i, j \cdot 2^i + 2^{i-1}, (j+1) \cdot 2^i)$ and $R_{j'} = R(j' \cdot 2^i, j' \cdot 2^i + 2^{i-1}, (j'+1) \cdot 2^i)$ in $\mathcal{C}_i$, $j < j'$, exist that are not disjoint. Then either $t_{OPT}(j' \cdot 2^i) < t_{OPT}(j \cdot 2^i + 2^{i-1})$, or $t_{OPT}(j' \cdot 2^i + 2^{i-1}) < t_{OPT}((j+1) \cdot 2^i)$. In the former case, since we define the rectangles using minimal vertices, since $j < j'$, then $M_{OPT}(j \cdot 2^i + 2^{i-1})$ is not a minimal vertex by definition. Similarly, in the latter case it follows that $M_{OPT}((j+1) \cdot 2^i)$ is not a minimal vertex by definition. Thus, we have a contradiction. □

Lemma 4.14. *Every vertex $v \notin \{r, s\}$ covered by OPT is contained in some rectangle in some collection in $\mathcal{F}$.*

Proof. Since r, s are the first/last vertices in M_{OPT}, any vertex v covered by OPT must be covered in the time range $t_{OPT}(r) \leq t_{OPT}(v) \leq t_{OPT}(s)$, and by assumption have deadline $D(r) \leq D(v) \leq D(s)$. This leads to two possible cases:

Case 1: v is a minimal vertex; suppose $v = M_{OPT}(k)$ (since $v \neq r$, $k \geq 1$). Note that any integer $k \geq 1$ can be written as $j \cdot 2^i + 2^{i-1}$ for $j \geq 0, i \geq 1$; let i be the largest integer at least 1 such that $k/2^{i-1}$ is an integer, and let $j = (k/2^{i-1} - 1)/2$. This is a corner of a rectangle in $\mathcal{C}_i$ by definition, so the lemma holds.

Case 2: v is not a minimal vertex. This implies we can find four minimal vertices in M_{OPT} with indices $k, k+1, \ell, \ell+1$ such that $t_{OPT}(k) \leq t_{OPT}(v) \leq t_{OPT}(k+1)$ and $D(\ell) \leq D(v) \leq D(\ell+1)$ (this additionally implies that $k < \ell$). We claim that an integer solution $i \geq 1, j \geq 0$ exists to the following set of inequalities:

$$j \cdot 2^i \leq k \tag{4.2}$$

$$j \cdot 2^i + 2^{i-1} \geq k + 1 \tag{4.3}$$

$$j \cdot 2^i + 2^{i-1} \leq \ell \tag{4.4}$$

$$(j+1) \cdot 2^i \geq \ell + 1 \tag{4.5}$$

If a solution does exist, then v would be contained in a rectangle in collection $\mathcal{C}_i$ by definition, proving the lemma.

Consider the following algorithm to find i, j. Choose the smallest i such that an integer j exists where inequalities (4.2) and (4.5) are satisfied. Such an i, j pair will always exist, since we could choose $j = 0$ and i such that $2^i \geq \ell + 1$. Note that if $j \cdot 2^i + 2^{i-1} < k + 1$, then we could instead have chosen $i - 1$, since $(2j + 1) \cdot 2^{i-1} \leq k$ and $(2j + 2) \cdot 2^{i-1} \geq \ell + 1$. A similar situation exists if $j \cdot 2^i + 2^{i-1} > \ell$, and in both cases we have a contradiction of how we chose i. Thus, we have a valid solution to the inequalities, proving the lemma. □

To obtain an $O(\log n)$-approximation given M_{OPT}, we can do the following. Pick a collection $\mathcal{C}_i \in \mathcal{F}$ that covers at least a $\frac{1}{\log n}$-fraction of the vertices in OPT; one must exist by the above arguments. For each rectangle $R(u, v, w) \in \mathcal{C}_i$, note that v is visited last by OPT at time $t_{OPT}(v) \leq D(v)$, and has the smallest deadline of all vertices in $R(u, v, w)$, so if we assign all vertices in $R(u, v, w)$ deadline $D(v)$, OPT is still a feasible solution. Further, the $u - v$ subpath of OPT restricted to this rectangle forms a valid orienteering tour of length at most $t_{OPT}(v) - t_{OPT}(u)$. Using the last vertex m visited in the previous rectangle in $\mathcal{C}_i$ as the starting point, we can therefore find an $m - v$ orienteering path with budget $t_{OPT}(v) - t_{OPT}(m)$. One must exist that covers at least as many vertices as OPT does between m and v. Once we have these paths for each rectangle in $\mathcal{C}_i$, we can stitch the paths together to obtain a complete deadline orienteering path.

