University of Alberta

APPROXIMATION TECHNIQUES FOR UNSPLITTABLE FLOW AND TRAVELING SALESMEN PROBLEMS

by

Zachary Lorne Friggstad

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2011
Edmonton, Alberta

Abstract

In this thesis, we present a variety of approximation algorithms for the Unsplittable Flow on Paths problem and some Traveling Salesman problems. The main contribution to the Unsplittable Flow on Paths problem is a logarithmic approximation algorithm which is the first non-trivial approximation for general instances of the problem. The algorithm works by using dynamic programming to approximate solutions on instances that cannot be approximated well through linear programming techniques. A generalization of this algorithm provides a constant-factor approximation in sub-exponential time. We also demonstrate that certain sparse instances can be approximated within a constant factor.

The Traveling Salesman problems we consider mostly deal with finding paths in asymmetric metrics, though we do consider others. First, we demonstrate that the integrality gap of a natural linear programming relaxation for the Asymmetric Traveling Salesman Path problem is $O(\log n)$ where n is the number of nodes in the metric. We then further generalize the problem and study the problem of finding up to k paths with minimum total distance in an asymmetric metric such that the union of these paths spans all nodes. In the case that all paths are required to share a common start and end node, we demonstrate a family of bicriteria approximation algorithms that find a little more than k paths whose total cost is within some bounded ratio of the optimum value of a linear programming relaxation. These results are extended to many other variants of finding multiple paths in metrics whose union spans all nodes. However, we show that the most general case when each path has its own start and end location specified in advance cannot be approximated within any bounded ratio unless P = NP.

Finally, we formulate a linear programming relaxation for the Minimum Latency problem in asymmetric metrics and prove that the integrality gap of this relaxation is $O(\log n)$. This critically relies on the fact that the integrality gap of a natural linear programming relaxation for the Asymmetric Traveling Salesman Path problem is $O(\log n)$. This is the first sub-polynomial approximation algorithm for the problem.

Acknowledgements

It is difficult to summarize how many ways I am grateful to my supervisor and mentor, Mohammad R. Salavatipour. Of course, I thank you mostly for teaching me how to think like a researcher and for the opportunities you presented to me. However, your support also extended beyond academics and I also thank you for your patience, understanding and advice as I balanced my research with other aspects of my life.

My experience would not be nearly as complete as it was without my fellow students and post-doctoral researchers. For inspiring discussions on optimization problems, I thank Amin Jorati, Babak Behsaz, Reza Khani, Zoya Svitkina, Imran Pirwani, and Mohammad A. Safari. I also appreciate the stimulating conversations I've had with Phillip Hendersen, Jessica Enright, Barry Gergel, Roshan Sharrif, and Travis Dick.

I want to thank Nikhil Bansal and Rohit Khandekar for their help in getting me started with Unsplittable Flow problems. You helped me bootstrap my studies and I'm grateful for all of the support you've given me through this process. Additionally, I would like to thank Nitish Korula and Aline Ene for a fruitful discussion on this topic.

I would also like to thank my references Mohammad R. Salavatipour, Nikhil Bansal, Mike MacGregor, and Martin Müeller for supporting me when I was applying for jobs. For personal funding and research grants, I thank the University of Alberta, NSERC, and AITF. I also appreciate the feedback for this thesis provided by Anupam Gupta, Ryan Hayward, Jim Hoover, and Mazi Shirvani. Thank you for examining this work and for your insightful questions.

Another big part of my graduate experience was my involvement with the ACM International Collegiate Programming Competition. Howard Cheng, Martin Müeller, and Piotr Rudnicki are some of the best coaches you could ask for and I'm indebted to them for their support in and beyond the contest setting. Competing in World Finals competitions with Sumudu Fernando, Andrew Neitsch, Steven Soneff, and Kevin Waugh was a lot of fun, thank you for the terrific experiences. I'm also grateful to the contestants that followed me. Being your coach for four consecutive World Finals competitions was truly an honour.

There were many professors I interacted with during my undergraduate studies that encouraged me to pursue a graduate degree. Through independent studies and summer research, I was inspired by Amir Akbary, Howard Cheng, Abdelaziz Fellah, Hadi Kharaghani, Hua Li, and Shelly Wismath.

Thank you for all of the time you spent with me while I was beginning to understand my research interests.

I dedicate this thesis to Jenne, my wife. I am humbled by the incredible love and support you have shown me throughout my studies. My boys, Gabriel and Lucas, have also encouraged me in their own ways. I am very proud to be your father. I am also grateful to my parents Lorne and Janice and my sisters Kjersti, Jannaya, and Courtney. I would not be here today without your prayers and encouragement. Finally, I thank God for my family and my successful studies.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problems Considered

In this thesis, we consider variants of some classic problems in combinatorial optimization. Most of the problems considered in this thesis are NP-hard optimization problems, so we address this difficulty by developing approximation algorithms: algorithms that find feasible solutions (usually in polynomial time) whose value is within some bounded ratio of the optimum value. We begin by breifly introducing the problems discussed. Previous work and our contributions to these problems are discussed in later sections of this introductory chapter.

Unsplittable Flow: In the general Unsplittable Flow problem, we are given a graph $G = (V, E)$ (perhaps directed) with a non-negative capacity c_e on each edge. We are also given a set of n tasks $T = \{(s_i, t_i, d_i, p_i)\}_{i=1}^{n}$ where $s_i, t_i \in V$ are the start and end nodes of task i, $d_i \in \mathbb{R}_{\geq 0}$ is the demand of task i, and $p_i \in \mathbb{R}_{\geq 0}$ is the profit of task i. The goal is to find a subset $T' \subseteq T$ and a path P_i from s_i to t_i for each $i \in T'$ of maximum possible profit $\sum_{i \in T'} p_i$ subject to the following constraint. For each edge e, the total demand of all tasks $i \in T'$ for which e lies on P_i should not exceed c_e. The problem is general enough to capture some well-studied problems in combinatorial optimization:

- If all demands, profits, and edge capacities are 1, then the problem is the *Edge-Disjoint Paths* problem.
- If the graph consists of b parallel edges with identical capacities c between two nodes s, t, then the problem is a sort of *Bin Packing* problem. Determining if all n tasks can be routed is the same as asking if items of size $d_1, \ldots, d_n$ can be packed into b bins of capacity c.
- If the graph is a single edge, then the problem is equivalent to the NP-hard *Knapsack* problem.

In general, we will see that this problem is very hard to approximate which motivates the study of Unsplittable Flow instances over restricted graph classes. In this thesis, we concentrate mostly on instances where the underlying graph G is a simple path.

Traveling Salesman: In the classic version of the Traveling Salesman problem, we are given a complete and undirected graph $G = (V, E)$ where each edge $e \in E$ has a non-negative distance d_e. Furthermore, these distances satisfy the *triangle inequality* $d_{uv} \leq d_{uw} + d_{wv}$ for every $u, v, w \in V$. Sometimes, such graphs G are called *metrics* or *symmetric metrics*. The goal is to find a Hamiltonian cycle in G of minimum total distance. Without the triangle inequality, no approximation algorithm can guarantee a reasonable bound on the approximation ratio unless P = NP. However, with the triangle inequality the problem may be approximated within constant factors. One variant of the problem is to find a Hamiltonian path with minimum total distance. Another is to allow directed graphs that still satisfy the directed triangle inequality $d_{uv} \leq d_{uw} + d_{wv}$ for any nodes u, v, w, though it might be that $d_{uv} \neq d_{vu}$ for some nodes $u, v \in V$. Such graphs may also be called *asymmetric metrics*. Again, both of these variants have been studied very well from the perspective of approximation algorithms. In this thesis, we mainly consider variants where we want to find paths in asymmetric metrics that include all nodes. This includes finding a single Hamiltonian path in an asymmetric metric or finding up to k paths whose union includes all nodes for some specified integer k. However, we do consider some other variants that involving symmetric metrics or cycles.

Minimum Latency: Here, we are given a start node s in a metric and our goal is to find a Hamiltonian path P starting at s. Rather than minimizing the total cost of P (as in Traveling Salesman problems), the goal is to minimize the average latency of the nodes on P. Here, the latency of a single node $v \in V$ is the total cost of the edges between s and v on the path P; the time it takes to reach v when following P. The problem is also referred to as the *Traveling Repairman* problem because the goal of the repairman is to minimize the average waiting time of clients who require repairs. In this thesis, we present approximation algorithms for instances of the Minimum Latency problem in asymmetric metrics.

1.2 Notations and Preliminaries

1.2.1 Graphs

In this thesis, the term *graph* is a simple, undirected graph $G = (V, E)$ consists of a set of edges E connecting distinct pairs of nodes V. We often denote an edge $e \in E$ that connects two nodes $u, v \in V$ by uv. A *directed graph* $G = (V, A)$ consists of a set of arcs A where each arc is an ordered pair of nodes in V. As in the case of undirected graphs, we often denote an arc that connects u to v by uv. So, in undirected graphs uv and vu refer to the same edge whereas in directed graphs uv and vu are different arcs. In some cases, it is more convenient, notationally, to use (u, v) to refer to a directed arc from u to v. Most graphs in this dissertation do not contain multiple copies of an edge or arc and there are no loops. If multiple arcs or loops are used, it will be explicitly mentioned. For reference, we list a variety of graph-theoretic concepts below.

- **Topological Ordering**: A ordering of the nodes $v_1, \ldots, v_{|V|}$ in a directed graph $G = (V, A)$ is topological ordering if v_i appears before v_j for every arc $v_i v_j \in A$. Such an ordering exists and can be found efficiently if and only if G contains no cycles.

- **Strongly Connected**: A directed graph $G = (V, A)$ is strongly connected if for any pair of distinct nodes $u, v \in V$, there is a path from u to v and a path from v to u in G.

- **Weakly Connected**: A directed graph $G = (V, A)$ is weakly connected if the undirected graph $G' = (V, E)$ with $E = \{uv : uv \in A \text{ or } vu \in A\}$ is connected. Simply put, G is weakly connected if the undirected graph obtained by removing the orientation of the arcs is connected.

- **Eulerian Graphs**: A graph (directed or undirected) is Eulerian if there is a circuit (a closed walk) that crosses each edge exactly once. Such a circuit is called an Eulerian circuit. An connected undirected graph is Eulerian if and only if every node has even degree. Similarly, a weakly connected directed graph is Eulerian if and only if each node has its indegree equal to its outdegree. An Eulerian walk is a walk that crosses each edge exactly once, but is not required to start and end at the same node. So, an Eulerian circuit is simply an Eulerian walk that begins and ends at the same node. Suppose $s \neq t$ are two nodes in a connected graph G. Then there is an Eulerian walk from s to t if both s and t have odd degree and every other node has even degree. If G is a weakly connected directed graph with distinct nodes s, t, then there is an Eulerian walk from s to t if the outdegree of s is one greater than its indegree, the indegree of t is one greater than its outdegree, and all other nodes have equal indegree and outdegree. Eulerian circuits and walks can be found in polynomial time.

- **Hamiltonian Cycles and Paths**: A Hamiltonian cycle in a graph (directed or undirected) is a cycle that visits each node exactly once. A Hamiltonian path in a graph is a path that visits each node exactly once. It is NP-complete to determine if a directed or undirected graph has either a Hamiltonian cycle or a Hamiltonian path [45].

- **Independent Set**: An independent set in an undirected graph $G = (V, E)$ is a subset of nodes W such that no edge in E has both endpoints in W. It is NP-complete to determine if a graph has an independent set of a given size [45]. The size of the largest independent set in a graph G is denoted by $\alpha(G)$.

- **Clique**: A clique in an undirected graph $G = (V, E)$ is a subset of nodes W such that any two distinct nodes $u, v \in W$ have $uv \in E$. It is NP-complete to determine if a graph has a clique of a given size [45]. The size of the largest clique in a graph G is denoted by $\omega(G)$.

- **Graph Colourings**: A colouring of an undirected graph $G = (V, E)$ is a partition of V into disjoint subsets $V_1, \ldots, V_k$ such that each such subset V_i is an independent set. For a given

k, if there is such a partition of G into k disjoint subsets then we say that G can be coloured with k colours. It is NP-complete to determine if a graph can be coloured with k colours for any $k \geq 3$ [45], though it is polynomial time solvable for $k = 2$. The minimum value k for which G can be coloured with k colours is called the *choromatic number* of G and is denoted by $\chi(G)$.

- **Perfect Graphs**: A perfect graph is an undirected graph $G = (V, E)$ such that $\omega(G') = \chi(G')$ for all vertex-induced subgraphs G' of G. If G is perfect, then $\alpha(G), \omega(G)$ and $\chi(G)$ can be computed in polynomial time [50]. If G is a cycle with an odd number k of nodes where $k \geq 5$ then $\chi(G) = 3$ but $\omega(G) = 2$ so G is not perfect. The Strong Perfect Graph Theorem asserts that G is perfect if and only if G and its complement does not contain any cycle with odd length $k \geq 5$ as an induced subgraph [33]. Some examples of perfect graphs are the following (see, *e.g.*, [48]):

  - Bipartite graphs. Trivially, $\chi(G) = \omega(G) = 2$ if G contains an edge.
  - Comparability graphs. A graph $G = (V, E)$ is a comparability graph if the edges can be oriented so that if arcs uv and vw are in G, then so is the arc uw.
  - Interval graphs. A graph $G = (V, E)$ is an interval graph if there are closed intervals $I_v \subseteq \mathbb{R}$ for each $v \in V$ such that $I_u \cap I_v \neq \emptyset$ if and only if $uv \in E$.

- **Metric Graphs**: An undirected graph $G = (V, E)$ with non-negative edge weights $d_{uv}, uv \in E$ is called a *metric* or a *symmetric metric* if G is a complete graph and the triangle inequality $d_{uv} \leq d_{uw} + d_{wv}$ holds for any $u, v, w \in V$. A directed graph $G = (V, A)$ with non-negative arc weights $d_{uv}, uv \in A$ is called an *asymmetric metric* if every pair of distinct nodes $u, v \in V$ is connected by an arc $uv \in A$ and the directed triangle inequality $d_{uv} \leq d_{uw} + d_{wv}$ holds. A distinguishing feature between symmetric metrics and asymmetric metrics is that we always have $d_{uv} = d_{vu}$ in symmetric metrics whereas we may have $d_{uv} \neq d_{vu}$ for some nodes u, v in an asymmetric metric. We can extend the weight notation and define d_{vv} in any metric (symmetric or asymmetric), which is always zero. A symmetric metric is similar to a metric from topology except we may have $d_{uv} = 0$ for distinct points u, v. An important metric that occurs in this thesis is the shortest path metric. In the undirected case, suppose $G = (V, E)$ is a connected (but not necessarily complete) graph with non-negative edge distances d_e. Then we obtain a symmetric metric $G' = (V, E')$ with distances d' over the same set of nodes where d'_{uv} is the length of the shortest path from u to v in G. Similarly, we can obtain an asymmetric metric $G'' = (V, A')$ from a strongly connected directed graph $G = (V, A)$ with non-negative arc distances.

- **Circulations**: A circulation C in a directed graph $G = (V, A)$ is an assignment of a non-negative value C_{uv} to each arc $uv \in A$ such that the total C-value of incoming arcs equals

the C-value of outgoing arcs at each node. Formally, for each $v \in V$ we have $\sum_{uv \in E} C_{uv} = \sum_{vw \in E} C_{vw}$. We also say that *flow conservation* holds at node v. The *support* of a circulation C is $\{uv \in A : C_{uv} > 0\}$, the set of arcs that are assigned a non-zero value in the C.

- **Flow**: Given two distinct nodes s, t in a directed graph $G = (V, A)$, a flow F from s to t is an assignment of a non-negative value F_{uv} to each arc $uv \in A$ such that flow conservation holds in V at each node except, perhaps, s, t. We say that the value of the flow F is $\sum_{sw \in A} F_{sw} - \sum_{us \in A} F_{us}$ (which equals $\sum_{ut \in A} F_{ut} - \sum_{tw \in A} F_{tw}$ by flow conservation on other nodes). In most cases in this thesis, there will be no flow entering s nor will there be any flow exiting t so the value of the flow in such cases would simply be $\sum_{sw \in A} F_{sw} = \sum_{ut \in A} F_{ut}$. We say that a flow F is integral if F_{uv} is an integer for each $uv \in A$. The support of a flow F is the set of arcs $uv \in A$ for which $F_{uv} > 0$. We also say that $v \in V - \{s, t\}$ *supports* some flow in F if $F_{uv} > 0$ for some $uv \in A$ (or, equivalently, $F_{vw} > 0$ for some $vw \in A$).

  Suppose that F is a flow of value k where each $F_{uv}, uv \in A$ is a non-negative integer. Then we may find a collection of k paths $P_1, \ldots, P_k$ from s to t such that each arc $uv \in A$ appears on at most F_{uv} such paths. Furthermore, if we define C_{uv} values for each arc $uv \in A$ to be F_{uv} minus the number of these paths that contain uv, then C is a circulation. Suppose F and F' are flows from s to t. Then we may form a flow $F + F'$ from s to t by assigning $F_{uv} + F'_{uv}$ to each arc $uv \in A$. Similarly, if F, F' are circulations then $F + F'$ is also a circulation. Finally, if F is an flow from s to t and F' is a circulation, then $F + F'$ is still an flow from s to t. Note that if both F and F' are integral, then so is $F + F'$.

- **Cuts**: Given a graph G (directed or undirected), a cut is simply a subset of nodes apart from $\emptyset$ or V. If $G = (V, E)$ is undirected, then for any subset of nodes $S \subseteq V$ we let $\delta(S) = \{uv \in E : u \in S, v \notin S \text{ or } u \notin S, v \in S\}$ be the collection of edges with one exacly endpoint in S. If $G = (V, A)$ is a directed graph and S is a cut then we let $\delta^+(S) = \{uv \in A : u \in S, v \notin S\}$ be the collection of arcs that exit S. Similarly, we let $\delta^-(S) = \{uv \in A : u \notin S, v \in S\}$ denote the collection of arcs that enter S. Sometimes we are interested in the set of edges or arcs incident to a node v. For simplicity, we use $\delta(v), \delta^+(v), \delta^-(v)$ to denote $\delta(\{v\}), \delta^+(\{v\}), \delta^-(\{v\})$, respectively. If we have a non-negative capacity c_e for each edge/arc e, then we say the capacity of a cut S in an undirected graph is $\sum_{uv \in \delta(S)} c_{uv}$. In directed graphs, the capacity of a cut S is the total capacity of the outgoing edges $\sum_{uv \in \delta^+(S)} c_{uv}$.

- **Capacitated Flow**: If $G = (V, A)$ is a directed graph with non-negative capacities c_e on the arcs and s, t are two distinguished nodes, a capacitated flow from s to t is a flow F from s to t that satisfies $F_{uv} \leq c_{uv}$ for each arc $uv \in A$. An $s - t$ cut is a cut S with $s \in S$ and $t \notin S$. It is not too hard to see that the value of any capacitated flow F from s to t cannot exceed the capacity of any $s - t$ cut. The max-flow/min-cut theorem (*e.g.* [84]) asserts that

there is a capacitated flow F' from s to t and an $s-t$ cut S' such that the value of F' equals the capacity of the cut S'. Furthermore, if all capacities $c_{uv}, uv \in A$ are integers, then such a flow F' may be taken to be integral. Finally, if S is a cut and F is a flow, then we let $F(\delta^+(S))$ denote $\sum_{uv \in \delta^+(S)} F_{uv}$, the total flow across arcs exiting the cut S. Similarly, we let $F(\delta^-(S)) = \sum_{uv \in \delta^-(S)} F_{uv}$ denote the total flow across arcs entering S.

1.2.2 Approximation Algorithms

An optimization problem Φ is abstractly defined as a collection of pairs of the form (F, M) where F is a set of *feasible solutions* and M is an objective function $F \to \mathbb{R}$. We think of a specific pair $(F, M) \in \Phi$ as an instance of the optimization problem Φ. Of interest is finding, for each $(F, M) \in \Phi$, some feasible solution $f \in F$ that maximizes or minimizes $M(f)$ (whenever such an extreme exists). If our goal is to maximize the objective function, then we say that Φ is a *maximization problem*. Otherwise, if our goal is to minimize the objective function, then we say that Φ is a *minimization problem*.

An instance (F, M) of an optimization problem Φ is often given implicitly. For example, an instance of the problem of finding the largest independent set in a graph is given by an undirected graph $G = (V, E)$. For such an instance (F, M), we have the feasible set F being all subsets S of V for which no two nodes in S are the endpoints of an common edge in E. The objective function M in this case is simply $|S|$.

In computing science, we are interested in optimization problems that can be implicitly encoded with a finite number of bits. For example, there are a variety of well-known ways to succinctly represent graphs using finite sequences of `0`s and `1`s so we may encode an instance of the maximum independent set problem where the number of bits used is polynomial in the number of nodes and edges of the graph. An integer k may also be encoded in $O(\log k)$ bits using the standard base 2 expansion. In general, we will not be concerned with such low-level details in this thesis. Apart from some discussion in the following subsection on linear programming and in the running time of some hardness reductions, we will proceed without giving further thought to the matter.