We can use dynamic programming to determine M_{OPT} (and associated t_{OPT} values) on the fly. Note that the vertices in M_{OPT} are sorted by time and deadline. We can therefore build a DP table with entries (m, t_m, v, t_v), where m is the previous minimal vertex (reached at time $t_m \leq D(m)$), and v is the next minimal vertex for the rectangle (reached at time $t_v \leq D(v)$). Each entry will store an approximate $m - v$ orienteering path covering as many vertices u as possible such that $D(u) \geq D(v)$. Note that M_{OPT} defines a valid walk though this table, so we obtain an $O(\log n)$-approximation by walking through the table and picking the best path. Since deadlines are assumed to be poly-bounded, this approximation will run in polynomial time.[1]

4.5.3 Improving to an $O(1)$-Approximation

To improve our approximation, we will define a new family $\mathcal{F}'$ with groups of bounded size, where paths in each group can be found easily. Let q be an integer parameter, to be chosen later. Let $\mathcal{C}'_i = \mathcal{C}_{iq} \cup \mathcal{C}_{iq+1} \cup \cdots \cup \mathcal{C}_{(i+1)q-1}$, and let $\mathcal{F}' = \{\mathcal{C}'_1, \mathcal{C}'_2, \ldots, \mathcal{C}'_{n/q-1}\}$. Note that $\mathcal{F}'$ contains the same set of rectangles as $\mathcal{F}$, just distributed differently between collections. Thus, lemma 4.14 still holds.

[1] With some tricks, it is possible to reduce the number of computed DP table entries down to $n^2 \log^2 D(s)$, and so deadlines do not actually need to be poly-bounded; see Bansal *et al.* [3] for details.

We claim that the size of the groups in each collection is bounded by $2^q - 1$. Recall that groups define a partitioning of $\mathcal{C}'_i$, where two rectangles may be in different groups only if they are disjoint. We will assume that groups are computed so that the number of groups is maximized; this partitioning can be found as follows. Represent each rectangle with a vertex, and draw an edge between two rectangles if they are not disjoint. The groups will be the connected components in this graph.

Lemma 4.15. *The number of rectangles in each group in $\mathcal{C}'_i$ is bounded by $2^q - 1$.*

Proof. Consider any minimal vertex $m_j = M_{OPT}(j \cdot 2^{(i+1)q})$, for some integral $j \geq 0$. Note that this vertex cannot be the corner of any rectangle in $\mathcal{C}'_i$, since no integer j' exists such that $j \cdot 2^{(i+1)q} = j' \cdot 2^{iq+k}$ for any $0 \leq k < q$. To bound the size of a group, we will show that no rectangle $R(u, v, w) \in \mathcal{C}'_i$ can have $t_{OPT}(u) < t_{OPT}(m_j)$ and $D(w) > D(m_j)$. This implies that any rectangle with $D(w) \leq D(m_j)$ will be disjoint from any other rectangle with $t_{OPT}(u) \geq t_{OPT}(m_j)$. The minimal vertices $m_0, m_1, \ldots, m_j, \ldots$ therefore define separation points between groups. The size of each group is then bounded by the number of rectangles with corners between m_j and m_{j+1}, of which there can be at most $2^q - 1$.

Suppose for some integral $j' \geq 0, 0 \leq k < q$, a rectangle $R(j' \cdot 2^{iq+k}, j' \cdot 2^{iq+k} + 2^{iq+k-1}, (j'+1) \cdot 2^{iq+k}) \in \mathcal{C}'_i$ existed with $t_{OPT}(j' \cdot 2^{iq+k}) < t_{OPT}(m_j)$ and $D((j'+1) \cdot 2^{iq+k}) > D(m_j)$. We therefore must have the following (by how the minimal vertices are ordered):

$$j' \cdot 2^{iq+k} < j \cdot 2^{(i+1)q} < (j'+1) \cdot 2^{iq+k},$$

which cannot be true for any integral $j, j' \geq 0$. Thus, no such rectangle can exist, which completes the lemma. □

We now wish to show that for any collection $\mathcal{C}'_i$, we can solve the deadline orienteering problem over that collection. Within each group, we have $2^q - 1$ rectangles where each vertex v' within rectangle $R(u, v, w)$ is visited by OPT at time $t_{OPT}(v') \leq t_{OPT}(v) \leq D(v)$. Thus, each vertex contained in the group can have its deadline rounded down to the closest $D(v)$, and OPT restricted to this group is still a feasible solution with these new deadlines. Since there at most $2^q - 1$ rectangles in each group, the number of distinct deadlines in this group is therefore also bounded by $2^q - 1$. So, for each group, a deadline orienteering path with a bounded number of deadlines exists that covers at least as many vertices as OPT.