Suppose that Φ is a minimization problem. An α-approximation for Φ is an algorithm which, for every instance (F, G) of Φ, is guaranteed to find some feasible solution $f \in F$ such that $M(f) \leq \alpha M(f^*)$ where $f^* \in F$ is the optimum solution for instance (in the problems we consider, there is indeed be a solution f^* with $M(f^*) = \sup_{f \in F} M(f)$). We often let OPT denote the optimum value of the instance at hand if it is clear from the context. If Φ is a maximization problem, then we choose the convention that an α-approximation is an algorithm that finds a feasible solution to a given instance with value at least an $\frac{OPT}{\alpha}$. For example, we develop an $O(\log n)$-approximation for the Unsplittable Flow on Paths problem where n is the number of tasks which means the algorithm finds a subset of tasks that can be feasibly routed whose profit is at least $\frac{OPT}{\Omega(\log n)}$.

Since the main point of studying approximation algorithms is to address NP-hard optimiza-

tion problems with efficient algorithms, all of the approximation algorithms we develop run in polynomial-time in the size of the input unless we explicitly state otherwise. One notable exception is in Section 2.3. We sometimes use the notion of a *quasi-polynomial time* algorithm in this thesis. These are algorithms whose running time is $n^{O(\log^k n)}$ for some constant k.

1.2.3 Linear Programming

One of the most important optimization problems is *Linear Programming*. An instance of Linear Programming is presented in the following form where $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$ and $c \in \mathbb{R}^m$ are fixed values and $x \in \mathbb{R}^n$ is a vector of variables $x_1, \ldots, x_n$.

$$\begin{aligned} \text{maximize:} \quad & \sum_{i=1}^{n} c_i x_i & & & \\ \text{subject to:} \quad & \sum_{i=1}^{n} A_{ji} x_i & \leq & b_i & \forall 1 \leq j \leq m \\ & x_i & \geq & 0 & \forall 1 \leq i \leq n \end{aligned}$$

The goal is to find values $x_1, \ldots, x_n$ to maximize the objective function $\sum_{i=1}^{n} c_i x_i$ such that each of the constraints are satisfied. We often refer to an instance of Linear Programming as, simply, a linear program or an LP. If $x \in \mathbb{R}^n$ satisfies all constraints of a particular LP then we will say that x is *feasible* for that LP. The set of feasible points is called the *polytope* of the linear program.

Through some elementary transformations of the objective functions and/or constraints, Linear Programming can be seen to be equivalent to variants where we minimize the objective function and/or allow equality constraints. An instance of a maximization version of Linear Programming falls into one of three categories. If the polytope of the LP is empty, then we say that the instance is *infeasible*. Otherwise, if for any $\delta \in \mathbb{R}$ we have some feasible solution x with $cx^t \geq \delta$ then the instance is *unbounded*. In the final case, we have that there is a feasible solution and there is a least upper bound γ such that all feasible solutions x have $cx^t \leq \gamma$. Since the set of feasible solutions is clearly closed in the standard topology on $\mathbb{R}^n$ (as the polytope is the intersection of the closed sets defined by the inequalities), then there is actually a feasible point x with $cx^t = \gamma$. In such a case, we say that the instance has a *finite optimum*.

Say a collection of constraints is linearly independent if the vectors corresponding to the constraints (*i.e.* the row vector A_j for a constraint of the form $A_j x^t \leq b_j$) are linearly independent over $\mathbb{R}$. We say that a constraint $A_j x^t \leq b_j$ is *tight* for a given $x \in \mathbb{R}^n$ if $A_j x^t = b_j$ holds. A feasible solution x is said to be a *basic* solution if there are n linearly independent tight constraints. It is a known fact that if a linear program has a finite optimum, then there must be some basic feasible solution x attaining the optimum value.

In this thesis, we consider instances of Linear Programming that have all values in A, b, and c being rational. Consider such an LP that has a finite optimum. Let x be a basic solution, A' be the $n \times n$ matrix whose rows correspond to some n linearly independent tight constraints, and b' be the column vector of length n corresponding to the b_j values for the constraints used to form A'. Since

x is the unique solution to the system $A'x^t = b'$ and all entries of A' and b' are rational, then all entries of x are rational too. This also implies that the optimum value of an LP with rational values is rational.

By saying an algorithm solves Linear Programming, we mean that the algorithm will either determine if a given linear program is infeasible, unbounded, or has a finite optimum. If it has a finite optimum, then the algorithm will also produce a basic feasible solution x attaining this optimum value. Linear programs can be solved efficiently, but we first have to mention the notion of bit complexity of integers. For a rational number q, let $l(q)$ denote the number of bits required to write the numerator and denominator of q in binary. That is, $l(q) = O(\log n + \log d)$ where $q = \pm n/d$. Given a linear program with rational entries described by A, b, c as above, let $\Delta := \max\ l(q)$ where the max is over values q appearing in either A, b, or c.

One approach for solving linear programming in polynomial time (in n, m and Δ) is to use the *interior point method* (*e.g.* Karmakar's algorithm [59]). In some cases, the number of constraints is exponentially larger than the number of variables and we cannot afford to explicitly list all of them. To deal with this, we use the concept of a *separation oracle*. A separation oracle is an algorithm that solves the following problem. Given a rational point $x \in \mathbb{Q}^n$, either determine that x is feasible or return the explicit description of a violated constraint. If a linear program has a separation oracle that runs in polynomial time in n, Δ, and the number of bits used to represent the rational point x in question, then it is still possible to solve the linear program in time that is polynomial in only n and Δ using the *ellipsoid method* as in [49].

A common technique in combinatorial optimization is to formulate a problem as an instance of Integer Programming. An instance of Integer Programming is presented in the same way as an instance of Linear Programming with the additional restriction that the variables $x_1, \ldots, x_n$ take only integer values. Such an integer program is usually formulated so there is a natural correspondence of feasible integer points in the integer program and feasible solutions to the original optimization problem. Integer Programming is an NP-hard optimization problem so formulating a combinatorial optimization problem in this way does not really gain us much in terms of polynomial-time solvability. However, if we relax the condition that the variables take integer values and allow fractional values, then we have an instance of Linear Programming that we will commonly call an *LP relaxation* of the optimization problem. In some sense, an optimum solution to an LP relaxation is a sort of fractional approximation to the original problem itself. An example of how to formulate and use an LP relaxation of a combinatorial optimization problem can be found in Section 1.2.7 following the presentation of background material.

Suppose that (F, G) is an instance of a particular optimization problem Φ. Suppose further that we have an integer programming formulation of the problem with optimum value OPT. Finally, say that OPT_f is the optimum value of the LP relaxation obtained from this integer program. Note that since any integer point that is a solution to the Integer Programming instance is a feasible point

in the LP relaxation, then $OPT \geq OPT_f$ if Φ is a minimization problem and $OPT \leq OPT_f$ if Φ is a maximization problem. We say that the *integrality gap* of this LP relaxation is the ratio $\frac{OPT}{OPT_f}$ if Φ is a minimization problem, or $\frac{OPT_f}{OPT}$ if Φ is a maximization problem.

More generally, suppose Φ is an optimization problem and we have a particular method to obtain an Integer Programming formulation and its LP relaxation. We will say that the integrality gap of these LP relaxations is the least upper bound Z such that the LP relaxation of any particular instance of Φ obtained in the given manner has integrality gap at most Z. In some cases, Z may be unbounded so we sometimes examine the integrality gap of Φ when restricted to inputs of a certain size. For example, if Φ is an optimization problem over a graph then saying the integrality gap of a certain LP relaxation of Φ is $O(\log |V|)$ means the integrality gap of an instance over a graph $G = (V, E)$ is dominated asymptotically by $\log |V|$ as $|V|$ increases.

In some fortunate cases, it is possible to find LP relaxations of an optimization problem Φ that are exact. By this, we mean the optimum value of the LP relaxation of an instance (F, M) of Φ is equal to the optimum value of the objective function M over F. In the language of integrality gaps, this means the integrality gap of the LP relaxation is precisely 1. Furthermore, the basic feasible solutions of these LP relaxations may also have integer values at each component. If a polytope has the property that all basic feasible solutions x have each component being integral, then we will say that the polytope is *integral*. We will encounter many (similar) integral polytopes in this thesis, especially in Chapter 3.

1.2.4 Matroids

A matroid can be thought of as a formal analogy of linear independence in vector spaces to other set systems.

Definition 1.2.1 *A pair $\mathcal{M} = (X, \mathcal{I})$ where X is a finite set and $\mathcal{I} \subseteq \mathcal{P}(X)$ is a collection of subsets of X is called a* matroid *if the following conditions are satisfied.*

1. $\emptyset \in \mathcal{I}$
2. *If $S \in \mathcal{I}$ then $S' \in I$ for any $S' \subseteq S$*
3. *If $S, T \in \mathcal{I}$ and $|S| < |T|$, then there is some $x \in T - S$ such that $S \cup \{x\} \in I$*

The second condition is the *hereditary propery* of matroids and the third condition is the *exchange property*. There are many examples of matroids; we list a few below.

- X is finite subset of a vector space and a $\mathcal{I}$ consists of all subsets of linearly independent elements in X.
- X is the set of edges of an undirected graph $G = (V, E)$ and I is the collection of all acyclic subsets of E. This is called the *graphic matroid*.

- X is the set of edges of an undirected graph $G = (V, E)$ and I is the collection of all subsets F of E where each connected component in the graph $G = (V, F)$ has at most one cycle. This is called the *bicircular matroid* [86].

- Suppose we have a partition of X into subsets $X_1, X_2, \ldots, X_k$. Furthermore, suppose we have a non-negative integer bound c_i for every $1 \leq i \leq k$. We get a *partition matroid* $(X, \mathcal{I})$ where $\mathcal{I}$ consists of all subsets Y of X satsifying $|Y \cap X_i| \leq c_i$ for each $1 \leq i \leq k$.

Any member of $\mathcal{I}$ in a matroid $\mathcal{M} = (X, \mathcal{I})$ is called an *independent set* (not to be confused with an independent set in a graph, this term is more related to the notion of a set of vectors being linearly independent). A *base* of $\mathcal{M} = (X, \mathcal{I})$ is simply a maximal, with respect to inclusion, independent set (analogous to a basis in a vector space). From the exchange property, we easily see that $|Y| = |Z|$ for any two bases $Y, Z \in \mathcal{I}$. The *rank* $r(\mathcal{M})$ of a matroid $\mathcal{M}$ is then unambiguously defined as the size of a base of $\mathcal{M}$. For example, if $G = (V, E)$ is a connected graph then the bases in the graphic matroid on G are the spanning trees of G, each having $|V| - 1$ edges. Generalizing this notion, for any subset S of X we let $r(S)$, the rank of S, be $\max_{T \in \mathcal{I}} |T \cap S|$. This is the same as the maximum of $|T|$ over $T \in \mathcal{I}$ with $T \subseteq S$. Note that $r(X) = r(\mathcal{M})$.

Of interest in optimization is when the items of a matroid have an associated weight.

Definition 1.2.2 *A* weighted matroid *is a matroid $\mathcal{M} = (X, \mathcal{I})$ with a weight function $w : X \to \mathbb{R}$. The weight $w(S)$ of a subset $S \subseteq X$ is $\sum_{i \in S} w(i)$.*

In this thesis, all weighted matroids will have $w(x) \geq 0$ for any $x \in X$.

If we have a polynomial time (in $|X|$) algorithm for determining if a given subset of X is in $\mathcal{I}$, then there is a very simple polynomial time algorithm that finds a base in $\mathcal{I}$ of minimum total weight. Start with an empty set $Y \leftarrow \emptyset$. Process the items in X in increasing order of weight and add an item i to Y if $Y \cup \{i\} \in \mathcal{I}$. Once all items have been processed, Y is a minimum weight base [84]. If we process the items in decreasing order of weight instead, then the resulting set Y is a maximum weight base.

There are also two polytopes that we can associate with a matroid $\mathcal{M}$. For each $i \in X$, we associate a real-valued variable z_i. The *independent set polytope* $\mathcal{P}_I(\mathcal{M})$ is given by the constraints:

$$\begin{array}{rcll} \sum_{i \in S} z_i & \leq & r(S) & \forall S \subseteq X \\ 0 \leq z_i & \leq & 1 & \forall i \in X \end{array}$$

Though, we could omit the upper bound $z_i \leq 1$ for each $i \in X$ because $r(\{i\}) \in \{0, 1\}$. The *base polytope* $\mathcal{P}_B(\mathcal{M})$ is the intersection of the polytopes $P_I(\mathcal{M})$ and the polytope defined by:

$$\sum_{i \in X} z_i = r(\mathcal{M})$$

In other words, $\mathcal{P}_B(\mathcal{M})$ is obtained by adding the above constraint to the list of constraints defining $\mathcal{P}_I(\mathcal{M})$. The new constraint asserts that the total z-value of the items is equal to the size of a base of the matroid.

Now, for any independent set $Y \in \mathcal{I}$ we let χ_Y denote the characteristic vector of Y with $\chi_{Y,i} = 0$ for $i \notin Y$ and $\chi_{Y,i} = 1$ for $i \in Y$. We then have $\chi_Y \in \mathcal{P}_I(\mathcal{M})$ and if Y is also a base then we have $\chi_Y \in \mathcal{P}_B(\mathcal{M})$ as well. Conversely, given any vector z with $z_i \in \{0, 1\}$ for each $i \in X$, we form the set $Y_z \subseteq X$ with $i \in Y_z$ if and only if $z_i = 1$. It is also true that if $z \in \mathcal{P}_I(\mathcal{M})$ then Y_z is an independent set and if $z \in \mathcal{P}_B(\mathcal{M})$ then Y_z is a base. Thus, the lattice points in $\mathcal{P}_I(\mathcal{M})$ and $\mathcal{P}_B(\mathcal{M})$ correspond exactly to independent sets and bases of $\mathcal{M}$, respectively.

What is even more remarkable is that the $\mathcal{P}_I(\mathcal{M})$ and in $\mathcal{P}_B(\mathcal{M})$ are both integral for every matroid $\mathcal{M}$. This means the maximum or minimum weight of a base in $\mathcal{M}$ is the same as the maximum or minimum of the objective function $\sum_{i \in X} w(i) z_i$ over points $z \in \mathcal{P}_B(\mathcal{M})$, respectively. Furthermore, finding a basic optimum point in the linear program whose goal is to maximize or minimize this objective function over $z \in \mathcal{P}_B(\mathcal{M})$ corresponds to such a base of maximum or minimum total weight [84].

Finally, we note that if there is a polynomial time algorithm that determines if a subset of X is independent, then there is also a polynomial time separation oracle for the polytopes $\mathcal{P}_I(\mathcal{M})$ and $\mathcal{P}_B(\mathcal{M})$ [84].

1.2.5 Matroid Intersection

Much is known regarding the intersection of two matroids [84]. Given two matroids $\mathcal{M}_1 = (X, \mathcal{I}_1)$ and $\mathcal{M}_2 = (X, \mathcal{I}_2)$ over a common ground set X, we define their intersection $\mathcal{M}_1 \cap \mathcal{M}_2$ as the pair $(X, \mathcal{I}_1 \cap I_2)$. In general, $\mathcal{M}_1 \cap \mathcal{M}_2$ is not a matroid since the exchange property might not hold. Consider the following example. We have $X = \{a, b, c\}$ and both matroids are partition matroids. In $\mathcal{M}_1$, we have $X_1 = \{a, b\}$ and $X_2 = \{c\}$ being the partition with bounds $c_1 = c_2 = 1$. In $\mathcal{M}_2$ we have $X'_1 = \{a\}$ and $X'_2 = \{b, c\}$ with bounds $c'_1 = c'_2 = 1$. In $\mathcal{M}_1 \cap \mathcal{M}_2$, both $\{b\}$ and $\{a, c\}$ are in $\mathcal{I}_1 \cap \mathcal{I}_2$, but neither $\{a, b\}$ nor $\{b, c\}$ are in $\mathcal{I}_1 \cap \mathcal{I}_2$. That is, the exchange property fails to hold for $\mathcal{M}_1 \cap \mathcal{M}_2$.

Though the exchange property fails for $\mathcal{M}_1 \cap \mathcal{M}_2$, we can still define a notion of rank. For any subset S of X, we can define the rank $r(S)$ in $\mathcal{M}_1 \cap \mathcal{M}_2$ as the maximum size of any subset $Y \subseteq S$ with $Y \in \mathcal{I}_1 \cap \mathcal{I}_2$. Notice that we say *maximum* instead of *maximal*, this is an important distinction. It is still of interest to find a maximum size subset of X that is in both $\mathcal{I}_1$ and $\mathcal{I}_2$.

For example, consider the special case where both $\mathcal{M}_1$ and $\mathcal{M}_2$ are partition matroids and all capacities c_i are 1 in both matroids. The problem of finding a maximum size set in $\mathcal{I}_1 \cap \mathcal{I}_2$ is equivalent to finding the maximum cardinality matching in a bipartite graph. To see this, let $B = (L \cup R; E)$ be the following bipartite graph where L and R denote the vertex sets of the two sides. Each partition in $\mathcal{M}_1$ has a node in L and each partition in $\mathcal{M}_2$ has a node in R. For each element in $i \in X$ let u be the node corresponding to the partition containing i in $\mathcal{M}_1$ and let v be the node corresponding to the partition containing i in $\mathcal{M}_2$. Add an edge connecting u to v in B. We then see that matchings in B correspond exactly to subsets of X that are independent in both

$\mathcal{M}_1$ and $\mathcal{M}_2$.

Finding the maximum size of a set common to both $\mathcal{I}_1$ and $\mathcal{I}_2$ can be solved in polynomial time provided we have a polynomial time algorithm for testing independence in both $\mathcal{M}_1$ and $\mathcal{M}_2$. Moreover, if we have a weight $w(i)$ on items $i \in X$ then for each integer k we can find a set $Y \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|Y| = k$ of maximum or minimum total weight or determine that no such set exists. In particular, we can find a set Y of size $r(X)$ of maximum or minimum total weight among all such sets. Extending the previous example, this captures the problem of finding a maximum or minimum weight perfect matching in a bipartite graph.

As before, we can associate polytopes to $\mathcal{M}_1 \cap \mathcal{M}_2$. The independent set polytope $P_I(\mathcal{M}_1 \cap \mathcal{M}_2)$ is simply $P_I(\mathcal{M}_1) \cap P_I(\mathcal{M}_2)$, the set of all vectors z that satisfy all the constraints defining both polytopes. This intersection is non-empty since the all-zero vector is in both $P_I(\mathcal{M}_1)$ and $P_I(\mathcal{M}_2)$. Similarly, we can define the "base" polytope $P_B(\mathcal{M}_1 \cap \mathcal{M}_2)$ as $P_B(\mathcal{M}_1) \cap P_B(\mathcal{M}_2)$. However, this intersection may be empty. It will certainly be empty if $r(\mathcal{M}_1) \neq r(\mathcal{M}_2)$. However, even if $r(\mathcal{M}_1) = r(\mathcal{M}_2)$ there may still be no vector z that is common to both.

The elegant properties of $P_I(\mathcal{M})$ and $P_B(\mathcal{M})$ being integral for any matroid $\mathcal{M}$ carry over to the intersection of two matroids. Specifically, $P_I(\mathcal{M}_1 \cap \mathcal{M}_2)$ is also integral and if $P_B(\mathcal{M}_1 \cap \mathcal{M}_2) \neq \emptyset$, then it too is integral [84]. This is not true in general for the intersection of two arbitrary integral polytopes as the discussion in the next paragraph highlights. Frequently encountered in this thesis is the intersection of two partition matroids. Suppose $X_1^1, \ldots, X_{k_1}^1$ are the partitions in the first matroid with capacities $c_1^1, \ldots, c_{k_1}^1$ and $X_1^2, \ldots, X_{k_2}^2$ are the partitions in the second matroid with capacities $c_1^2, \ldots, c_{k_2}^2$. There is a simpler set of constraints that defines the same polytope for the intersection of these two partition matroids. Namely, the following polytope is also integral.

$$\begin{array}{rcll} \sum_{i \in X_j^1} z_j & \leq & c_i^1 & \forall 1 \leq j \leq k_1 \\ \sum_{i \in X_j^2} z_j & \leq & c_i^2 & \forall 1 \leq i \leq k_2 \\ 0 \leq z_i & \leq & 1 & \forall i \in X \end{array}$$

We remark that the intersection of three matroids $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ does not enjoy any of these nice properties. For example, suppose $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ are all partition matroids where the capacities c_i in the partitions in each matroid are all 1. Then finding the largest subset $Y \subseteq X$ that is independent in all three matroids is equivalent to the NP-hard tripartite matching problem in 3-uniform hypergraphs. Using this example one can also show that $P_I(\mathcal{M}_1) \cap P_I(\mathcal{M}_2) \cap P_I(\mathcal{M}_3)$ may be non-empty and not integral (*i.e.* some coordinates z_i may be non-integers at some basic point z).

1.2.6 Complexity and Lower Bounds

In the decision version of a maximization problem Φ, we are given an instance of Φ and a value k. The problem is to determine if the optimum value of the instance is at least k. Similarly, the decision version of a minimization problem is presented in the same way and we are to determine if the optimum value of an instance is at most k. We say an optimization problem Φ is NP-hard if

there is a polynomial time reduction from every problem in NP to the decision version of Φ. Thus, the existence of a polynomial time algorithm to find the optimum value of an NP-hard optimization problem Φ implies P = NP since it could be used to solve the decision version of the problem.

Proving a decision problem Φ to be NP-hard shows that finding the optimum solution is intractible so a natural question is to ask how well we may approximate the optimum value of Φ. Indeed, some NP-hard problems can be approximated very well (*e.g.* within $1+\epsilon$ for ϵ being inverse-polynomial in the input size) and others cannot be approximated within any non-trivial bound (*e.g.* not within $\Omega(n^{1-\epsilon})$ for any constant $\epsilon > 0$) unless P = NP. In this section, we will survey a hierarchy of categories of approximation algorithms. We will start with approximation algorithms that find solutions extremely close to the optimum.

Definition 1.2.3 *A Fully Polynomial Time Approximation Scheme (FPTAS) for an optimization problem Φ is an approximation algorithm with approximation ratio $1+\epsilon$ whose running time is polynomial in the size of the instance and $\frac{1}{\epsilon}$.*

For example, the famous Knapsack problem has a relatively simple FPTAS that runs in time $O(\frac{n^3}{\epsilon})$ where n is the number of items [55]. Note that we can even choose ϵ to be inverse polynomial in n and still obtain a polynomial time algorithm. However, even an FPTAS has its limitations. If one looks closely at the known NP-hardness reductions for the Knapsack problem, we see that the difference between "yes" and "no" instances is inverse exponential in n (which can be written in poly(n) bits) so we cannot distinguish between such instances using an FPTAS.

A slightly weaker approximation algorithm is the following.

Definition 1.2.4 *A Polynomial Time Approximation Scheme (PTAS) for an optimization problem Φ is an approximation algorithm with approximation ratio $1+\epsilon$ whose running time is polynomial for any fixed constant $\epsilon > 0$.*

For example, a PTAS may have running time $O(n^{1/\epsilon})$ on inputs of size n which is polynomial for fixed constants $\epsilon > 0$, but not for $\epsilon = \frac{1}{\text{poly}(n)}$. It is possible that an NP-hard optimization problem can be approximated by a PTAS whereas the existence of an FPTAS would imply P = NP. Consider the Feedback Arc Set problem in tournaments. The input consists of a directed graph $G = (V, A)$ with a single arc between any two nodes (so, for any two distinct nodes $u, v \in V$ either $uv \in A$ or $vu \in A$, but not both). The problem is to delete the fewest arcs possible so the resulting graph has no cycles. The problem is NP-complete (*e.g.* [2]) and it does have a PTAS [66]. However, the existence of an FPTAS would imply P = NP for the following reason. Any instance $G = (V, A)$ of the Feedback Arc Set problem in tournaments has optimum value that is an integer at most $|A| < |V|^2$. Suppose there was a $(1+\epsilon)$-approximation for the problem running in time $O(\text{poly}(|V|)\cdot\text{poly}(1/\epsilon))$. By choosing $\epsilon = \frac{1}{|V|^2}$, we would have a polynomial-time $(1+\frac{1}{|V|^2})$-approximation. However, such an approximation would be an exact algorithm because

$$\left(1+\frac{1}{|V|^2}\right)\cdot OPT = OPT + \frac{OPT}{|V|^2} < OPT + 1$$

Since the number of arcs deleted is an integer, then the number of arcs deleted by such an algorithm would then be exactly OPT and we could use this algorithm to solve the Feedback Arc Set problem.

In general, consider the notion of *strong NP-hardness*. An NP-hard problem is said to be strongly NP-hard if it remains NP-hard if all numbers appearing in the input are, in absolute value, at most polynomial in the length of the input. This means the gap between optimum solutions to "yes" instances (instances produced through a reduction from a "yes" instance of a problem in NP) and optimum solutions to "no" instances is at least $1 + \frac{1}{\text{poly}(n)}$. So, if a problem is strongly NP-hard, then an FPTAS could be used to distinguish between yes and no instances and, ultimately, show P = NP. That is, unless P = NP, then no strongly NP-hard problem has an FPTAS.

The class of optimization problems that can be approximated with a PTAS is similarly denoted PTAS. Also of interest is the class of optimization problems APX consisting of problems that have constant-factor approximation algorithms. Clearly PTAS $\subseteq$ APX and it is possible to show the inclusion is strict (see the example at the end of the Subsection 1.2.8). Similar to the theory of NP-hardness, there is a notion of APX-hardness. Say a PTAS reduction from one optimization problem Φ to another Ψ is a polynomial-time reduction f such that for any instance I of Φ, a solution within a factor $1 + \epsilon$ from the optimum value of the instance $f(I)$ of Ψ corresponds to a solution to I within a factor $1 + c\epsilon$ of the optimum value of I where c is some constant. That is, a PTAS for Ψ would imply a PTAS for Φ. Then a problem Φ is APX-hard if there is a PTAS reduction from every problem in APX to Φ. Finally, a problem is APX-complete if it is both in APX and APX-hard. See [10] for more information.

For example, it can be shown that the following problem is APX-hard. Given a graph $G = (V, E)$, we are to assign one of three colours to each node of V to maximize the number of edges whose endpoints do not receive the same colour. It is one optimization variant of the NP-complete 3-colouring problem. This problem is also in APX since the simple greedy algorithm that colours the nodes one at a time and always chooses the colour that introduces the least number of monochromatic edges is a 3/2-approximation. The idea behind this is that when considering a node v, at most 1/3 of the edges incident to v that already have their other endpoint coloured will be monochromatic.

The celebrated PCP theorem [7, 8] can be shown to be equivalent to the fact that unless P = NP, there is some constant $c > 1$ such that the above optimization version of 3-colouring has no polynomial-time c-approximation. An important implication is that unless P = NP, no APX-hard problem has a PTAS. Another example of an APX-complete problem is the classical Traveling Salesman problem in symmetric metrics. That it is in APX follows from any of the constant-factor approximations (the best being 3/2 [32]). APX-hardness of the Traveling Salesman problem was originally shown in [74]. In the next subsection, we will demonstrate the more simple result that it is strongly NP-hard which will rule out a PTAS unless P = NP.

After APX comes problems that can be approximated within some bounded ratio $O(f(n))$ where $f(n) \to \infty$ as $n \to \infty$ where n denotes the size of the input. The classic Set Cover problem,

a generalization of the Minimum Weight Vertex Cover problem mentioned in the next section to hypergraphs (graphs where edges may connect more than two nodes), can be approximated within a factor $H_n = \ln n + O(1)$ where n is the number of elements (or hyperedges if we use the terminology of hypergraphs) [35]. Here, H_n is the n'th harmonic number $\sum_{k=1}^{n} \frac{1}{k}$. For some problems, there are also super-constant lower bounds on how they may be approximated. For instance, there is a constant $0 < c$ such that there is no $c \ln n$ approximation for Set Cover unless P = NP [15]. Sometimes these assumptions are strengthed to provide tighter lower bounds. For any constant $\epsilon > 0$, it is known that we cannot approximate Set Cover with a factor $(1 - \epsilon) \ln n$ in polynomial time unless all problems in NP can be solved in time $n^{O(\log \log n)}$ [37]. This is a stronger assumption that P $\neq$ NP, but it is still unknown and it highlights a more general computational barrier to finding a better approximation algorithm for Set Cover.

In fact, sometimes lower bounds can be established that rule out all but the most trivial approximation algorithms. A classic example is Maximum Independent Set problem in which we are given an undirected graph $G = (V, E)$ and we are asked to find the largest possible independent set. This problem cannot be approximated well; unless P = NP there is no $O(n^{1-\epsilon})$-approximation for the Maximum Independent Set problem for any constant $\epsilon > 0$ [91]. We note that the extremely naive algorithm that simply returns a single node is, trivially, an n-approximation.

Finally, there are optimization problems that cannot be approximated within any reasonable bound. Again, in the next section we will see that the non-metric Traveling Salesman problem cannot be approximated within any polynomial-time computable bound $f(n)$ (such as 2^n) unless P = NP. At the end of Chapter 3, we will see another problem that cannot be approximated within any such ratio unless P = NP.

1.2.7 An Integrality Gap Example

To highlight how one can study approximation algorithms using linear programming, consider the Minimum Weight Vertex Cover problem. In this problem, we are given an undirected graph $G = (V, E)$ with non-negative node weights w_v for each $v \in V$. A *vertex cover* of G is a subset $W \subseteq V$ of nodes such that $u \in W$ or $v \in W$ (or both) for each edge $e = uv \in E$. The goal is to find a vertex cover of minimum total weight. The problem is NP-hard [45].

The following approximation algorithm was presented by Hochbaum [53]. Consider the following integer programming formulation of the problem. We use a 0/1 variable x_v for each $v \in V$.

$$\begin{array}{rlcll} \min & \sum_{v \in V} w_v x_v & & & \\ \text{subject to:} & x_u + x_v & \geq & 1 & \forall e = uv \in E \\ & x_v & \in & \{0, 1\} & \forall v \in V \end{array}$$

Any vertex cover W corresponds to a feasible point x in the integer program with $x_v = 1$ for $v \in W$ and $x_v = 0$ for $v \notin W$. Conversely, any feasible point x corresponds to a vertex cover $\{v \in V : x_v = 1\}$. Thus, the cost of the optimum vertex cover equals the minimum cost of the

integer program.

The following is a simple 2-approximation for the Minimum Weight Vertex Cover problem. Relax the integer constraints in the integer program to simply $0 \leq x_v \leq 1$ for each $v \in V$. Since integer points remain feasible in this relaxation, then the minimum cost of this linear program is at most the cost of the optimum vertex cover. Solve the resulting LP relaxation of the Minimum Weight Vertex Cover problem in polynomial time (*e.g.* using the interior point method [59]) and let x^* be the solution found. Let $W = \{v \in V : x_v^* \geq \frac{1}{2}\}$. The cost of W is no more than twice the cost of the LP solution x^* (thus, at most twice the minimum cost of a vertex cover for G) because the only variables x_v that were "rounded up" to 1 were those that were already at least $\frac{1}{2}$. Also, W is indeed a vertex cover since each edge $e = uv$ has $x_u^* + x_v^* \geq 1$ so at least one of x_u^* or x_v^* is at least $\frac{1}{2}$.

Not only is this a 2-approximation for the Minimum Weight Vertex Cover problem, it also proves that the integrality gap of the LP relaxation obtained is at most 2. This upper bound on the integrality gap is essentially tight. Suppose G is a complete graph on n nodes and all nodes have weight 1. Since G is a complete graph, then the weight of the minimum vertex cover is $n - 1$. However, the point x with $x_v = \frac{1}{2}$ for each $v \in V$ is a point in the LP relaxation with objective function value $\frac{n}{2}$. So, the ratio of the optimum integer solution and the optimum fractional solution is at least $2 - \frac{2}{n}$ and we see that the upper bound of 2 is essentially tight.

1.2.8 Lower Bound Examples

In this subsection, we demonstrate an example of a reduction that establishes lower bounds on how well a problem can be approximated. In fact, essentially the same reduction can be used to demonstrate a few such hardness results.

First, consider the non-metric Travling Salesman problem. In this problem, we are given a complete graph $G = (V, E)$ with edge distances d_e for every $e \in E$. The goal is to find a Hamiltonian cycle in G with minimum total edge distance. The distinguishing feature of this problem from the classic Traveling Salesman problem is that the distances do not need to satisfy the triangle inequality. The following is well-known.

Theorem 1.2.5 *For any polynomial-time computable function $f(\cdot)$, there is no polynomial-time $f(|V|)$-approximation for the non-metric Travling Salesman problem unless P $\neq$ NP.*

Proof. Consider an instance of the Hamiltonian Cycle problem. We have a (not necessarily complete) graph $H = (V, E')$ and the goal is to determine if H has a Hamiltonian cycle or not. Form a complete graph $G = (V, E)$ on the same set of nodes as H with edge distances $d_e, e \in E$ where $d_e = 1$ if $e \in E'$ and $d_e = |V| \cdot f(|V|)$ if $e \notin E'$. Notice the running time of the reduction is polynomial since $f(|V|)$ is polynomial-time computable.

Now, if H has a Hamiltonian cycle then the same Hamiltonian cycle in G will have total edge distance $|V|$ (since every edge used is in E'). Otherwise, if H does not have a Hamiltonian cycle then

any Hamiltonian cycle of G will have total edge distance strictly greater than $|V| \cdot f(|V|)$ since any such cycle must use an edge not in E' and at least one other edge. Thus, any $f(|V|)$-approximation for the non-metric Traveling Salesman problem can be used to determine if the original graph H has a Hamiltonian cycle. □

Notice that the graph G produced by in the reduction is not metric in general. If $uv, vw \in E'$ but $uw \notin E'$ then we cannot have $d_{uw} \leq d_{uv} + d_{vw}$ if $f(|V|) > \frac{2}{|V|}$. Consider the same reduction that assigns 2 to d_e (instead of $|V| \cdot f(|V|)$) for edges $e \notin E'$. Then $d_{uv} \leq 2 \leq d_{uw} + d_{wv}$ always holds so we do have an instance of the metric Traveling Salesman problem. If H has a Hamiltonian cycle then there is a solution of cost $|V|$ and if H does not have a Hamiltonian cycle then all solutions have cost at least $|V| + 1$. Thus, the gap between these two types of instances is $1 + \frac{1}{|V|}$. This shows that even the classic metric Traveling Salesman problem does not have an FPTAS unless P = NP. More sophisticated reductions are known that rule out approximating this problem within certain small constant factors. These will be mentioned in Section 1.3.

Finally, consider the *Bottleneck Traveling Salesman* problem. In this problem we, again, have a complete graph $G = (V, E)$ with edge distances d_e. The goal is to find a Hamiltonian cycle that minimizes the *largest* distance of any edge, rather than the total distance. The reduction from Theorem 1.2.5 (with $f(|V|)$ instead of $|V| \cdot f(|V|)$ being assigned to edges not in E') establishes that the non-metric Bottleneck Traveling Salesman problem cannot be approximated within any ratio better than $f(|V|)$ unless P = NP. Also, the reduction for the metric Traveling Salesman problem (that assigns 2 for edges not in E') does show that we cannot approximate the metric Bottleneck Traveling Salesman problem within any factor strictly less than 2 unless P = NP. This result is tight as there is a 2-approximation for the metric Bottleneck Traveling Salesman problem [21]. Note that the metric Bottleneck Traveling Salesman problem is an example of a problem that is in APX but not in PTAS.

1.3 Previous Work

In this section, we will mention some of the high-profile results concerning the problems discussed in this thesis. A more detailed account of the previous work on each problem can be found in their respective chapters in this thesis. Still, to supply some context for this thesis' contributions we feel that some of the main results in previous work should be mentioned here.

Unsplittable Flow: In general, Azar and Regev showed that the Unsplittable Plow problem cannot be approximated within $\Omega(|E|^{1-\epsilon})$ for any constant $\epsilon > 0$ unless P = NP where E is the set of edges in the underlying graph [11]. Despite this strong inapproximability result, the problem remains interesting as many special cases admit better approximations. One special case is where all demands, profits, and capacities are one known as the *Edge-Disjoint Paths* problem. The Edge-Disjoint Paths problem is NP-hard in undirected graphs and remains NP-hard in directed graphs

even if there are only two tasks to consider (see Fortune *et al.* [39]). Kleinberg [61] demonstrated that the Edge-Disjoint Paths problem in both undirected and directed graphs can be approximated within $O(\sqrt{|E|})$.

An instance of the Unsplittable Flow problem is said to satisfy the *no bottleneck* assumption if $\max_i d_i \leq \max_e c_e$. That is, the maximum demand of all tasks does not exceed the minimum capacity of all edges. Note that the Edge-Disjoint Paths problem, when viewed as an instance of the Unsplittable Flow problem, satisfies the no bottleneck assumption. Chekuri *et al.* generalized Kleinberg's approach to the Edge-Disjoint paths problem to show that instances of the Unsplittable Flow problem that satisfy the no bottleneck assumption can be approximated within $O(\sqrt{|V|})$.

If the underlying graph is just a single edge, then the Unsplittable Flow problem is identical to the *Knapsack* problem if we view the single edge e as a knapsack with size c_e. Even though this, seemingly simple, restriction of the Unsplittable Flow problem is NP-hard, it does admit an FPTAS [55]. The next natural step to consider is when G is a path which we will refer to as UFP (Unsplittable Flow on paths). Instances of UFP satisfying the no bottleneck assumption can be approximated within constant factors. Currently, the best approximation ratio for UFP with the no bottleneck assumption is a $(2+\epsilon)$-approximation for any constant $\epsilon > 0$ [20]. UFP also admits a quasi polynomial-time $(1+\epsilon)$-approximation for any constant $\epsilon > 0$ if the demands of all tasks are integers that are at most quasi polynomial in the number of tasks [12]. However, this is a very strong assumption since instances of knapsack can be solved exactly in quasi polynomial time if the sizes of all items are integers at most $O(n^{\log^k n})$ for some constant k where n is the number of items. Before the results appearing in this thesis, no approximation algorithms were known for general instances of UFP.

Traveling Salesman Problems: In the classic Traveling Salesman problem, our task is to find a Hamiltonian cycle of minimum total edge distance in a symmetric metric. A classic approximation algorithm by Christofides finds Hamiltonian cycles whose cost is at most $\frac{3}{2}$ times the minimum cost Hamiltonian cycle [32]. The following linear program is an LP relaxation for the classic Traveling Salesman problem. It was first considered by Held and Karp [52].

$$\begin{aligned} \text{minimize :} \quad & \sum_{e \in A} d_{uv} x_{uv} & & (1.1) \\ \text{subject to :} \quad & x(\delta(v)) = 2 & \forall v \in V & \\ & x(\delta(S)) \geq 2 & \forall \emptyset \subsetneq S \subsetneq V & \\ & 0 \leq x_{uv} \leq 1 & \forall uv \in A & \end{aligned}$$

Wolsey [90] and Shmoys & Williamson [85] showed that the integrality gap of this relaxation is also at most $\frac{3}{2}$.

The related Traveling Salesman Path problem asks us to find a Hamiltonian path. Hoogeveen [54] demonstrated that if at most one of the endpoints of this path is specified in advance, then the

problem may still be approximated within $\frac{3}{2}$. However, if both endpoints are specified then the best algorithm known is only a $\frac{5}{3}$-approximation. Recently, this algorithm (when both endpoints are specified) was analyzed by An and Shmoys [3] and a matching bound of $\frac{5}{3}$ was placed on an LP relaxation of the problem that is similar to LP 1.1.

The Asymmetric Traveling Salesman problem is presented the same way as the classic Travling Salesman problem, except the metric is allowed to be asymmetric. An $O(\log |V|)$-approximation for this problem was presented by Frieze *et al.* [42] almost 30 years ago and this remained the best asymptotic guarantee of any approximation algorithm for the problem until very recently. Williamson [89] considered an LP relaxation of the problem and showed an alternative proof of the $O(\log |V|)$ approximation ratio of this algorithm that also bounds the integrality gap of the LP relaxation by the same ratio. Less than two years ago, an $O(\log |V| / \log \log |V|)$-approximation for the Asymmetric Traveling Salesman problem was presented by Asadpour *et.al.* [9] that also bounds the integrality gap of the LP relaxation by this ratio.

In this thesis, one variant of the Traveling Salesman problem we consider is finding a Hamiltonian path in an asymmetric metric (and some generalizations). Lam and Newman [62] first presented an $O(\sqrt{|V|})$-approximation when both endpoints are fixed, but this did not bound the integrality gap of any LP relaxation for the problem. Chekuri and Pál [30] provided the first logarithmic approximation that, again, did not bound the integrality gap of an LP relaxation. Later, Feige and Singh [38] proved that the approximability of finding minimum length Hamiltonian Cycles and Hamiltonian Paths (with both endpoints fixed) are within a factor $2 + \epsilon$ of each other for any constant $\epsilon > 0$. At the time, it implied an $O(\log |V|)$-approximation for the problem but recent improvements for cycles have improved this to $O(\log |V| / \log \log |V|)$. Again, this did not imply any bounds on the integrality gap for the problem. Finally, Nagarajan and Ravi [70] proved that the integrality gap of an analog of LP relaxation 1.1 to the case of Hamiltonian paths in asymmetric metrics (specifically, LP 3.3 from Chapter 3) was bounded by $O(\sqrt{|V|})$.

Minimum Latency: Most of the study behind Minimum Latency problems has been for the following variant in symmetric metrics. We are given a symmetric metric $G = (V, E)$ with distances on the edges and a specific start node s. The goal is to find a Hamiltonian path P starting at s that minimizes the average, over all $v \in V$, of the distances from s to v along path P. Blum *et al.* [18] exhibited the first constant-factor approximation with an approximation ratio of 72 for this variant of the Minimum Latency problem. The current best approximation guarantee is 3.59 by Chaudhuri *et al.* [25].

The variant of Minimum Latency we study in this thesis is similar to the above version, except the metric is asymmetric. Nagarajan and Ravi [70] demonstrated that this version can be approximated with $O(|V|^{\frac{1}{2}+\epsilon})$ for any constant $\epsilon > 0$. Their algorithm heavily relied on the fact that the integrality gap of an LP relaxation for the Asymmetric Traveling Salesman Path problem was $O(\sqrt{|V|})$.