We now make use of the constant approximation of Chekuri and Kumar [12] for the deadline orienteering problem. Recall that their algorithm runs in time polynomial in $(n\Delta)^k$, where k is the number of distinct deadlines and $\Delta = \max_{u,v} d(u, v)$. Using this, we can find a deadline orienteering path through the group given start and end points in time $n^{O(2^q)}$. Following the stitching idea from the $O(\log n)$-approximation we developed previously, we can use this subroutine to build a complete deadline orienteering path over the entire collection $\mathcal{C}'_i$, covering a $O(\frac{q}{\log n})$ fraction of the vertices of OPT. We complete the algorithm

by defining a DP for computing M_{OPT}. This is in fact identical to the previous DP, except that each entry has the form $(m, t_m, v_1, t_{v_1}, v_2, t_{v_2}, \ldots, v_{2^q-1}, t_{v_{2^q-1}})$ - we need to guess *all* minimal vertices in each group, not just the last one. This table has $n^{O(2^q)}$ entries if deadlines are poly-bounded, bringing the running time of the entire algorithm to $n^{O(2^q)}$.

We can now prove Theorem 4.4 and Corollary 4.5.

Proof of Theorem 4.4. To obtain a constant approximation, we choose $q = \log(n^\epsilon)$ for some $0 < \epsilon < 1$. This gives an approximation ratio of $O(\frac{\log n}{\log(n^\epsilon)})) = O(1/\epsilon)$. The running time of this algorithm will be $n^{O(n^\epsilon)} = 2^{\log n \cdot O(n^\epsilon)} = 2^{O(n^\epsilon \log n)}$, which is sub-exponential with $\epsilon < 1$.[2] □

Proof of Corollary 4.5. Suppose a hardness result of the following form exists: unless **P** = **NP**, there is no α-approximation for deadline orienteering with poly-bounded edge lengths, for some $\alpha = \omega(1)$. Such a hardness result will necessarily provide a polynomial-time gap-introducing reduction algorithm $\mathcal{A}$ from some **NP**-hard decision problem to the deadline orienteering problem. The gap size of this reduction must be at least α, since an α-approximation for deadline orienteering is sufficient to decide the problem.

Assuming **P** $\neq$ **NP**, the existence of $\mathcal{A}$ would disprove the Exponential Time Hypothesis for the following reason. $\mathcal{A}$ allows us to reduce in polynomial time an instance of k-SAT of size n to an instance of our deadline orienteering problem of size $O(n^c)$, for some fixed $c > 0$. Choose some fixed ϵ such that $\epsilon/c < 1$. Use our $O(1/\epsilon)$-approximation for deadline orienteering to approximate the instance; for a sufficiently large instance, this approximation ratio will be less than the gap size α since $\alpha = \omega(1)$. This is therefore sufficient to decide the k-SAT instance by the properties of $\mathcal{A}$, and requires $2^{o(n)}$ time by Theorem 4.4, disproving the Exponential Time Hypothesis. □

[2] Alternatively, by choosing $q = \log\log n$, we obtain an $O(\log n / \log\log n)$-approximation in $O(n^{\log n})$ time.

Chapter 5

Conclusions and Future Work

In this thesis, we presented new approximation algorithms for a number of vehicle routing problems, including capacitated travelling repairmen problems, deadline orienteering variants, and regret orienteering variants. The subexponential time constant approximation algorithm we presented for deadline orienteering also suggests that a poly-time constant approximation for the deadline orienteering problem should exist.

We also presented an improved approximation algorithm for the maximum coverage problem with groups, which was a crucial component in some of our algorithms. This improved approximation allowed our more general algorithm for the capacitated travelling repairmen problems we consider to match the current-best result for the uncapacitated version, but unlike the current-best result, our presented algorithm is (almost) completely de-randomized.

Other problems were considered during the course of this research that we did not find reasonable solutions for, and even some of the results we presented have limitations that are irritating. We leave these problems to a future work:

- For the MD-CkTRP, can we still find a constant approximation when deliveries to a client can be *split* over multiple trips? In this case, we say the client c has latency lat_c if its entire delivery has been received by time lat_c. The oracle we present for the unsplit-delivery case can be modified to give an approximation for the split-delivery case when the demand of each client is at most Q. However, when the demand is more than Q, the approximation ratio of our algorithm becomes unbounded. This seems to be a fundamental issue inherent to the tour-splitting approach we use.