1.4 New Results

Unsplittable Flow on Paths: In Chapter 2 we demonstrate that general instances of UFP (without any further restrictions on the input) can be approximated within $O(\log n)$ where n is the number of tasks. This is the first non-trivial approximation for general instances of UFP. We then generalize this approach to demonstrate that we can approximate UFP within $O(\log_d n)$ for any integer $d \geq 2$. The running time of this algorithm is $n^{O(d)}$ which means we can shrink the constant supressed in the $O(\cdot)$ notation of the approximation ratio to be arbitrarily small in polynomial time. Furthermore, this is a constant-factor approximation that runs in sub-exponential $2^{o(n)}$ time (by choosing, *e.g.*, $d = \lceil\sqrt{n}\rceil$). We also consider certain sparse instances of the problem and demonstrate that these can be approximated within constant factors in polynomial time.

Traveling Salesman Problems: Our first new contribution in Chapter 3 is an $O(\log |V|)$ approximation algorithm for the Asymmetric Traveling Salesman Path problem. While this is marginally worse than the $O(\log |V| / \log\log |V|)$-approximation that can be obtained by combining results in [38] and [9], its advantage is that it bounds the integrality gap of a natural LP relaxation by $O(\log |V|)$. This greatly improves on the previous bound of $O(\sqrt{|V|})$ in [70].

We also consider a generalization of this problem to a setting with multiple traveling salesmen. Specifically, we are given nodes s and t in an asymmetric metric as well as a positive integer k. The goal is to find k paths from s to t such that each node lies on at least one such path. Setting $k = 1$ gives the standard Asymmetric Traveling Salesman Path problem. We generalize our algorithm for $k = 1$ to arbitrary k and obtain a bicriteria approximation. For any positive integer b, our bicriteria approximation finds at most $\left(1 + \frac{1}{b}\right) \cdot k$ paths from s to t such that every node is on at least one of these paths. The total cost of these paths is at most $O(b \log |V|)$ times the optimum value of an LP relaxation for the problem of using exactly k paths. Note that setting $b = k + 1$ gives a true $O(k \log |V|)$-approximation using exactly k paths whereas setting $b = 1$ gives an $O(\log |V|)$-approximation using at most twice the number of allowed paths. As far as we know, even the case $k = 2$ with two salesmen has not been considered in asymmetric metrics and setting $b = 3$ gives a true $O(\log |V|)$ approximation for this problem. We then extend these results to different instances with multiple traveling salesmen where the start and/or end nodes are either not specified or may be different for different salesmen.

In one of these variants, we are given k pairs of nodes $(s_1, t_1), \ldots, (s_k, t_k)$. The goal is to find a path from s_i to t_i for each $1 \leq i \leq k$ such that every node lies on one such path. In the case of a symmetric metric, we give a simple 3-approximation for the problem. In asymmetric metrics, if $s_i = t_i$ then the problem can be thought of as the problem of finding a collection of cycles (including, perhaps, loops) where each cycle contains one of k nodes s_i and every node lies on one such cycle. We show that this problem can be approximated within $O(\log |V|)$. Finally, in the most general case for asymmetric metrics, we demonstrate that the problem cannot be approximated within any

(polynomial-time computable) bounded ratio unless P $=$ NP.

Minimum Latency: Our $O(\log |V|)$ bound on the integrality gap of an LP relaxation for the Asymmetric Traveling Salesman Path problem implies an $O(|V|^{\epsilon})$-approximation for the Minimum Latency problem in asymmetric metrics for any constant $\epsilon > 0$ using the framework in [70]. More specifically, the actual approximation ratio in [70] is $O(\frac{\rho+\log |V|}{\epsilon^3}|V|^{\epsilon})$ for any $\Omega(\frac{1}{\log |V|}) < \epsilon < 1$ where ρ is the integrality gap of the LP relaxation for the Asymmetric Traveling Salesman Path problem and the running time is $n^{O(\frac{1}{\epsilon})}$. So, by choosing $\epsilon = O(\frac{1}{\log |V|})$, our improved bound of $O(\log |V|)$ on ρ implies a quasi-polynomial time $O(\log^4 |V|)$-approximation. We improve on this in Chapter 4 by presenting a polynomial-time $O(\log |V|)$-approximation algorithm. This algorithm also bounds the integrality gap of a particular LP relaxation we introduce for the problem.

Finally, this thesis concludes in Chapter 5 with some directions for future research.

Chapter 2

The Unsplittable Flow Problem on Paths

Recall the definition of the Unsplittable Flow problem on a path (UFP). The underlying graph G is a simple path $v_1, v_2, \ldots, v_m$ where each edge $e = v_i v_{i+1}$ has some capacity $c_e \geq 0$. We say $v_i \prec v_j$ when $i < j$ and $e \prec e'$ when, say, $e = v_i v_{i+1}$ and $e' = v_j v_{j+1}$ and $i < j$. Similarly, we use $\preceq, \succ$ and $\succeq$ to mean, respectively, "to the left of or equal to", "strinctly to the right of", and "to the right of or equal to". Informally, if we drew the path in a "left-to-right" manner, then $v_i \prec v_j$ if v_i is drawn to the left of v_j and $e \prec e'$ if e is drawn to the left of e'. So, more generally, we say $v \prec e$ for an edge $e = uw$ if $v \preceq u$ and $e \prec v$ if $v \succeq w$. Finally, we sometimes denote the capacity of an edge $e = uv$ with c_u where $u \prec v$. That is, c_u is the capacity of the edge whose leftmost node is u.

In addition to this graph G, we have a collection of n tasks $T = \{(s_i, t_i, d_i, p_i)\}_{i=1}^n$ where $s_i \prec t_i$ are the start and end points of task i on the path, $d_i \geq 0$ is the demand of task i, and $p_i \geq 0$ is the profit of task i. In the case of a simple path, we are not concerned with how to route the tasks we select since there is a unique path between the endpoints of any task. For a task i, we let $span(i)$ denote the set of edges $\{v_i v_{i+1} : s_i \preceq v_i \text{ and } v_{i+1} \preceq t_i\}$ on the path from s_i to t_i. If a task i is selected, then it will requires d_i units of capacity along each edge $e \in span(i)$. Let $length(i)$ denote $|span(i)|$, the length of task i.

We call a subset of tasks $T' \subseteq T$ *feasible* if, for each edge e, the total demand of tasks in T' that use edge e does not exceed c_e. Formally,

$$\sum_{i \in T' : e \in span(i)} d_i \leq c_e.$$

The problem is to find a feasible subset of T with maximum possible profit. For a subset $T' \subseteq T$, we let $d(T')$ be the total demand of all tasks in T' and $p(T')$ denote the total profit of all tasks in T'. Thus, our goal is to find a subset T' that maximizes $p(T')$ while guaranteeing $d(\{i \in T' : e \in span(i)\}) \leq c_e$ for every edge e. We, furthermore, assume that every task i is feasible by itself by discarding any task i having $d_i > c_e$ for some $e \in span(i)$.

Approximating the general Unsplittable Flow problem (over arbitrary graphs) is difficult. As

mentioned earlier, Azar and Regev [11] demonstrated there is no $O(|E|^{1-\epsilon})$-approximation for the general Unsplittable Flow problem unless P = NP. Kleinberg [61] demonstrated that the special case with unit demands, capacities, and profits known as the Edge-Disjoint Paths problem (EDP) has an $O(\sqrt{|E|})$-approximation in both directed and undirected graphs. Srinivasan later showed that EDP can be approximated within $O(|V|^{\frac{2}{3}} \log^{\frac{1}{3}} |V|)$ in directed graphs [87]. The approximability of EDP is much better understood in directed graphs than in undirected graphs. Directed EDP cannot be approximated within $O(|V|^{\frac{1}{2}-\epsilon})$ for any constant $\epsilon > 0$ unless P = NP [51] whereas we only know that undirected EDP cannot be approximated within $O(\log^{\frac{1}{2}-\epsilon} |V|)$ for any constant $\epsilon > 0$ unless all problems in NP can be decided in randomized quasi-polynomial time [4] (specifically, unless NP $\subseteq$ ZPTIME$(n^{O(\text{polylog}(n))})$). Finally, we note that for any fixed constant integer k we can solve the EDP problem in undirected graphs if the number of pairs (s_i, t_i) is at most k [82]. In contrast, it is NP-complete to determine if both tasks can be routed in EDP instances in directed graphs with only two tasks [39].

EDP and, more generally, the Unsplittable Flow problem has been studied on expander graphs. When G is a (large) constant-degree graph that satisfies a strong edge expansion property, Frieze [41] demonstrates that any EDP instance on such a graph can be approximated within $O(\log |V|)$. Later, Srinivasan demonstrated that the more general instances of Unsplittable Flow with uniform capacities on expander graphs can be approximated within $O(\log^3 |V|)$ [87]. Additionally, Chakrabarti *et al.* [23] present an $O(\log^2 |V|)$-approximation for constant-degree expander graphs that satisfy the no-bottleneck assumption (defined in the next paragraph). For the expanders considered by Frieze in [41], they also develop an $O(\sqrt{\log |V|})$-approximation if the edge capacities are uniform.

A problem whose difficulty lies between EDP and general Unsplittable Flow is the Unsplittable Flow problem with the *no bottleneck* assumption. The no bottleneck assumption says that the maximum demand of all tasks is at most the minimum capacity of all edges. Instances of the Unsplittable Flow problem satisfying the no bottleneck assumption can be approximated within $O(\sqrt{|V|})$ [28] (in undirected graphs). The no bottleneck assumption has also been used to develop better approximation algorithms for restricted versions of the Unsplittable Flow problem that we discuss below.

The Unsplittable Flow problem remains NP-hard even on graphs where there is a unique path between any two nodes (namely trees). That is, the problem is simply to choose the maximum profit of a feasible subset of tasks without having to worry about how to route these tasks. As we said in Chapter 1, if the graph is a single edge then the problem is identical to the NP-hard Knapsack problem. In trees, the Unsplittable Flow problem is APX-hard as was shown by Garg *et. al.* [46]. Chekuri *et. al.* [29] demonstrated that the Unsplittable Flow problem in trees can indeed be approximated within constant factors in no bottleneck instances and, in fact, the integrality gap of a natural LP relaxation (the tree version of LP 2.1 introduced later) is bounded by a constant. Finally, after our $O(\log n)$-approximation for UFP (Unsplittable Flow on Paths) in Section 2.2 appeared [13], Chekuri *et. al.* [26] devised an $O(\log^2 n)$-approximation for general instances of Unsplittable

Flow in trees.

An instance of UFP with unit demand, capacity, and profit can be solved efficiently since it is equivalent to the maximum independent set problem on interval graphs, which can be solved in polynomial time [48]. More generally, instances of UFP with arbitrary weights and arbitrary integer capacities but unit demands can be solved exactly because LP relaxation 2.1 has the consecutive ones property meaning it is integral. Alternatively, one can model such an instance as a maximum cost circulation problem on a graph with integer capacities. Instances of UFP with arbitrary demands and uniform capacities was first shown to have a constant factor approximation by Phillips *et. al.* [75]. This was improved first by Bar-Noy *et. al.* [14] and then by Calinescu *et. al.* [20] to a $2 + \epsilon$ approximation. For the more general no bottleneck instances of UFP (with, perhaps, non-uniform capacities), Chakrabarti *et. al.* [23] presented the first constant-factor approximation which was also improved to a $2 + \epsilon$ approximation by Chekuri *et. al.* [29]. Bansal *et. al.* [12] consider a different variant of UFP. When all demands are integers at most quasi-polynomial in n, they present a quasi-polynomial $(1 + \epsilon)$-approximation for any constant $\epsilon > 0$ even if the instance does not satisfy the no bottleneck assumption.

The first non-trivial approximation for general UFP instances (that may not satisfy the no bottleneck assumption) was the $O(\log n)$-approximation algorithm appearing in Section 2.2 [13]. As we show later, LP 2.1 has an $\Omega(n)$ integrality gap for these instances. Chekuri *et. al.* [26] considered a stronger LP relaxation and demonstrate the the integrality gap of this new relaxation is $O(\log^2 n)$. In a personal communication, they have indicated that they can improve this bound to $O(\log n)$. Finally, a recent result by Bonsma *et. al.* [19] demonstrates that general instances of UFP can be approximated within $7 + \epsilon$ for any constant $\epsilon > 0$. Their approach, like ours, is a hybrid of LP rounding and dynamic programming and does not bound the integrality gap of any LP relaxation by a constant factor. We discuss their approach further in our conclusion to this chapter in Section 2.5. Bonsma *et. al.* also establish that UFP is strongly NP-hard. For a long time, the only hardness result known for UFP was the weak NP-hardness it inherits from knapsack.

Our results in Sections 2.3 and Section 2.4 were obtained after our paper [13] was published. They are now mostly subsumed by the recent constant-factor approximation by Bonsma *et. al.* [19] (except for the fact that the algorithm in Section 2.4 also bounds an integrality gap). The results of these sections were obtained after papers [13] and [26] appeared, but long before [19] was made public.

The main results of this chapter are the following.

Theorem 2.0.1 *For any integer $d \geq 2$ (perhaps a function of n), UFP can be approximated within $O(\log_d m)$ in time $n^{O(d)}$.*

We argue that we can assume that $m \leq 2n$ in any UFP instance, so this is also an $O(\log_d n)$-approximation.

Say an instance is *q-conflicting* if for every task i and every edge e spanned by i, there are at most q tasks $i' \neq i$ for which $e \in span(i')$ and $d_i + d_{i'} > c_e$. That is, there are at most q other tasks i' such that $\{i, i'\}$ is not feasible because the capacity of edge e is violated. The motivation for studying this case is discussed in section 2.4.

Theorem 2.0.2 *There is a polynomial-time $O(q)$-approximation algorithm for q-conflicting UFP instances.*

A good place to start looking for approximation algorithms is a linear programming (LP) relaxation of the problem. For each task i, define a variable x_i. Consider LP 2.1 which is, perhaps, the most natural linear programming relaxation for UFP.

$$\begin{aligned} \text{maximize :} \quad & \sum_i p_i x_i & & (2.1) \\ \text{subject to :} \quad & \sum_{i:e \in span(i)} d_i x_i \leq c_e & \forall \text{ edges } e \\ & 0 \leq x_i \leq 1 & \forall \text{ tasks } i \end{aligned}$$

To see this is a valid LP relaxation of UFP, consider some feasible subset $T' \subseteq T$. Define a point x in the polytope of linear program 2.1 by $x_i = 1$ for $i \in T'$ and $x_i = 0$ for $i \notin T'$. Since T' is feasible, then the capacity constraints are satisfied. The value of the objective function under this assignment is also the same as the profit of T'. Conversely, if we have an integer point x in the polytope, then we define $T' = \{i \in T : x_i = 1\}$. We see that T' is feasible with profit equal to the value of the objective function at point x.

Unfortunately, the integrality gap of this LP relaxation is large so we cannot solely rely on it when designing approximation algorithms.

Lemma 2.0.3 *There are instances of UFP with uniform profit whose optimum value is 1 such that the LP relaxation of the problem has value at least $\frac{n}{2}$.*

Proof. Consider the following instance with n tasks on a path having $n + 1$ nodes. Identify the tasks with integers $1, \ldots, n$ and the nodes on the path with integers $0, \ldots, n$. Then task i has $s_i = 0, t_i = i, d_i = 2^{-i}, p = 1$. Furthermore, the capacity c_j of the edge from point $j - 1$ to j is 2^{-j}. See Figure 2.1 for an illustration.

The optimum solution is 1 because no two tasks can be chosen simultaneously. To see this, consider tasks $i < i'$. Their total demand is $2^{-i} + 2^{-i'} > 2^{-i}$. Both tasks i and i' cross the edge connecting $i - 1$ to i which has capacity 2^{-i} so they cannot both be chosen simultaneously.

On the other hand, the variable assignment $x_i = 1/2$ has objective function value $n/2$ in LP 2.1. Consider an edge from $j - 1$ to j. The LP constraint for this edge has the left-hand side being

$$\sum_{i=j}^{n} d_i x_i = \sum_{i=j}^{n} \frac{2^{-i}}{2} < 2^{-j} = c_e$$

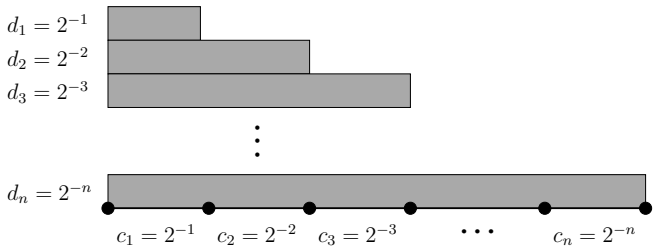

Figure 2.1: An instance of UFP on paths with integrality gap $\Omega(n)$.

so the LP has optimum value at least $n/2$. □

To address the difficulty presented by the bad integrality gap example, the set of tasks will be split into two instances. One instance can be handled well with LP rounding and the other, which may have a bad integrality gap, will be handled with dynamic programming. We define these two groups as follows:

Definition 2.0.4 *A* bottleneck edge *for a task i is any edge $e \in span(i)$ with $d_i > c_e/2$. If a task i has no bottleneck edge then it is* slack*, otherwise it is* tight*.*

We partition the tasks T into the slack tasks and the tight tasks. Suppose we have an α-approximation algorithm for finding the optimum feasible subset among the slack tasks and a β-approximation algorithm for finding the optimum feasible subset among the tight tasks. Returning the more profitable of these two solutions is then a $2 \cdot \max\{\alpha, \beta\}$-approximation. To see this, suppose T^* is an optimum feasible subset of T and that T_s, T_t are, respectively, the slack and tight tasks in T. Then $p(T^*) = p(T^* \cap T_s) + p(T^* \cap T_t)$. Say T'_s, T'_t are, respectively, the solutions found by the α-approximation for slack tasks and the β-approximations for tight tasks. The total profit returned is:

$$\begin{aligned}
\max\{p(T'_s), p(T'_t)\} &\geq \max\{p(T^* \cap T_s)/\alpha, p(T^* \cap T_t)/\beta\} \\
&\geq \frac{p(T^* \cap T_s)/\alpha + p(T^* \cap T_t)/\beta}{2} \\
&\geq \frac{p(T^* \cap T_s) + p(T^* \cap T_t)}{2\max\{\alpha, \beta\}} \\
&= \frac{p(T^*)}{2\max\{\alpha, \beta\}}
\end{aligned}$$

Of interest to our algorithm is the following special case.

Lemma 2.0.5 *If there is a constant-factor approximation for instances of UFP when all tasks are slack and a β-approximation for instances of UFP when all tasks are tight, then there is an $O(\beta)$-approximation for general instances UFP.*

In fact, this is a general theme of the algorithms that follow. In many cases, we partition the instance into different parts, approximate the optimum feasible subset in each part, and keep the

subset of maximum total profit among all parts. If γ is the worst approximation ratio among the different parts and there are k parts, then the profit of the returned solution is within a $k\gamma$ factor of the optimum feasible subset of tasks for the instance being considered (*i.e.* a $k\gamma$-approximation).

2.1 Simplifying Assumptions

Since the approximation ratios presented in this chapter are usually super constant, we make some simplifying assumptions at the expense of a constant factor in the approximation ratios. The few constant factor approximations that are presented are more proofs of concept that we can do better than an $O(\log n)$-approximation in certain cases and we also do not worry about optimizing these constants. In this section, we show how to reduce a general instance of UFP to instances satisfying certain assumptions while losing only a constant factor. In subsequent sections, we show how to exploit these assumptions when developing approximation algorithms.

Informally, using these simplifications we can restrict our attention to instances whose tasks have large demands (relative to the edges they span) and small profits and such that any two tasks either have similar demands or the demand of one is much larger than the demand of the other. We also show that we can assume the length of the underlying path is not much longer than the number of tasks.

The optimum value of the slack tasks is easy to approximate. The following was proven in [26] using techniques from [20]. It was proven after our result in Section 2.2 appeared [13], but before we developed the more general approximation algorithm in Section 2.3. We use it in our algorithm for Section 2.3, but we will provide an explicit proof of it in the special case considered in Section 2.2. The proof is similar to our proof in [13].

Lemma 2.1.1 (Chekuri, Korula, and Ene [26]) *The integrality gap of linear program 2.1 is constant if all tasks are slack.*

So, unless explicitly stated, all tasks in the rest of this chapter are assumed to be tight. Another assumption we work with allows us to use dynamic programming approaches.

Lemma 2.1.2 *If there is a polynomial-time β-approximation for tight instances of UFP where all profits are integers in the range $[0, 2n]$, then there is a polynomial-time 2β-approximation for tight instances of UFP with arbitrary profits.*

Proof. This is the standard trick of scaling. Our proof follows the presentation of Lemma 8.3 in [88] for approximating the Knapsack problem. Note that we could have assumed the values are in the range $[1, n/\epsilon]$ and obtained a $\beta/(1-\epsilon)$ approximation for any constant $\epsilon > 0$, but our final approximation ratios are at least constant factors so we consider only $\epsilon = 1/2$ for simplicity.

Let $P = \max_i p_i$ be the maximum profit of any task and let $K = \frac{P}{2n}$. Define new profits $p'_i := \lfloor \frac{p_i}{K} \rfloor$ where $\lfloor x \rfloor$ is the greatest integer not exceeding x. Note that all of the remaining tasks

have p'_i being an integer in the range $[0, 2n]$. Use the β-approximation algorithm on the instance with profits p'_i and say T' is the subset of tasks found. Since the demands and capacities do not change, the returned instance is also feasible for the original instance.

Suppose T^* is the optimum subset of T under profits p. Since $|T^*| \leq n$ and since $p_i \geq Kp'_i \geq p_i - 1$, then $Kp'(T^*) \geq p(T^*) - Kn$. Since T^* is a feasible solution with value at least $p(T^*)/K - n$ with respect to profits p', the value of T' with respect to profits p' is at least $(p(T^*)/K - n)/\beta$. So, the total profit of T' under the original profits p is at least

$$(p(T^*) - Kn)/\beta = (p(T^*) - P/2)/\beta \geq p(T^*)/2\beta$$

where we observe $p(T^*) \geq P$ since any task i with $p_i = P$ is feasible by itself. □

We coarsely organize the tasks according to their demands. By scaling the demands and capacities uniformly, we may assume each demand d_i is at least 1. Define $D_k = \{i \in T : 2^k \leq d_i < 2^{k+1}\}$. By losing a factor of 2 in the approximation ratio, we further partition T into two sets T_{even}, T_{odd} where $T_{even} = \cup_{k \geq 1} D_{2k}$ and $T_{odd} = \cup_{k \geq 1} D_{2k-1}$ and run the subsequent algorithms on each subset T_{even}, T_{odd}, returning the better of the two solutions. The benefits of doing this will be made apparent later.

Finally, we may assume that $m \leq 2n$ where m is the number of nodes on the underlying path and n is the number of tasks. If either the start node or the end node on the path is not an endpoint of some task, then that node may be removed. Also, if some internal node j with incident edges, say, e, e' is not the start or end point of some task, then we may remove j and combine the edges e, e' into one edge with capacity $\min\{c_e, c_{e'}\}$. It is easy to see that a subset $T' \subseteq T$ is feasible before such an update if and only if it is feasible after such an update.

To summarize, we are assuming the following structure on the instance (at the expense of losing a constant factor in the approximation ratio).

Definition 2.1.3 *Say an instance of UFP is* simplified *if the following statements are true.*

1. *All tasks are feasible on their own.*
2. *All tasks are tight.*
3. *The profit of each task is an integer in the range* $[0, 2n]$.
4. *We have either* $D_k = \emptyset$ *for all even integers* k *or* $D_k = \emptyset$ *for all odd integers* k.
5. *The number of nodes* m *on the underlying path is at most* $2n$ *(twice the number of tasks).*

Unless stated otherwise, we assume throughout the rest of the chapter that all UFP instances are simplified. The only exception is that we briefly discuss slack tasks in Section 2.2.2.

Combining the observations in this section, we have the following

Theorem 2.1.4 *If there is a polynomial-time* β*-approximation for simplified instances of UFP, then there is a polynomial-time* $O(\beta)$*-approximation for general instances of UFP.*

2.2 A Logarithmic Approximation for UFP

Eventually we will present an $O(\log_d n)$-approximation running in time $n^{O(d)}$ for any integer $d \geq 2$. However, the ideas involved can get quite technical. Here, we present a simpler algorithm that achieves an $O(\log n)$ approximation in polynomial time to highlight some of the main ideas of the more general result without getting too involved in the technical details.

The basic structure of the algorithm has two main parts. First, we argue that we can restrict our attention to instances called *intersecting* where all tasks span a common edge e. Then, we present a constant-factor approximation for such instances. However, there are a number of cases to be considered for intersecting instances based on whether the tasks have bottleneck to the left of e, to the right of e, or both. We need to develop different algorithms for these cases. Though we may assume that all tasks are tight due to result in [26], we also demonstrate how to approximate intersecting instances of slack tasks since this is much simpler than the general case of non-intersecting slack tasks. We do this to demonstrate some of the techniques in approximating slack tasks. We also briefly demonstrate how to use an approximation algorithm for UFP to approximate instances of unsplittable flow when the underlying graph is a cycle.

2.2.1 A Reduction to Intersecting Cases

Two tasks *intersect* if they share a common edge. That is, i and j intersect if $span(i) \cap span(j) \neq \emptyset$. A collection of tasks is *intersecting* if all of the tasks share a common edge; this is equivalent to the property that the tasks in the collection pairwise intersect. This is similar to the fact that the intervals corresponding to a clique in an interval graph must share a common point (*e.g.* [48]). Finally, a collection of tasks C is *feasible* if, for all edges e, we have $\sum_{i \in C: e \in span(i)} d_i \leq c_e$.

We now describe a reduction procedure which allows us to focus only on intersecting cases while losing an $O(\log n)$-factor in the approximation guarantee. This is the only place in our algorithm where more than a constant factor is lost in the approximation guarantee.

Lemma 2.2.1 *If there is a ρ-approximation for instances of UFP on a line where all tasks intersect, then there is an $O(\rho \log n)$-approximation for the general instance of UFP on a line.*

Proof. Consider an instance of UFP on a line. We first group the tasks according to their lengths. Say a task i belongs to group G_r if $2^r \leq length(i) < 2^{r+1}$. Since we have $length(i) < m \leq 2n$ for all tasks i, then $r \in \{0, 1, \ldots, \lceil \log_2 2n \rceil - 1\}$. Focus on an optimum set of feasible tasks T^* with profit OPT. Note that one of the groups G_r must have at least a $\frac{1}{\lceil \log_2 2n \rceil}$-fraction of the total profit of T^*. That is, if OPT_r is the optimum profit over all feasible subsets of tasks in group G_r, then $OPT_r \geq \frac{OPT}{\lceil \log_2 2n \rceil}$ for some r.

Consider a group G_r, each task $i \in G_r$ must span some edge $e = v_b v_{b+1}$ where $b = k2^r$ for some integer k. Create groups $H_{r,k}$ for $k \in \mathbb{Z}$ and place $i \in G_r$ in group $H_{r,k}$ if k is the *least* integer for which $v_b v_{b+1} \in span(i)$ for $b = k2^r$. One sees that $H_{r,k}$ is an intersecting collection of tasks

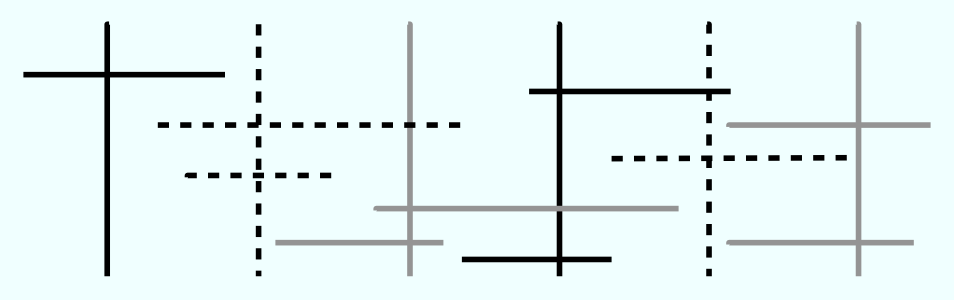

Figure 2.2: Grouping the tasks according to the left-most point of the form $k2^r$ for some integer k.

(Figure 2.2 helps illustrate this). Observe that for tasks $i \in H_{r,k}$ and $j \in H_{r,l}$ with $k+3 \leq l$ we have $span(i) \cap span(j) = \emptyset$. This follows since $length(i) < 2^{r+1}$ and $s_i \leq k2^r$ imply $t_i < (k+2)2^r$. Furthermore, since l is the least integer for which $s_j \leq l2^r$ then $s_j \geq (k+2)2^r > t_i$.

Now, apply the ρ-approximation to each $H_{r,k}$ and let $C_{r,k}$ denote the collection of tasks chosen by the algorithm. For each $l = 0, 1, 2$, let $\mathcal{C}_{r,l}$ be the union of all $C_{r,l'}$ with $l' \equiv l \pmod 3$. By arguments in the previous paragraph, none of the tasks in $C_{r,k}$ can intersect any task in $C_{r,l}$ if $k+3 \leq l$ so $\mathcal{C}_{r,l}$ is a feasible collection of tasks for each $l = 0, 1, 2$. Furthermore, by looking at the restriction of the optimum solution OPT_r for group G_r to the subgroups $H_{r,k}$, we see that at least one of the three groups $\mathcal{C}_{r,l}$ has profit at least $\frac{OPT_r}{3\rho}$. Thus, for some $r \in \{0, 1, \ldots, \lceil \log_2 2n \rceil - 1\}$ and some $l \in \{0, 1, 2\}$ we have the total profit of tasks in $\mathcal{C}_{r,l}$ is at least $\frac{1}{3\rho\lceil \log_2 2n \rceil} \cdot OPT = \Omega\left(\frac{1}{\rho \log n}\right) OPT$. □

In the next section, we develop a constant-factor approximation for instances of UFP on a line where all tasks are intersecting. Combining this with the Lemma 2.2.1 and Theorem 2.1.4 yields the following:

Theorem 2.2.2 *There is an $O(\log n)$-approximation for UFP.*

Consider an instance of UFP where all tasks share a common edge t. We refine our classification of the tasks in the following way:

1. if no edge is a bottleneck for i then say i is *slack*
2. if i has bottleneck edges on both sides of t then say i has *both endpoints tight*
3. if $k \preceq t$ for all bottleneck edges k for i then say i is *left-tight*
4. if $k \succ t$ for all bottleneck edges k for i then say i is *right-tight*

Since the instances are simplified (*cf.* Definition 2.1.3), we can exclude the case of slack tasks. However, there is a relatively simple LP rounding mechanism for intersecting cases of slack tasks that we describe in the following section for the sake of completeness. It is easy to verify that none

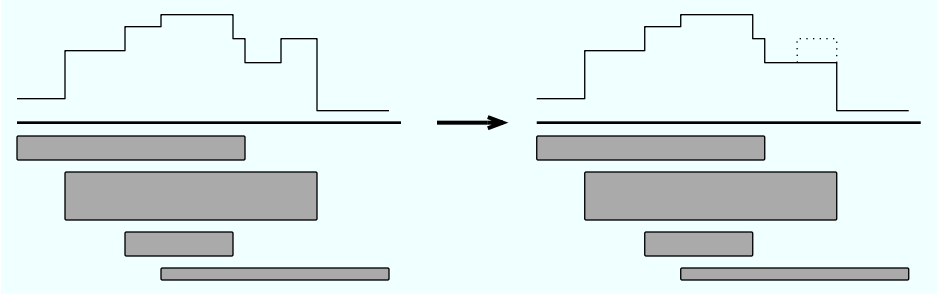

Figure 2.3: Illustrating why we may assume the capacity profile is unimodal. The capacity profile is drawn above the line and the tasks are drawn below the line.

of the steps made so far rely on the tasks being tight, so we may assume, for this algorithm only, that the input also contains slack tasks.

Partition the tasks into four groups according to the classification above. We describe a constant-factor approximation for each such group in the next three subsections (the algorithm for right-tight tasks is essentially identical to the algorithm for left-tight tasks so it is omitted). The maximum total profit of these four approximate solutions is then within a constant factor of the optimum solution (since one of these groups has a solution consisting of at least 1/4 of the optimum profit).

There is one further simplification we can apply to intersecting cases. If all tasks share a common point t, then we may assume the following structure on the capacities. For each $e \prec e' \preceq t$, we have $c_e \leq c_{e'}$ and for each $t \succeq e' \succ e$ we have $c_{e'} \geq c_e$. That is, the capacities increase as we move from the left to t and decrease as we move from t to the right. The reason is this: in any feasible collection of tasks T and any for $e \prec e' \preceq t$ we have that the total demand in T at edge e' is at least the total demand in T at edge e. By this reasoning, we may reduce the value c_e to $c_{e'}$ and not worry about affecting feasibility of any subset of tasks. If the capacities satisfy this, then we say the capacity profile is *unimodal*. See Figure 2.3 for an illustration.

2.2.2 Slack Tasks

The following rounding algorithm is similar to the algorithm of Chakrabarti et al. [23] for the no-bottleneck case. Though the slack tasks in our case may not satisfy this assumption, essentially the same rounding algorithm can be seen to provide a constant factor approximation for intersecting cases of slack tasks. For the sake of completeness, and to get an idea how we can deal with slack tasks, we present the full algorithm and proof in the case of intersecting slack tasks. The analysis is simpler in our case because the tasks are intersecting.

Recall the standard LP for UFP.

$$\begin{aligned} \text{maximize :} \quad & \sum_i p_i x_i \\ \text{subject to :} \quad & \sum_{i:e \in span(i)} d_i x_i \leq c_e && \forall \text{ edges } e \\ & 0 \leq x_i \leq 1 && \forall \text{ tasks } i \end{aligned}$$

Though it has an $\Omega(n)$ integrality gap in general cases, we will prove it has an $O(1)$-integrality gap for intersecting cases of slack tasks. Chekuri *et. al.* proved it has an $O(1)$-integrality gap in general slack cases that are not necessarily intersecting [26]. From now on, let x^* denote an optimum solution to the above LP.

Consider the following algorithm. Since the capacity profile is unimodal, the minimum capacity of all edges spanned by a task i is across either edge starting at s_i or across the edge ending at t_i. Let $C_{\leq}$ be the set of tasks with $c_{s_i} \leq c_{t_i-1}$ and let $C_{>}$ be the set of tasks with $c_{s_i} > c_{t_i-1}$. That is, $C_{\leq}$ is the collection of tasks whose most constrained edge is its start edge and $C_{>}$ is the collection of tasks whose most constrained edge is its final edge.

The rounding algorithm proceeds as follows. We first ignore the tasks in $C_{>}$ and focus only on tasks in $C_{\leq}$. The algorithm for rounding tasks in $C_{>}$ is similar to what follows so it is omitted. Next, order the tasks in $C_{\leq}$ in increasing order of their starting nodes s_i. We choose each task $i \in C_{\leq}$ independently with probability $x_i^*/4$. Let R denote the set of chosen tasks and say these tasks are $i_1 \leq i_2 \leq \cdots \leq i_{|R|}$. We construct a sequence of sets $\emptyset = S_0, S_1, \ldots,$ as follows: let $S_r = S_{r-1} \cup \{i_r\}$ if $S_{r-1} \cup \{i_r\}$ is feasible; or let $S_r = S_{r-1}$ otherwise. The algorithm outputs the set $S = S_{|R|}$.

Note that S is a random set, and the decision whether task i lies in S or not is correlated to whether other tasks lie in S or not. We show that:

Theorem 2.2.3 *Any particular request $i \in C_{\leq}$ lies in S with probability at least $x_i^*/8$.*

Proof. Define the following random variables: for $i \in C_{\leq}$, let $X_i = 1$ if $i \in R$, and 0 otherwise; and let $Y_i = 1$ if $i \in S$, and 0 otherwise. Note that X_i variables are independent, but the Y_i variables are not.

Fix $1 \leq r \leq |R|$ and consider the task $i = i_r$. We are interested in $E[Y_i]$. Since $S \subseteq R$, we have $Y_i \leq X_i$ and hence $E[Y_i] \leq E[X_i]$. Consider the event E_r that $[Y_i = 0 \mid X_i = 1]$. If E_r happens, then it must be the case that $S_{r-1} \cup \{i\}$ is not feasible. The lemma below characterizes the reason E_r happens.

Lemma 2.2.4 *The event E_r holds if and only if the capacity of the first edge c_{s_i} in $span(i)$ is violated by the set of tasks $S_{r-1} \cup \{i\}$.*

Proof. The proof is based on the fact that the capacity profile is unimodal and the defining property of the tasks in $C_{\leq}$. By definition, E_r happens if and only if the capacity constraint at some edge

$e \in span(i)$ is violated by $S_{r-1} \cup \{i\}$. Note that the order in which we considered the tasks implies all tasks in $S_{r-1} \cup \{i\}$ cross the first edge c_{s_i} of i. Since these tasks are in $C_{\leq}$ and the capacity profile is unimodal, this edge has the least capacity among all edges in $span(i)$. Thus, if $S_{r-1} \cup \{i\}$ violates the capacity of some edge and since S_{r-1} is feasible, then surely it must violate the capacity of c_{s_i}. □

Thus, for E_r to hold, the total demand of tasks in $R \cap \{1, \ldots, i-1\}$ must exceed $c_{s_i} - d_i$. For $j = 1, \ldots, i-1$, consider a random variable $D_j = d_j$ if $j \in R$, and 0 otherwise. Let $D = \sum_{j=1}^{i-1} D_j$.

Lemma 2.2.5 $Pr[E_r] \leq 1/2$

Proof. We know that $Pr[E_r] \leq Pr[D \geq c_{s_i} - d_i] \leq Pr[D \geq c_{s_i}/2]$. The second step follows as all tasks are slack.

We have $E[D] = \sum_{j=1}^{i-1} E[D_j] = \sum_{j=1}^{i-1} x_j^*/4 \leq c_{s_i}/4$. The last inequality holds since the fractional solution x^* satisfies the capacity constraint on edge s_i. Thus, by Markov's inequality, $Pr[D \geq c_{s_i}/2] \leq 1/2$. □

Now,

$$\begin{aligned} E[Y_i] &= Pr[Y_i = 1 \mid X_i = 1] \cdot Pr[X_i = 1] + Pr[Y_i = 1 \mid X_i = 0] \cdot Pr[X_i = 0] \\ &= Pr[Y_i = 1 \mid X_i = 1] \cdot Pr[X_i = 1] \\ &= (1 - Pr[E_r]) \cdot x_i^*/4 \\ &\geq x_i^*/8 \end{aligned}$$

as claimed. □

Say that $S_{\leq}$ and $S_{>}$ are, respectively, the subsets of $C_{\leq}$ and $C_{>}$ found through the above rounding algorithm. If z^* is the value of the LP solution for the slack tasks, using Theorem 2.2.3 and returning the most profitable of $S_{\leq}$ and $S_{>}$, the expected value of the solution obtained is at least $z^*/16$.

2.2.3 Both Endpoints Tight

As a warm-up, consider the special case where the tasks form a sequence of nested intervals. That is, say the tasks can be ordered such that $s_i \preceq s_j \preceq t_j \preceq t_i$ for all $i \leq j$. The following notation is useful in this case and in the more general cases of tight tasks to be considered. For a task i, let $cap(i) = \min_{e \in span(i)} c_e$ denote the minimum capacity over all edges in the span of i.

Theorem 2.2.6 *There is a polynomial time exact algorithm for nested instances if all profits are integers in the range* $[0, 2n]$.

Proof. The algorithm is based on a dynamic programming algorithm similar to those used for Knapsack problems. For integers i, p, let $f(i, p)$ be the minimum total demand among feasible

subsets of tasks $S \subseteq \{1, \ldots, i\}$ that achieve profit exactly p. If it is not possible to obtain profit exactly p with a feasible subset of the first i tasks then say $f(i,p) = \infty$. The values of $f(i,p)$ are computed in the order of increasing i. Clearly $f(0,0) = 0$ and $f(0,p) = \infty$ for $p > 0$. We claim the following recurrence is satisfied by the values $f(i,p)$ for $i > 0$:

$$f(i,p) = \begin{cases} \min\{f(i-1,p), f(i-1,p-p_i) + d_i\} & \text{if } p_i \leq p \\ & \text{and } f(i-1,p-p_i) + d_i \leq cap(i) \\ f(i-1,p) & \text{otherwise} \end{cases}$$

To see this, consider some $i > 0$ and profit p. If $f(i,p) = \infty$ then surely $f(i-1,p) = \infty$. Furthermore, if $p_i \leq p$ and $f(i-1,p-p_i) < \infty$ then we claim that $f(i-1,p-p_i) + d_i > cap(i)$. If this were not so, then consider some feasible set S' of the first $i-1$ tasks with minimum possible demand with profit exactly $p - p_i$. By definition, all $e \in span(i)$ have $c_e \geq cap(i)$. Thus, if $f(i-1,p-p_i) + d_i \leq cap(i)$ then $S' \cup \{i\}$ is a feasible subset of the first i tasks obtaining profit p which contradicts $f(i,p) = \infty$. Therefore, the recurrence is satisfied for pairs (i,p) with $i > 0$ for which $f(i,p) = \infty$.

On the other hand, suppose $f(i,p) < \infty$. Consider some set S of the first i tasks with minimum possible demand that obtains profit exactly p (*i.e.* the demand of S is $f(i,p)$). If $i \notin S$ then $f(i-1,p) \leq f(i,p)$ since S is also a feasible set of the first $i-1$ tasks. On the other hand, $f(i-1,p) \geq f(i,p)$ since any subset of the first $i-1$ tasks is also a subset of the first i tasks which implies $f(i-1,p) = f(i,p)$. If $i \in S$ then $S \setminus \{i\}$ is a feasible subset of the first $i-1$ demands so $f(i-1,p-p_i) \leq f(i,p) - d_i$. If $f(i-1,p-p_i) < f(i,p) - d_i$, then by reasoning in a manner similar to the previous paragraph, any feasible set S' of profit $p - p_i$ of the first $i-1$ tasks with demand $f(i-1,p-p_i)$ can be extended to a feasible set $S' \cup \{i\}$ of the first i elements with profit p and demand $f(i-1,p-p_i) + d_i < f(i,p)$ which is a contradiction. Therefore, $f(i-1,p-p_i) + d_i = f(i,p)$. In either case of $i \in S$ or $i \notin S$, the recurrence is satisfied.