- The $O(1)$-approximation we give for deadline orienteering relies on the fact that edge-distances and deadlines are poly-bounded; this allows us to use a time-indexed approach to solve the problem. A similar issue was encountered by Bansal *et al.* [3] in their $O(\log n)$-approximation to the problem. In their paper, they work around it by pre-computing the relevant paths, and using binary search to guess the times. With

our approach however, pre-computing the paths still requires guessing arrival times, so we cannot easily avoid this time-dependence. Finding a true polynomial-time constant approximation, even for the case of poly-bounded edge lengths and deadlines, remains a big open problem.

- The algorithm we give for Max-Bounded-RVRP is only quasi-polynomial time, because of the guessing we need to do. Finding a way to either reduce the number of guesses needed or avoid them entirely would be very nice, although avoiding the guessing entirely is probably not possible while still reducing the problem to MCG.

- Improving on the 8.497 approximation of Post and Swamy [28] for MD-kTRP is a big open problem. While we were able to show that a more combinatorial approach can achieve the same result, we still use the same basic algorithm idea, which is likely why we were not able to improve on their result.

Bibliography

[1] A. Ageev and M. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[2] K. Altinkemer and B. Gavish. Heursitics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

[3] N. Bansal, A. Blum, S. Chalwa, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. *In Proceedings of STOC*, 2004.

[4] L. Bianco, A. Mingozzi, and S. Ricciardelli. The travelling salesman problem with cumulative costs. *Networks*, 23(2):81–91, 1993.

[5] A. Blum, P. Chalasani, B. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. *In Proceedings of STOC*, pages 163–171, 1994.

[6] A. Blum, S. Chalwa, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *In Proceedings of FOCS*, 2003.

[7] A. Bock, E. Grant, J. Könemann, and L. Sanità. The school bus problem on trees. *In Proceedings of ISAAC*, pages 10–19, 2011.

[8] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

[9] B. Carr and S. Vempala. Randomized meta-rounding. *In Proceedings of STOC*, 2000.

[10] K. Chaudhuri, G. Brighten, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. *In Proceedings of FOCS*, pages 36–45, 2003.

[11] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *In Proceedings of TALG*, 8(3), 2012.

[12] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. *In Proceedings of APPROX*, pages 72–83, 2004.

[13] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 338, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[14] P. Crescenzi and L. Trevisan. On approximation scheme preserving reductibility and its applications. *In Proceedings of FSTTCS*, pages 330–341, 1994.

[15] M. Desrochers, J. Lenstra, M. Savelsbergh, and F. Soumis. Vehicle routing with time windows: optimization and approximation. *Vehicle Routing: Methods and Studies*, pages 65–84, 1988.

[16] J. Fakcharoenphol, C. Harrelson, and S. Rao. The k-traveling repairman problem. *In Proceedings of SODA*, pages 655–664, 2003.

[17] Z. Friggstad. *Approximation techniques for unsplittable flow and travelling salesman problems*. PhD thesis, University of Alberta, 2011.

[18] Z. Friggstad, M. Salavatipour, and Z. Svitkina. Asymmetric traveling salesman path and directed latency problems. *SIAM Journal on Computing*, 42(4):1596–1619, 2013.

[19] Z. Friggstad and C. Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. *In Proceedings of STOC*, pages 744–753, 2014.

[20] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorics and optimization. *Combinatorica*, 1:169–197, 1981.

[21] M. Haimovich and A. H. G Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

[22] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62:367–375, 2001.

[23] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[24] A. Lucena. Time-dependant travelling salesman problem - the deliveryman case. *Networks*, 20(6):753–763, 1990.

[25] J. Lysgaard and S. Wohlk. A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem. *European Journal of Operational Research*, 236(3):800–810, 2014.

[26] E. Minieka. The delivery man problem on a tree network. *Annals of Operations Research*, 18(1–4):261–266, 1989.

[27] V. Nagarajan and R. Ravi. The directed minimum latency problem. *In Proceedings of APPROX*, 2008.

[28] I. Post and C. Swamy. Linear-programming based approximation algorithms for multi-vehicle minimum latency problems. *In Proceedings of SODA*, pages 512–531, 2015.

[29] J. C. Rivera, H. M. Afsar, and C. Prins. A multistart iterated local search for the multitrip cumulative capacitated vehicle routing problem. *Computational Optimization and Applications*, 61(1):159–187, 2015.

[30] R. Sitters. The minimum latency problem is NP-hard for weighted trees. *In Proceedings of IPCO*, 2337:230–239, 2002.

[31] R. Sitters. Polynomial time approximation schemes for the travelling repairman and other minimum latency problems. *In Proceedings of SODA*, 2014.

[32] A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. *In Proceedings of FOCS*, 2001.

[33] V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[34] D. West. *Introduction to Graph Theory*. Prentice-Hall, 2001.

[35] D. Williamson and D. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[36] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.