The value of the optimum solution is then the largest value p for which $f(n,p) < \infty$. Since the instances are simplified according to Definition 2.1.3, the only values of p which may be finite are integers in the range $[0, 2n^2]$. Since i ranges from 0 to n, then all values $f(i,p)$ can be computed with dynamic programming in time $O(n^3)$. □

More generally, a collection of tasks with both endpoints tight can be made to look something like a sequence of nested intervals. Recall that for any non-negative integer k we defined D_k to be the collection of tasks i whose demand d_i lies in the range $[2^k, 2^{k+1})$. We have the following structure between groups D_k which says if task i has much less demand than task j then task j is nested in task i. The basic idea is that task j, being feasible on its own, cannot cross any bottleneck edge for task i since the demand for j is much higher than the demand of i while any bottleneck edge for i has capacity close to the demand of i.

Lemma 2.2.7 *If $i \in D_k$ and $j \in D_l$ with $k + 2 \leq l$ then $s_i \prec s_j$ and $t_i \succ t_j$.*

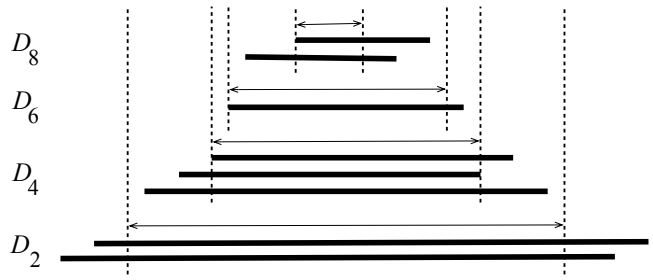

Figure 2.4: A sketch of the structure exploited by the dynamic programming. Thick lines are tasks in a feasible solution (with the corresponding demand class written to the left of the image) which is why each demand class has only three tasks shown. Pairs of dotted lines connected by a thin, double-arrowed line indicate the last start node and the first end node among all tasks in the corresponding demand class. The edges spanned by tasks of any higher demand class must be contained between these two dotted lines.

Proof. Since $d_i < 2^{k+1}$ and $2^l \leq d_j$, then $2d_i < d_j$ (by $k+2 \leq l$). Also, since i is tight and the capacity profile is unimodal, then $d_i > c_{s_i}/2$ which shows $d_j > c_{s_i}$. Since j is feasible by itself and $d_j > c_{s_i}$, then $s_i \prec s_j$. A similar argument shows $t_i \succ t_j$. □

Recall that since the instances are simplified according to Definition 2.1.3 then we have either $D_k = \emptyset$ for all even k or $D_k = \emptyset$ for all odd k. We also have the following observation that bounds the size of a feasible subset of any D_k. Since all tasks share a common point, then a bottleneck edge (say e) of some task (say i) must be spanned by all other tasks. Since the capacity of e is close to d_i and the demands of other $j \in D_k$ are close to d_i, then not too many tasks in D_k can fit across edge e in a feasible solution. Formally:

Lemma 2.2.8 *Let B_k be any feasible subset of D_k. Then $|B_k| \leq 3$.*

Proof. Let $i \in B_k$ be such that $s_i \succeq s_j$ for all $j \in B_k$. Notice that $s_i \in span(j)$ for all $j \in B_k$. Now, by definition of D_k we have $2d_j \geq d_i$ for all $j \in B_k$. Furthermore, since i is tight we have $d_i > c_{s_i}/2$. Therefore, the total demand in B_k at edge s_i exceeds $|B_k|c_{s_i}/4$. Since B_k is feasible, then the total demand crossing s_i must be at most c_{s_i}. Therefore, $|B_k| \leq 3$. □

Lemmas 2.2.7 and 2.2.8 lead to a dynamic programming solution. We build a table $f(k,p)$ that is the minimum total demand of a feasible subset of tasks in $D_k \cup D_{k-2} \cup D_{k-4} \cup \ldots$ that has profit exactly p. To build the $f(k,p)$ values from the $f(k-2,p)$ values, we try adding subsets of D_k of size at most 3. By Lemma 2.2.7, each task in D_k is contained in the span of every task in D_{k-2} which resembles the property that the tasks form a nested sequence of intervals. Figure 2.4 illustrates some of these ideas.

Extend the definition of $f(k,p)$ to include $k = -1$ and $k = -2$ which should simply read as $f(k,0) = 0$ and $f(k,p) = \infty$ for $p > 0$ whenever $k = -1$ or -2 (in other words, we can only obtain a profit of 0 if no tasks are chosen).

Let $A(k,p)$ be the collection of all subsets S of D_k of size at most 3 with $p(S) \leq p$ and the following additional property. For any feasible subset of tasks T' with total profit $p(T') = p - p(S)$ and total demand $d(T') = f(k-2, p-p(S))$ we have $d(T') + \sum_{i \in S: e \in span(i)} d_i \leq c_e$ for each edge e contained in the common intersection of the spans of all tasks in classes $D_l, l < k$. Intuitively, a set S in $A(k,p)$ is one that can extend any minimum set T^* corresponding to $f(k-2, p-p(S))$ to a feasible solution $S \cup T^*$. We only have to verify the capacity constraints are satisfied for those edges in the common intersection of all tasks in some lower class $D_l, l+2 \leq k$. Again, by Lemma 2.2.7 this is because the span of each task in D_k is completely contained in the span of each task in some $D_l, l+2 \leq k$. Lemma 2.2.8 essentially says we can restrict our attention to small subsets of D_k since any subset larger than 3 is not feasible on its own.

Formally, the recurrence looks like:

- $f(k,0) = 0$ for $k \in \{-1, -2\}$
- $f(k,p) = \infty$ for $k \in \{-1,-2\}, p > 0$
- $f(k,p) = \infty$ for $k \geq 0$ if $A(k,p) = \emptyset$ (note that can only happen if $A(k-2,p) = \emptyset$)
- $f(k,p) = \min_{S \in A(k,p)} f(k-2, p-p(S))) + d(S)$ for $k \geq 0, A(k,p) \neq \emptyset$

Lemma 2.2.9 *The recurrence correctly relates the values of* $f(k,p)$.

Proof. The base cases with $k < 0$ are clearly correct (when interpreted as suggested above). Now, consider some $k \geq 0$ and profit p. If $f(k,p) < \infty$ then let $S = S' \cup B$ be a subset of $D_k \cup D_{k-2} \cup \ldots$ obtaining profit p with total demand $f(k,p)$ and $S \cap D_k = B$. We first verify that $B \in A(k,p)$. By Lemma 2.2.8, we know $|B| \leq 3$ (in fact, B may be empty). Furthermore, we also clearly have $p(B) \leq p$. Finally, since S' is a feasible subset of $D_{k-2} \cup D_{k-4} \cup \ldots$ then $f(k-2, p-p_B) \leq f(k,p) - d_B$. By Lemma 2.2.7, any optimum set S^* with profit $p - p_B$ and demand $f(k-2, p-p_B)$ places less demand across each edge spanned by B than set S'. For such a set S^* we have $S^* \cup B$ being feasible. Therefore, $B \in A(k,p)$.

In fact, we actually have $d(S^*) = d(S')$ for S^* an optimum set corresponding to $f(k-2, p-p_B)$. Indeed, if $d_{S^*} < d_{S'}$ then the feasible set $B \cup S^*$ has demand strictly less than $B \cup S'$ and profit p which contradicts that $d_S = d_{B \cup S'} = f(k,p)$. Therefore, any subset S of $D_k \cup D_{k-2} \cup \ldots$ with profit p and demand $f(k,p)$ has $S \cap D_k \in A(k,p)$ and $S \setminus D_k$ being an optimum subset of $D_{k-2} \cup D_{k-4} \cup \ldots$ with profit $p - p(B)$ and demand $f(k-2, p-p(B))$ so the recurrence correctly determines $f(k,p)$ in this case.

On the other hand, if $f(k,p) = \infty$ then $A(k,p) = \emptyset$. This is because any $B \in A(k,p)$ is such that $f(k-2, p-p(B)) + d(B) \leq c_j$ for all tasks j in the common intersection of tasks in $D_{k-2} \cup D_{k-4} \cup \ldots$. Thus, by definition we would be able to extend any such set of tasks with demand $f(k-2, p-p(B))$ to a feasible set of tasks obtaining profit p with demand $f(k,p)$. $\square$

Theorem 2.2.10 *There is an exact algorithm for simplified instance of UFP when the tasks are also intersecting and tight on both sides.*

Proof. As in Theorem 2.2.6 the highest p for which $f(k,p) \neq \infty$ is the optimum profit. By Lemma 2.2.9, we can compute the values $f(k,p)$ using dynamic programming (notice the recurrence for a given pair (k,p) only refers to pairs (k',p') for which $k' < k$).

The total profit is $O(n^2)$ and the profit of any subset of tasks is an integer. The number of integers k for which $D_k \neq \emptyset$ is also at most n. Therefore, the total number of $f(k,p)$ entries that need to be considered is $O(n^3)$. For each k and each p, we have $|A(k,p)| \leq n^3$ by Lemma 2.2.8 so the values $f(k,p)$ can be computed in a dynamic programming fashion in $O(n^6)$ time. □

2.2.4 Left-Tight and Right-Tight Tasks

We describe the algorithm for left-tight tasks. The algorithm for right-tight tasks is essentially identical. We have the following lemmas whose proofs are readily adapted from the analogous results for tight tasks.

Lemma 2.2.11 *If $i \in D_k$ and $j \in D_l$ with $k+2 \leq l$, then $s_i \prec s_j$.*

Lemma 2.2.12 *Let B_k be any feasible subset of D_k. Then $|B_k| \leq 3$.*

Furthermore, we have the following observation. Recall that t is an edge that is shared by all tasks.

Lemma 2.2.13 *If S is any (not necessarily feasible) subset of T such that $|S \cap D_k| \leq 1$ for each k, then the total demand in S at any given edge $m \succ t$ does not exceed c_m.*

Proof. Let e be any edge to the right of t (*i.e.* $e \succ t$). Since all tasks are left-tight, then for any task i with $e \in span(i)$ we have $d_i \leq c_e/2$. Order the tasks $i \in S$ having $e \in span(i)$ in order of demand $d_{i_1} \leq d_{i_2} \leq \ldots \leq d_{i_b}$. Note that $d_{i_j} < d_{i_{j+1}}/2$ because we assume that only even k have $D_k \neq \emptyset$ or only odd k have $D_k \neq \emptyset$ and we also have $|S \cap D_k| \leq 1$. Inductively, we have $d_{i_j} \leq d_{i_b} 2^{j-b}$ so the total demand of tasks in S across edge e is bounded by:

$$\sum_{j=1}^{b} d_{i_j} \leq d_{i_b} \sum_{j=1}^{b} 2^{j-b} \leq 2d_{i_b} \leq c_e$$

□

The preceding lemmas indicate that we can use a dynamic programming algorithm similar to the one for tight tasks. The main difference is that we only need to be concerned with the edges $e \preceq t$ if we ensure we only take subsets of D_k of size at most 1. The structure exploited by the following dynamic programming algorithm is illustrated in Figure 2.5.

Since the optimum solution chooses at most 3 tasks from each D_k then the resulting solution found is within a factor 3 of the optimum. Let $f(k,p)$ denote the minimum total demand of a feasible

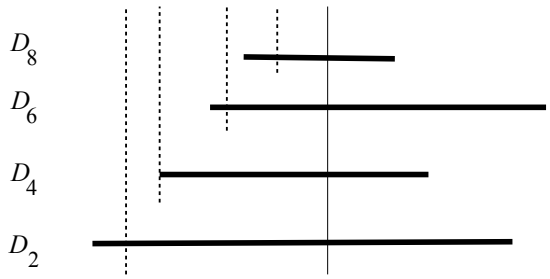

Figure 2.5: The tasks are drawn as thick lines. The common point in this intersecting case is indicated by the thin line. The dotted lines are the latest start times over all tasks in the respective demand classes. Demands in higher classes must start later than these lines. Finally, while the intervals do not look "nested" to the right of the common point, if we choose one task from each demand class then since $d_i \leq c_e/2$ for each edge e right of the common point that is spanned by a left-tight task i and since $d_{i'} < d_i/2$ if i' is in a lower demand class than i, then by summing a geometric series we see that we do not violate the capacity of any edge to the right of the common point.

collection of tasks from groups $D_k, D_{k-2}, D_{k-4}, \ldots$ that has total profit exactly p and contains at most one task from each demand class $D_l, l \leq k$. The recurrence looks similar to the one for tight tasks except the set $A(k,p)$ is restricted to singleton subsets of D_k and the only edges we need to check for feasibility those edges $e \preceq t$ in the common intersection of all tasks in $D_{k-2} \cup D_{k-4} \cup \ldots$.

The proof of correctness is similar to that of the recurrence for tasks with both endpoints tight. The main difference is that Lemma 2.2.13 assures us that no edge to the right of the common edge t is violated by any subset S that represents any finite $f(k,p)$ entry. Furthermore, the recurrence can be computed in polynomial since there are at most n distinct values for k, the maximum total profit p to be considered is $O(n^2)$, and the number of items of each D_k that need to be iterated over is at most n so the total running time is $O(n^4)$.

Lemma 2.2.14 *The dynamic programming algorithm finds a subset whose total profit is at least 1/3 of the optimum total profit for these left-tight tasks.*

Proof. Let T^* be an optimum collection of left-tight tasks. For each D_k, discard all but the most profitable task in $T^* \cap D_k$ from T^*; call this new set T'. Since $|T^* \cap D_k| \leq 3$ then one third of the total profit of T^* remains in T'.

The dynamic programming routine finds the optimum feasible subset of tasks S having $|S \cap D_k| \leq 1$ for each integer k, so surely $p(S) \geq p(T') \geq p(T^*)/3$. □

Theorem 2.2.15 *There is a polynomial-time 3-approximation for intersecting instances of UFP that are either left-tight or right-tight.*

We summarize what we have done to prove Theorem 2.2.2. We lost a constant factor in the approximation ratio in the reduction to simplified instances as in Definition 2.1.3. We further reduced the problem to intersecting cases in Section 2.2.1 and lost an $O(\log n)$ factor. Next, we partitioned

the tasks in the intersecting instances into three groups both-tight, left-tight, and right-tight (we can also remove the condition that the tasks are tight from our definition of simplified and deal with the slack tasks in Section 2.2.2). Finally, in Sections 2.2.3 and 2.2.4 we demonstrated how to either solve or approximate the solutions to each of these groups within a constant factor. The composition of these steps results in an $O(\log n)$-approximation for general instances of UFP, thereby proving Theorem 2.2.2.

2.2.5 An Extension to Cycles

The unsplittable flow problem on cycles can be solved approximately using the algorithm for paths. The following approach was observed in [23]. Consider an edge e in the cycle with the smallest capacity c_e and partition the tasks used in an optimum solution, say T^*, into two groups. Group 1 is the collection of tasks that are routed along edge e and group 2 is the collection of tasks that are not routed along edge e. Say the total profit of these groups is, respectively, OPT_1 and OPT_2.

We can find a subset of tasks with profit at least $OPT_1/(1-\epsilon)$ for any constant $\epsilon > 0$ by using the known FPTAS for Knapsack [55]. For each task i with demand d_i and profit p_i, we create an item for the knapsack with size d_i and value p_i. The overall capacity of the knapsack is c_e. Any feasible packing to the Knapsack instance maps directly to a feasible solution for UFP on the cycle by simply routing all tasks whose corresponding Knapsack item is packed. These tasks are routed along the route using edge e. Since all tasks in this solution use edge e and e has the minimum capacity over all edges, then surely no other edge cannot have its capacity constraint violated.

Notice that the tasks in group 2 (which are not routed across e) correspond to a feasible solution to the UFP problem on the line obtained by deleting edge e and all tasks in T^* whose paths use edge e. Using the UFP approximation algorithm described in this paper, we can find a feasible subset of tasks whose total profit is at least $\Omega\left(\frac{1}{\log n}\right) OPT_2$. Thus, we get an $O(\log n)$-approximation to UFP on cycles by taking the best of our two approximations to OPT_1 and OPT_2.

2.3 An $O(\log_d n)$-Approximation in Time $n^{O(d)}$

The main result of this section is that for any integer $d \geq 2$ (perhaps a function of n), there is an $O(\log_d n)$-approximation for UFP with running time $n^{O(d)}$. This is interesting for a few reasons:

- For any constant $c > 0$, this yields a polynomial-time $c \log_2 n$-approximation for UFP. Informally, we can select the constant suppressed by the $O(\cdot)$ notation in the $O(\log n)$ approximation to be arbitrarily small. Say the approximation ratio of the following algorithm is actually bounded by $c' \log_d n$. Then to get a $c \log_2 n$-approximation for some constant $c > 0$, we simply choose any integer d greater than $2^{c'/c}$.
- There is a quasi-polynomial time $O\left(\frac{\log n}{\log \log n}\right)$-approximation by choosing $d = \Theta(\log n)$.

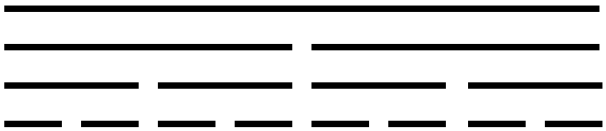

Figure 2.6: An sketch of an instance that requires logarithmically many groups of "disjoint intersecting instances".

- There is a constant-factor approximation running in sub-exponential (*i.e.* $2^{o(n)}$) time. For example, choosing $d = \Theta(\sqrt{n})$ results in an $O(1)$-approximation running in time $n^{O(\sqrt{n})} = 2^{O(\sqrt{n}\log n)}$. More generally, choosing $d = \Theta(n^{\epsilon})$ for small constant values of $\epsilon > 0$ results in an $O\left(\frac{1}{\epsilon}\right)$-approximation running in time $2^{O(n^{\epsilon}\log n)}$.

First, we want to emphasize that it is not possible to improve on the logarithmic approximation from Section 2.2 through a more clever combination of intersecting cases. In Section 2.2, one of the basic ideas was to partition the input into $O(\log n)$ groups. In each group, the tasks were partitioned into intersecting instances where no two tasks from different instances in this group shared a common edge. One might wonder if it is possible partition an input instance differently into even fewer groups of "disjoint" intersecting instances. The example sketched in Figure 2.6 shows this is not possible. A more formal construction of this instance is the following. Let I_1 denote the instance that includes a single task that spans at least one edge. For $k > 1$, let I_k be the instance that is formed as the disjoint union of two copies of I_{k-1} one where one instance of I_{k-1} has all tasks starting strictly later than the end nodes of all tasks in I_{k-1}. Finally, add one more task to I_k that spans all tasks in both instances of I_{k-1}.

The total number of tasks in I_k is $n_k = 2^k - 1$. We argue, by induction, that at least k groups are required to partition the tasks of I_k so that each partition can be expressed as the union of intersecting instances where no two tasks from different instances share a common edge. For $k = 1$, it is trivial. Now, for $k > 1$, consider such a partition of the tasks in I_k. If the long task spanning both instances of I_{k-1} is in a partition by itself, then at least $k - 1$ groups are required to partition the remaining nodes by induction (since the two instances of I_{k-1} are disjoint). Finally, if one of the instances of I_{k-1} has a task in the same group as the long task in I_k, then the other instance of I_{k-1} cannot have any tasks in the same group. By induction, we require at least $k - 1$ more groups for this instance of I_{k-1}. Since $k = \log_2 n_k - o(1)$, then we see that we require at least a logarithmic number of partitions.

To deal with this difficulty we generalize the notion of an instance being intersecting. The basic ideas of the algorithm in this section are similar to the ideas in the $O(\log n)$-approximation in Section 2.2. We lose an $O(\log_d n)$-factor by reducing to instances where there is a collection of $d - 1$ edges E' (that we call *partition edges*) such that every task spans at least one edge in E' (setting $d = 2$ produces intersecting instances as considered in Section 2.2). We call such instance d-intersecting

instances.

This reduction was inspired by an analogous reduction by Gamzu and Segev for the Highway Problem [44]. The Highway Problem is similar in spirit to UFP in that we have a collection of tasks over an underlying path. However, the tasks come with a budget b_i rather than a demand and a profit and there are no capacities on the edges. Our job is to assign prices to each edge of the path to maximize our profit, which is calculated as follows. For each task i, if the sum of the prices of all edges spanned by i does not exceed b_i, then that task pays this total price. Otherwise, task i pays nothing. In [44], they also reduce to cases where each task spans one of $d-1$ edges and lose an $O(\log_d n)$ factor. They then describe a constant-factor approximation for such instances whose running time is a polynomial factor larger than $(O(\log nm))^{O(d)}$. By choosing $d = \Theta(\sqrt{\log n})$ (or, more generally, $d = \Theta(\log^\delta n)$ for any constant $0 < \delta < 1$), this is a polynomial time $O(\log n / \log\log n)$-approximation. Actually, their algorithm works for the more general case where the underlying graph is a tree, but when their ideas are restricted to a path then our reduction to $d-1$ intersecting cases is similar to their reduction.

To approximate instances where each task spans one of $d-1$ edges, we extend the ideas of the $O(\log n)$-approximation. There is no good notion of "left-tight" or "right-tight" in such instances since tasks may have bottlenecks between edges of E', but we demonstrate that, after guessing a certain subset of up to $d-1$ tasks, the remaining tasks may be classified as "left-tight" or "right-tight" (or both) in some sense. Then, the dynamic programming algorithms from Section 2.2 are generalized to d-intersecting instances.

2.3.1 An Alternative Goal

To further simplify our search in some cases, we use the following definitions.

Definition 2.3.1 *A (not necessarily feasible) subset of tasks $T' \subseteq T$ is* conflict-free *if for any two tasks $i, i' \in T'$ the subset $\{i, i'\}$ is feasible.*

Definition 2.3.2 *A (not necessarily feasible) subset $T' \subset T$ is* demand class independent *if, for each demand class D_k, no two tasks in $T' \cap D_k$ share a common node in the underlying path.*

The following lemma demonstrates the usefulness of these properties.

Lemma 2.3.3 *Let $T' \subseteq T$ be a conflict-free subset that is also demand class independent. Then T' may be partitioned into four feasible subsets of T in polynomial time.*

Proof. We begin by forming a planar graph H. For each task $i \in T'$, add two points in the Euclidean plane at (s_i, d_i) and (t_i, d_i) and connect these by a straight line segment. Notice that since T' is demand class independent then no two line segments share a common point in this drawing.

For each edge $e = (j, j+1)$ on the underlying path, if e is spanned by at least two tasks in T', then let i, i' be the two such tasks with greatest demand values $d_i, d_{i'}$. Add a point to the paths for

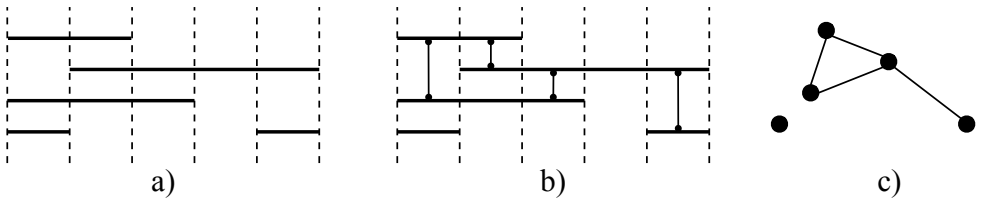

Figure 2.7: Tasks with larger demands are drawn higher in the figure. Figure a) shows an example of tasks with the vertical dashed lines corresponding to nodes in the underlying path. Figure b) illustrates the planar graph H drawn from the given tasks. Figure c) is the planar graph H' obtained by contracting each path P_i into a single node.

i and i' at location $(j + 1/2, d_i)$ and $(j + 1/2, d_{i'})$, respectively, and connect these points with a straight line segment. It is easy to see that no line segment drawn in this manner crosses any other line segments or touches any other node other than its endpoints. That is, we have a planar graph if we view the endpoints of all line segments as vertices and the segments between points as edges. Let P_i be the nodes on the path drawn horizontally between the points corresponding to s_i and t_i. Before adding the vertical edges, P_i was simply a line segment but it may have been subdivided into a path with internal nodes when the vertical edges were added. Contract each path P_i to a single point, say p_i, and call this new graph H'. This construction is illustrated in Figure 2.7. Since the property of a graph being planar is preserved under contractions, then H' is also planar. Also, each node in H', being the contraction of some path P_i, corresponds naturally to a task in T. We can colour the nodes of H' using 4 colours in polynomial time [81].

The claim is that the nodes in any one of these colour classes corresponds to a feasible subset of tasks. To see this, let us first consider the total demand across any edge e in the original set of tasks T'. If there is only one task that spans e, then surely no colour class of H has the total demand across e exceeding c_e. Otherwise, let i, i' be the two tasks with largest demands $d_i > d_{i'}$ that span e. Since T' is conflict-free, then we have $d_i + d_{i'} \leq c_e$.

Consider any two tasks i_1, i_2 in T' that span e and say that $d_{i_1} \leq d_{i_2}$. Since T' is demand class independent and since $D_k = \emptyset$ for all even k or $D_k = \emptyset$ for all odd k, then it must be that $d_{i_1} \leq d_{i_2}/2$. So, the sum of the demands of tasks in T' that span e is at most $d_i + d_{i'} + d_{i'}/2 + d_{i'}/4 + d_{i'}/8 + \ldots \leq d_i + 2d_{i'}$. Now, we have an edge between points p_i and $p_{i'}$ in H' since they are the largest two demands that span e so the total demand across e in any particular colour class is at most $d_i + d_{i'}/2 + d_{i'}/4 + d_{i'}/8 + \ldots \leq d_i + d_{i'} \leq c_e$. That is, the total demand in any colour class across edge e does not exceed c_e. Since this holds for any edge e, then every colour class of our 4 colouring of H' corresponds to a feasible subset of T'. □

Also, in some cases when we are searching for OPT, the following lemma allows us to restrict our search to certain solutions while only losing a constant factor in the approximation ratio. In particular, it says that there is a feasible solution that is also demand class independent whose total

profit is at least $OPT/4$.

Lemma 2.3.4 *For any feasible subset $T' \subseteq T$, we can partition T' into four groups in polynomial time such that each of these groups is conflict-free and demand class independent.*

Proof. Focus on a particular demand class $T' \cap D_k$. The claim is that no node v is spanned by more than four tasks in $T' \cap D_k$. Suppose some node v was spanned by five tasks in $T' \cap D_k$. Then either three of these tasks have some bottleneck edge before v or three of these tasks have some bottleneck edge after v. Suppose it is the former case (the latter is similar) and that these tasks are i_1, i_2, i_3. Also, suppose that i_1 has the right-most bottleneck $e \preceq v$ meaning i_2 and i_3 also span edge e. Then the total demand across edge e is at least $d_{i_1} + d_{i_2} + d_{i_3} \geq d_{i_1} + d_{i_1}/2 + d_{i_1}/2 = 2d_{i_1}$. Since e is a bottleneck for task i_1 then $d_{i_1} > c_e/2$ meaning the total demand across e exceeds c_e contradicting feasibility of T'.

Now, form the graph G_k whose nodes are tasks in $T' \cap D_k$ with two such nodes being connected by an edge (in G_k) if and only if they share a common node in the underlying path. Then G_k is an interval graph and we just argued that all cliques in G_k have size at most four. Interval graphs are perfect graphs so we may efficiently colour the nodes in G_k with four colours so that no two adjacent nodes receive the same colour [48]. Do this for each demand class $T' \cap D_k$ (using the same four colours) and let the four colour classes be the four partitions of T' with the desired property. Finally, since T' is feasible then it is conflict-free. Any subset of T' (in particular, each of the partitions we just found) must also be conflict-free. □

These two lemmas say that the optimum profit of a feasible subset of tasks and the optimum profit of a conflict-free and demand class independent subset of tasks are within a constant factor of each other (if the tasks are tight). We note that this does not hold if we only consider conflict-free subsets or only demand class independent subsets. For example, any feasible subset of tasks in the bad gap example from the proof of Lemma 2.0.3 has profit at most 1 whereas no two tasks in this example are in the same demand class so the optimum profit of a demand class independent subset is n. On the other hand, no two tasks in the following instance with unit profits conflict whereas the optimum solution is only 3. The example is similar to the example from the proof of Lemma 2.0.3 except the demand d_i of task i is $1 + 2^{-i}$ and the capacity c_i of the i'th edge from the left is $2 + 2^{-i} + 2^{-i-1}$. The tasks are tight since task i spans the i'th edge and

$$d_i = 1 + 2^{-i} > \frac{2 + 2^{-i} + 2^{-i-1}}{2} = \frac{c_i}{2}$$

It is also easy to check that no two tasks conflict. However, no three tasks can be chosen in any feasible solution since any three tasks have total demand exceeding 3 across the first edge, which has capacity $2\frac{3}{4}$. So, there may be an $\Omega(n)$ gap between the optimum profit of a feasible subset of tasks and the optimum profit of either a conflict-free subset of tasks or a demand class independent subset of tasks. This is why we simultaneously consider both the property of being conflict free and the property of being demand class independent.

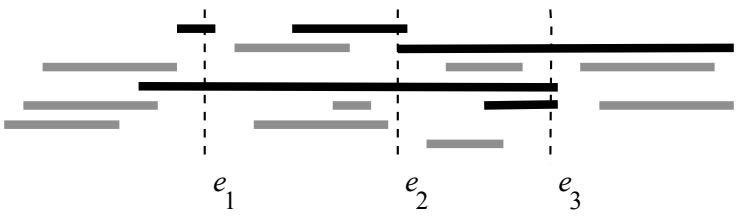

Figure 2.8: An example with $n = 15$ and $d = 4$. The black tasks are those in T' and the remaining tasks in some T_i are displayed in grey.

The following two lemmas are essentially restatements of some results in the previous section, but they are useful in this form in what follows.

Lemma 2.3.5 *For any two tasks i, i' with $d_i \leq d_{i'}$, either $d_{i'} < 2d_i$ or task i' does not span any edge that is a bottleneck for task i.*

Proof. Say $i \in D_k$ and $i' \in D_{k'}$ and suppose $2d_i \leq d_{i'}$. Let e be any bottleneck edge of task i. Then $c_e/2 < d_i \leq d_{i'}/2$ so $c_e < d_{i'}$. Since every task is feasible on its own, then e is not spanned by i'. □

Corollary 2.3.6 *For any two tasks $i \in D_k$, $i' \in D_{k'}$ with $k \leq k'$. Either $k = k'$ or no bottleneck of i is spanned by i'.*

Proof. If $k \neq k'$ then $k + 2 \leq k'$ since the instance is simplified (*cf.* Definition 2.1.3). Then $2d_i < 2^{k+2} \leq 2^{k'} \leq d_{i'}$ and Lemma 2.3.5 says that no bottleneck of i is spanned by i'. □

2.3.2 A Reduction to d-Intersecting Instances

Recall $d \geq 2$ is an integer that we may consider as a parameter to the algorithm. Let $e_1 \prec \ldots \prec e_{d-1}$ be any collection of edges whose deletion breaks the underlying path into paths containing at most m/d vertices (*e.g.* the e_i are spaced as equally as possible). Let T' be the tasks that span some $e_1, \ldots, e_{d-1}$ and partition the remaining tasks $T_1, \ldots, T_d$ based on the path in $P - \{e_1, \ldots, e_{d-1}\}$ they are contained in. More specifically, let T_1 be the collection of tasks that end before e_1, T_d be the collection of tasks that start after e_{d-1}, and for every $1 < i < d$, let T_i be the collection of tasks that start after e_{i-1} and end before e_i. This partition of the tasks is illustrated in Figure 2.8.

Later, we develop a constant-factor approximation for finding the maximum total profit of a feasible subset of T' that runs in time $n^{O(d)}$. Suppose this constant factor is $c \geq 1$. We get a $c \log_d n$ approximation for the general problem by the following routine. Since the edges spanned by any two i, i' in different T_k are disjoint (as i and i' are separated by some e_j), then we may naively take the union of feasible solutions to each T_k as a feasible solution for R. Let S' be the c-approximate solution for the instance with tasks from T' and let $S_1, \ldots, S_d$ be the approximate solutions for

the instances with tasks from $T_1, \ldots, T_d$, respectively, by recursively calling this algorithm for each T_k in turn and using the union of these solutions. Return the more profitable of the two sets S' or $S_1 \cup \ldots \cup S_d$.

Consider an optimum subset of tasks $S^* \subseteq T$ of profit OPT. Either the total profit in $S^* \cap T'$ is at least $\frac{OPT}{\log_d m}$ or else the total profit in $S^* \cap (T_1 \cup \ldots \cup T_d)$ is at least $\left(1 - \frac{1}{\log_d m}\right) OPT$. In the former case, our c-approximation finds a solution of cost at least $\frac{OPT}{c \log_d m}$.

In the latter case, let OPT_k denote the optimum profit of a feasible subset of T_k and notice that $\sum_{k=1}^{d} OPT_k \geq \left(1 - \frac{1}{\log_d m}\right) OPT$. Inductively (the base cases with $m \leq d$ are trivial), we have that the profit of S_k is at least $\frac{OPT_k}{c \log_d m/d}$ for all $1 \leq k \leq d$. Thus, the profit of the solution $S_1 \cup \ldots \cup S_d$ is at least

$$\begin{aligned}
\sum_{k=1}^{d} \frac{OPT_k}{c \log_d m/d} &\geq \frac{1}{c \log_d m/d} \cdot \left(1 - \frac{1}{\log_d m}\right) OPT \\
&= \frac{1}{c \log_d m/d} \cdot \left(\frac{\log_d m - 1}{\log_d m}\right) OPT \\
&= \frac{1}{c \log_d m/d} \cdot \left(\frac{\log_d m/d}{\log_d m}\right) OPT \\
&= \frac{OPT}{c \log_d m}.
\end{aligned}$$

Notice that there are at most $m/(d-1)$ calls to the constant factor approximation since each edge of the original path is used as one of the $d-1$ partition edges at most once. Summarizing:

Theorem 2.3.7 *Let $d \geq 1$ be an integer. If we have a constant-factor approximation for instances of UFP where there are $d-1$ edges $e_1, \ldots, e_{d-1}$ such that every task spans at least one of the e_k, then we have an $O(\log_d m)$-approximation for general instances of UFP. Recalling that $m \leq 2n$, this is also an $O(\log_d n)$-approximation. Furthermore, the running time of the general approximation algorithms is only a polynomial factor larger than the running time of the constant-factor approximation.*

2.3.3 Simplifying the Instances

Our goal in this section is to restrict our attention to d-intersecting instances with even more structure while losing only a constant factor in the approximation ratio. There are two main steps in this process. The first step ensures no "interval" between edges $e_1 \prec \ldots \prec e_{d-1}$ has tasks both starting and ending in that interval. The second step guesses the task with the largest demand (if any) across each of the $d-1$ edges. Our goal is then to approximate the optimum feasible subset of the remaining tasks. Once the largest demand tasks are known across each of the $d-1$ edges, we are able to impose a lot of structure on the remaining tasks which is exploited in the dynamic programming phases in subsequent sections.

For the first step, observe that the nodes in the underlying path are naturally partitioned into d intervals $I_1, \ldots, I_d$ by the edges $e_1, \ldots, e_{d-1}$ in the following way. Consider a node v on the underlying path. If $v \prec e_1$ then we say $v \in I_1$. Similarly, if $v \succ e_{d-1}$ then we say $v \in I_d$. Finally, if $e_{k-1} \prec v \prec e_k$ then we say $v \in I_k$. For each task i, let $l(i)$ be the interval with $s_i \in l(i)$ and let $r(i)$ be the interval with $t_i \in r(i)$. Consider a function $X : \{I_1, \ldots, I_d\} \to \{S, E\}$ that assigns a label $X(I_k)$ to each interval. We use the labels S and E to denote that the interval will be a *start* or an *end* interval. Given such a labeling X, let $T_X = \{i \in T : X(l(i)) = S \text{ and } X(r(i)) = E\}$. That is, each task $i \in T_X$ starts in an interval labelled S and ends in an interval labelled E. We want to find a labelling X so that the optimum solution does not decrease very much after we remove some tasks from T to obtain T_X. Suppose T^* is an optimum subset of tasks in T with profit $OPT = p(T^*)$.

Lemma 2.3.8 *There is a label function X such that the maximum profit of a feasible subset of T_X is at least $OPT/4$. Furthermore, we can, in polynomial time, find a collection of $O(d)$ label functions $\{X_\alpha\}$ such that the optimum of some T_{X_α} is at least $OPT/4$.*

Proof. Consider a random label function. Let $X(I_k)$ be randomly and independently chosen to be S or E for every $1 \leq k \leq d$. The probability any $i \in T$ is in T_X is at exactly 1/4 since its two endpoints s_i and t_i lie in separate intervals and these intervals are labeled independently. Then the expected total profit of items in $T_X \cap T^*$ is exactly $OPT/4$.

Notice that the probability any given task i is in T_X depends only on the labels of the two intervals containing its endpoints. We may find, in polynomial time, a collection of $O(d)$ label functions $\{X_\alpha\}$ through a textbook application of using a family of pairwise-independent random values (*e.g.* [68]) such that the profit of $T^* \cap T_{X_\alpha}$ is at least $OPT/4$ for some label function X_α in the collection. □

Suppose, now, that X is a label function that induces a subset of tasks T_X whose cost is at least $OPT/4$. By Lemma 2.3.8, we only have $O(d)$ different label functions to try and we proceed to run the rest of the algorithm on each of these label functions and keep the most profitable answer returned.

From now on, we suppose that we have a label function $X\{I_1, \ldots, I_d\} \to \{S, E\}$ such that $T = T_X$ and that the optimum value OPT we are concerned with is for a subset T^* of T_X. For the next step in this section, consider any feasible subset T' of T. Say a subset C of T' is a *canopy* of T' if the following two criteria hold:

- If $i \in C$, then there is some partition edge $e_k, 1 \leq k \leq d-1$ such that $e_k \in span(i)$ and any other $i' \in T'$ that also has $e_k \in span(i')$ satisfies $d_{i'} \leq d_i$.
- Every edge e_k that is spanned by some task in T' is also spanned by some task $i \in C$ such that $d_{i'} \leq d_i$ for any $i' \in T'$ that spans e_k.

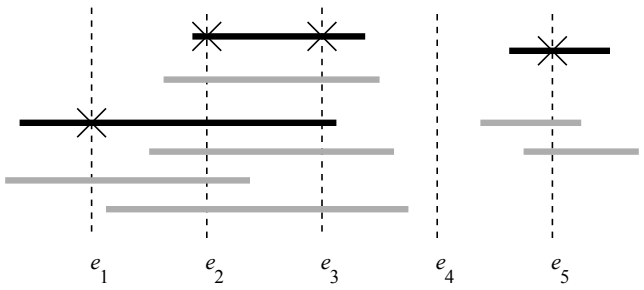

Figure 2.9: Tasks with larger demands are drawn higher in the figure. The dark tasks form a canopy for the given set of tasks. Each partition edge e_k that is spanned by some task has the largest demand task highlighted with $\times$ where the task crosses e_k.

Informally, a canopy of T' contains the largest demand task in T' across each of the $d-1$ partitions, if any. Ties for the largest demand across a specific e_k may be broken in any way. Figure 2.9 highlights a canopy for a feasible solution.

Notice that there are at most $(n+1)^{d-1} = n^{O(d)}$ possible canopies over all feasible subsets T' of T; for a feasible subset T' and for each edge e_k we have that either e_k is not spanned by any task in T' or we have one of the largest tasks in T' spanning e_k in the canopy. We may try all $n^{O(d)}$ guesses for a canopy and suppose we have properly guessed a canopy C of an optimum solution T^*. Form a new set of tasks $\hat{T}$ from T by discarding the following tasks:

1. all tasks in C
2. all tasks that span a partition edge e_k that is not covered by the canopy C
3. all tasks i that span a partition e_k where $d_i > d_{i'}$ for all $i' \in C$ spanning e_k
4. all tasks i such that $C \cup \{i\}$ is not feasible

The first type of task is removed because we already guessed it to be in T^*, the second type is removed because T^* does not have any task spanning the given edge, the third type is removed because it exceeds our guess for the maximum demand spanning e_k in T^*, and the fourth is removed because they clearly cannot be in an optimum solution when we guess the proper canopy. Notice that the optimum solution of $\hat{T}$ is precisely $p(T^* - C)$.

The benefits of guessing the canopy when given a labeling X are summarized in the following lemma. The lemma basically says that all bottleneck edges for any task i not in a canopy for a feasible solution must be in one of the two end intervals $l(i)$ or $r(i)$.

Lemma 2.3.9 *Suppose C is a canopy for some feasible subset T' and let $\hat{T}$ be the subset of T obtained by removing the above four types of tasks from T given canopy C. Then for any $i \in \hat{T}$ and for any bottleneck edge e for task i we have that either $e \in l(i)$ or $e \in r(i)$.*

Proof. Suppose that e is a bottleneck of $i \in \hat{T}$ and that $e \notin l(i) \cup r(i)$. It cannot be that e is one of the partition edges $e_j \in \{e_1, \ldots, e_{d-1}\}$ because we would otherwise have some task $i' \in C$ with larger demand than i spanning the same edge e_j. But then $d_{i'} \geq d_i > c_{e_j}/2$ since e_j is a bottleneck for i so i would have been discarded after the canopy C was guessed.

So, say that $e \in I_k$ with $1 < k < d$ and notice that i spans both e_{k-1} and e_k since the start node s_i of task i lies to the left of I_k and the end node t_i of task i lies to the right of I_k. Suppose that $X(I_k) = S$ (the other case is similar). Since $i \notin C$ there is some $i' \in C$ such that i' also spans e_{k-1}. Let i' be such a task in C of largest demand; in particular, $d_{i'} \geq d_i$ since i' is in the canopy and both span the common edge e_{k-1}. Now, since $X(I_k) = S$ it must be that i' also spans e_k. Since i' spans both e_{k-1} and e_k, then it must also span edge e (recall $e_{k-1} \prec e \prec e_k$). Then $d_i + d_{i'} \geq 2d_i > c_e$ since e is a bottleneck for i. But then $\{i, i'\}$ is not feasible meaning $C \cup \{i\}$ is not feasible. This contradicts $i \in \hat{T}$. □

Now, we partition $\hat{T}$ into three groups, one of which consists of tasks i with all bottlenecks in $l(i)$, another consists of tasks i with all bottlenecks in $r(i)$, and a final group consisting of tasks i with a bottleneck in both $l(i)$ and $r(i)$. We finally solve each of these three partitions either exactly or within a constant factor which, by taking the better of the three solutions, will lead to a constant factor approximation for the optimum solution in T.

2.3.4 Tasks With Both Endpoints Tight

In Section 2.2.3, an exact algorithm was presented for tight intersecting instances whose profits were integers in the range $[0, 2n]$. The main property that was exploited by the algorithm is that the instance looked somewhat like a nested sequence of intervals in that if two tasks were in different demand classes, then the span of the one with larger demand was contained in the span of the one with smaller demand. A nested sequence of intervals is a special case of the property of being *laminar*. Reall that a collection of subsets $\mathcal{S} = \{S \subseteq V\}$ of a set V is called laminar if for any $S, T \in \mathcal{S}$ either $S \cap T = \emptyset, S \subseteq T$, or $T \subseteq S$. Extend this definition to say that a collection of tasks T' is *laminar* if for any $i, j \in T'$ we have either $span(i) \cap span(j) = \emptyset, span(i) \subseteq span(j)$, or $span(j) \subseteq span(i)$.

As a first step, adjust the capacities in each interval to be monotone across that interval in a similar way to how the capacity profile was made unimodal in the $O(\log n)$ approximation. For each interval I_k with $X(I_k) = S$, we can assume that the capacities are non-decreasing in the interval T_k by the following reason. For $e, e' \in I_k$ with $e \prec e'$, the total demand spanning e in a feasible subset of $\hat{T}$ does not exceed the total demand spanning e' because no task $i \in \hat{T}$ ends before e'. So, if $c_e > c_{e'}$, then we may decrease c_e to be exactly $c_{e'}$. Similarly, if $X(I_k) = E$ then the capacities in the interval I_k can be modified to be non-increasing. The whole point is that we may now say that both the first and last edge of every task are now bottlenecks, rather than merely saying that both $l(i)$ and $r(i)$ contain a bottleneck edge for i. The property we exploit to solve instances of

tasks with both endpoints tight is that the sets of edges spanned by these tasks must be laminar, as we now show.

Lemma 2.3.10 *Any feasible subset T' of $\hat{T}$ is laminar.*

Proof. Let $i, i' \in \hat{T}$ be two tasks that cross (*i.e.* $s_i \prec s_{i'} \prec t_i \prec t_{i'}$). Suppose that $d_i \geq d_{i'}$ (the other case is similar). Since the first edge of task i' is also spanned by task i and since the first edge of task i' is a bottleneck edge for i', then $\{i, i'\}$ conflict across the first edge of i' because $d_i + d_{i'} \geq 2d_{i'} > c_{s_i}$. Therefore, both i and i' cannot be in any feasible solution. □

Modify $\hat{T}$ in the following way to ensure, for simplicity in the following algorithm, that no two tasks $i, i' \in \hat{T}$ have $span(i) = span(i')$. While there are two $i, i' \in \hat{T}$ with $span(i) = span(i')$, do the following. Say e, e' are the edges incident to s_i such that $e \prec s_i \prec e'$. Then "insert" a new edge e'' between e and e' by creating a copy s_i' of s_i and arranging the edges so $e \prec s_i' \prec e'' \prec s_i \prec e'$. Set the capacity of e'' to $c_{e'}$ and adjust the endpoints of the tasks as follows. For tasks $i'' \neq i$ (including i') whose start node was s_i, keep the start node of i'' as s_i. Finally, set the start node of task i to be s_i'. If it so happened that s_i was the first node on the path (so edge e was not present) then the new graph simply looks like $s_i' \prec e'' \prec s_i \prec e'$. It is easy to verify that a subset of nodes is feasible before this update if and only if it is feasible after this update, that e'' is a bottleneck for i, and that no other edge becomes a bottleneck for any task for which it was already a bottleneck. Though we have increased the number of edges in the path from m to at most $2n$, the algorithm that follows is no worse than constant-factor approximations so we may still claim that the final approximation ratio is $O(\log_d m)$. Now we may assume $span(i) \neq span(i')$ for every distinct $i, i' \in \hat{T}$.

The basic idea of the dynamic programming algorithm is that any feasible solution T' can be decomposed in the following way. Consider the set of edges spanned by some task in T'. Since T' is laminar, then either there is a task in T' that spans all of these edges or there is a node v such that all tasks in T' either end no later than v or start no earlier than v (the node v divides the set into two parts). These ideas are illustrated in Figure 2.10. The dynamic programming solves the problem of determining if it is possible to obtain profit exactly p using tasks that are entirely contained between two given points. Notice that after dividing a feasible solution into two parts by a "dividing node" v, we can do no worse by assuming that the parts on both sides of v leave the largest possible residual capacity over all edges spanned by that part.

The dynamic programming table we are interested is the following.

Definition 2.3.11 *For nodes $v \prec v'$ and a value p, let $R(v, v', p)$ denote the collection of feasible subsets of $\hat{T}$ of total profit exactly p where each subset consists only of tasks i with $v \preceq s_i$ and $t_i \preceq v'$. For such a subset $S \in R(v, v', p)$, let $m(S)$ denote the minimum remaining capacity of all edges in $[v, v']$ (the collection of edges that appear between v and v') after routing all tasks in S.*

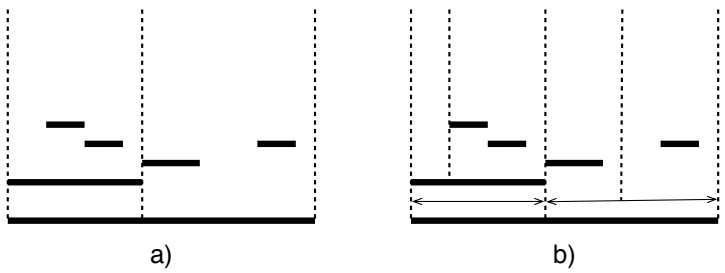

Figure 2.10: Decomposing a feasible solution. a) The first dashed line is the start of e and the last dashed line is the end of e'. After choosing the task that spans the entire interval, we may break the remaining solution into two halves by the middle dashed line. b) Recursively decomposing these subproblems further (the thin, double-arrowed line highlights the two subproblems).

Finally, let $a(v, v', p)$ denote the maximum of $m(S)$ over all $S \subseteq R(v, v', p)$. If $R(v, v', p) = \emptyset$ then let $a(v, v', p) = -\infty$.

Basically, $a(v, v', p)$ is the most capacity that can possibly be left across edges in $[v, v']$ over all feasible subsets of tasks contained in $[v, v']$ of total profit exactly p. The $a(v, v', p)$ values are related through the following recurrence. When $[v, v']$ consists of a single edge e, we simply have

$$a(v, v', p) = \begin{cases} c_e & \text{if } p = 0 \\ c_e - d_i & \text{if } \exists i \text{ with } s_i = v, t_i = v', p_i = p \\ -\infty & \text{otherwise} \end{cases}$$

The following lemma is the structure we exploit in the recurrence to calculate $a(v, v', p)$ values. It has probably been proven before for a laminar set of intervals, but we include a proof because it helps illustrate how the recurrence works.

Lemma 2.3.12 *If $v \prec v'$ are nodes and S is a laminar subset of tasks whose start and end nodes are all contained between v and v' then either a) there is some $i \in S$ with $span(i') \subseteq span(i)$ for all $i' \in S$ or b) there is some node v'' with $v \prec v'' \prec v'$ such that $t_i \preceq v''$ or $v'' \preceq s_i$ for all $i \in S$.*

Proof. Suppose there is no task i with $span(i') \subseteq span(i)$ for all $i' \in S$. Now, if there is no task in S with $s_i = v$ then we may chose the node immediately proceeding v in the underlying path as v'' and note that $v \prec v'' \prec v'$ and $v'' \preceq s_i$ for all $i \in S$. Otherwise, suppose task $i \in S$ has $s_i = v$ and that i spans the most edges among tasks in S starting at v. The claim is that we can chose $v'' = t_i$. Since we assumed case a) in the statement of the lemma is not satisfied, then $v \prec t_i = v'' \prec v'$. Suppose, for the sake of contradiction, that there was some $i' \in S$ with $s_{i'} \prec v'' \prec t_{i'}$. If $s_{i'} = v$, then i' is a task in S starting at v that spans more edges than i, contradicting our choice of i. Otherwise, if $v \prec s_{i'}$ then we have $v = s_i \prec s'_i \prec t_i \prec t_{i'}$ which contradicts S being laminar. Therefore, selecting $v'' = t$ demonstrates that S satisfies case b) in the statement of the lemma. □

For cases where $[v, v']$ contains multiple edges, it is useful to define the quantity $q(v, v', p)$. Intuitively, it is the maximum remaining capacity across the interval $[v, v']$ of a feasible subset of

the tasks contained in $[v, v']$ with profit exactly p *without* using the task i with $s_i = v, t_i = v'$ (if such a task exists). More precisely, $q(v, v', p)$ is the maximum of $m(S)$ over subset $S \in R(v, v', p)$ such that no $i \in S$ has $s_i = v$ and $t_i = v'$.

The $q(v, v', p)$ values can be computed from $a(v, v', p)$ values with smaller interval lengths $|[v, v']|$ by the following expression.

$$q(v, v', p) := \max_{v \prec v'' \prec v'} \left\{ \max_{0 \leq p' \leq p} \min\{a(v, v'', p'), a(v'', v', p - p')\} \right\}$$

The outer-most max in the expression tries all such nodes v'' and the inner-most max in the expression then "guesses" the profit of the two subsets of tasks on each side of this node v''.

We finally have that:

$$a(v, v', p) = \begin{cases} \max\{q(v, v', p), q(v, v', p - p_i) - d_i\} & \text{if } \exists i \text{ with } s_i = v, t_i = v', p_i \leq p, \\ & \text{and } d_i \leq q(v, v', p - p_i) \\ q(v, v', p) & \text{otherwise} \end{cases}$$

In the first case of the piecewise expression, the first argument in the $\max$ corresponds to task i not being used and the second corresponds to task i being used. The quantity $q(v, v', p - p_i)$ is used to determine the maximum possible remaining capacity across $[v, v']$ when total profit $p - p_i$ is routed from the tasks strictly contained in $[v, v']$. The second case is when there is no i with $p_i \leq p$ and $span(i) = [v, v']$ or when it is impossible to have at least d_i capacity remaining across all edges in $[v, v']$ by routing a total profit of $p - p_i$ of tasks strictly contained in $[v, v']$.

Lemma 2.3.13 *The recurrence correctly relates the $a(v, v', p)$ values.*

Proof. Correctness for the base of $[v, v']$ containing a single edge is immediate. For the remaining cases, we consider two options. If $a(v, v', p) = -\infty$ then surely $a(v, v'', p') = -\infty$ or $a(v'', v', p - p') = -\infty$ for all $v \prec v'' \prec v'$ and $0 \leq p' \leq p$ since we could otherwise take the union of two corresponding sets in $R(v, v'', p')$ and $R(v'', v', p - p')$ (respectively) as a set in $R(v, v'', p)$. If there is a task i with $s_i = v, t_i = v'$ and $p_i \leq p$ then it must be that $d_i > q(v, v', p - p_i)$ or else we could add i to the two sets corresponding to the argument of $q(v, v', p)$ to get a feasible set in $R(v, v', p)$. So, the recurrence is correct when $a(v, v', p) = -\infty$.

Now, suppose $a(v, v', p) \neq -\infty$ and that $S \in R(v, v', p)$ has $m(S) = a(v, v', p)$. If there is no task $i \in S$ with $s_i = v, t_i = v'$ and $p_i \leq p$ then Lemma 2.3.12 shows the set S can be split into two halves by a dividing node v''. For the proper guess of profit p' to the left of this dividing node, the corresponding term in $q(v, v', p)$ will be $a(v, v', p)$ so surely $q(v, v', p) \geq a(v, v', p)$. If there is such a task $i \in S$ then the set $S - \{i\} \in R(v, v', p - p_i)$ can be similarly divided (recall we are assuming no two tasks have the same start and end points) and we have $q(v, v', p - p_i) \geq a(v, v', p) + d_i$ In either case, the recurrence certainly determines a value that is at least $a(v, v', p)$. Using a similar argument, we see that all values that are not $-\infty$ that are determined by the recurrence correspond to feasible subsets in $R(v, v', p)$. Since $a(v, v', p)$ is the maximum possible residual capacity of sets in $R(v, v', p)$, then the recurrence actually determines the correct value of $a(v, v', p)$. $\square$

Say that v_s, v_e are equal to, respectively, the leftmost start node s_i and the rightmost end node t_j over all tasks in the instance. The profit of the optimum feasible subset of $\hat{T}$ is then the largest p such that $a(v_s, v_e, p) \neq -\infty$ and such a subset can be obtained through appropriate bookkeeping. Recall we assumed that all profits of tasks in $\hat{T}$ were integers in the range $[0, 2n]$. Since there are n tasks in total, then the only values of p with $a(v_s, v_e, p) \neq -\infty$ are integers in the range $[0, 2n^2]$. Also, since there are at most $2n$ points on the underlying path (even after we adjusted the endpoints so no two tasks span the same set of edges), then there are at most $2n$ choices for each v_s and v_e entry. Thus, the size of the table is $O(n^4)$. Now, each entry can be computed by making only $O(n^3)$ calls to other entries with strictly smaller intervals $[v, v']$ since there are $O(n)$ guesses for the dividing node v'' and $O(n^2)$ guesses for profit p' in the definition of $q(v, v', p)$. By using the recurrence, the $a(j, j', p)$ values can computed with dynamic programming in time $O(n^7)$. Note that this running time does not depend on d.

2.3.5 Tasks With One Tight Endpoint

As in Section 2.2.4, we only demonstrate the algorithm for left-tight tasks which we denote by $\hat{T}$. The algorithm for right-tight tasks is essentially identical. We exploit Lemmas 2.3.3 and 2.3.4 and restrict our search to only finding an optimum subset of conflict-free and demand class independent tasks (*cf.* Definitions 2.3.1 and 2.3.2) and lose only constant factor in the process. Note that the canopy C that we have guessed is not necessarily demand class independent, but Lemma 2.3.4 allows us to find a high-value subset of C that satisfies this property. Also, recall that all left-tight tasks i have a bottleneck in interval $l(i)$ and *all* bottlenecks of i lie in $l(i)$.

The following summarizes why we may make these simplifications.

Lemma 2.3.14 *If T' is an optimum conflict-free and demand class independent subset of $\hat{T}$, then we may find a feasible subset of $C \cup T'$ whose cost is within a constant factor of the cost of the optimum feasible subset F^* of $C \cup \hat{T}$ that includes all tasks in C.*

Proof. Say $p(F^*) = OPT$. Suppose $F^* \cap \hat{T} = T_0$ and note that $p(F^*) = p(C) + p(T_0)$ since F^* contains all of C. Then T_0 is conflict-free since it is feasible. By Lemma 2.3.4, there is a conflict-free and demand class independent subset $T_1 \subset T_0$ with $p(T_1) \geq p(T_0)/4$. Since T' is an optimum conflict-free and demand class indepenent subset of $\hat{T}$, then $p(T') \geq p(T_0)/4$ as well.

Now, consider the set $T' \cup C$. It is conflict-free since C is conflict-free, T' is conflict-free and no task in T' can conflict with a task in C because $T' \subseteq \hat{T}$ and we did not include tasks in $\hat{T}$ that conflicted with C. However, $T' \cup C$ is not necessarily demand class independent. The following argument is similar to the one made in Lemma 2.3.4. Consider any demand class D_k and any node v. Since C is feasible, then an argument similar to Lemma 2.2.8 says that at most 4 tasks in $C \cap D_k$ span v. Since T' is demand class independent then at most one task in T' spans v. So, at most five tasks in $C \cup T'$ span v.

Since the interval graph associated to tasks in $(C \cup T') \cap D_k$ has maximum clique size 5, then we can partition these tasks into 5 groups of which no two in the same group share a common point. Do this for all demand classes D_k and keep the most profitable of the 5 partitions in each class. This results in a conflict-free and demand class independent subset F of $C \cup T'$ with profit at least $p(C \cup T')/5$. Stacking the inequalities shows:

$$p(F) \geq p(C \cup T')/5 = p(C)/5 + p(T')/5 \geq p(C)/5 + p(T_0)/20 \geq p(C \cup T_0)/20 = p(F^*)/20$$

Finally, we use Lemma 2.3.3 to find a feasible subset of $p(F)$ with cost at least $p(F^*)/80$. □

As in other sections, we employ dynamic programming to find such an optimum conflict-free and demand class independent subset of $\hat{T}$. The dynamic programming in this section is a fair bit more complicated and technical, so we spend some time developing the intuitions and basic ideas behind it. As a pre-processing step, we can assume that the capacities of the edges in intervals labelled S are increasing so that the first edge of every task is a bottleneck but it is simpler to not modify the capacities of the edges in intervals labelled E. Since we do not alter the capacity of any edge in an interval labelled E and since no left-tight task has a bottleneck in an interval labelled E before this modification, then it is still true that all left-tight tasks we consider in this section still do not have a bottleneck edge in an interval labelled E. It is important to remember that this means no two left-tight tasks i, i' can conflict across an edge in an interval labelled E since, for any $e \in span(i) \cap span(i')$ in an interval labelled E, we have $d_i + d_{i'} \leq c_e/2 + c_e/2 = c_e$. We note that it may be that a left-tight task i now has a bottleneck across an interval labelled S apart from $l(i)$, but this will not be a problem in what follows.

Suppose the tasks are sorted so that $s_{i_1} \preceq s_{i_2} \preceq \ldots \preceq s_{i_n}$ (where $n = |\hat{T}|$ now). Let $T_j = \{i_j, i_{j+1}, \ldots, i_n\}$ and suppose that T' is a conflict-free and demand class independent subset of T_j for some $1 \leq j \leq n$. Suppose $i = i_{j'}$ where $j' < j$ and consider the set $T' \cup \{i\}$. We describe some conditions under which $T' \cup \{i\}$ cannot be conflict-free or demand class independent that help us build our dynamic programming routine. We begin by introducing some notation. First, for a task i' and an interval I_k, let $r(i', I_k)$ denote the least residual capacity left across interval I_k after choosing i'. That is, define the residual capacity $c_e^{i'}$ on edge e by

$$c_e^{i'} = \begin{cases} c_e - d_{i'} & \text{if } e \in span(i') \\ c_e & \text{otherwise} \end{cases}$$

Then $r(i', I_k) = \min_{e \in I_k} c_e^{i'}$. Given this notation, we define the following.

Definition 2.3.15 *Suppose T' is a conflict-free and demand class independent subset of left-tight tasks. For each interval I_k labelled S, let*

- *$F_{T'}(I_k)$ be the task $i \in T'$ that spans some node in I_k that minimizes the residual capacity $r(i, I_k)$ across interval I_k. If there are multiple such tasks, then any will do (*e.g. *the one with least index i_j).*

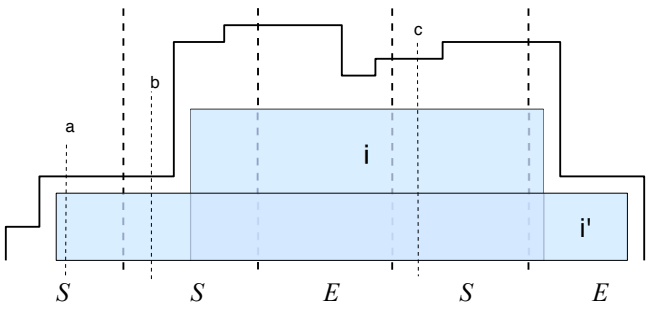

Figure 2.11: An illustration of why a conflict can be blamed on one of at most d tasks. The height of the task corresponds to their demand and the polyline surrounding the image is the capacity profile. The two rectangles shown are the tasks T'. We have $F_{T'}(I_1) = F_{T'}(I_2) = i'$ because of edges a and b and $F_{T'}(I_4) = i$ because of edge c. Note that the residual capacity left across I_2 by choosing only i' is strictly less than the residual capacity left across I_2 by choosing only i even though i has larger demand than i'. For each I_k labelled S, any other task i'' with $s_{i''} \preceq s_{i'}$ must span edges a and b because the first interval labelled E appears after these edges. Similarly, i must either end in I_3 or span c.

- *$G_{T'}(I_k)$ be the task $i \in T'$ that spans some node in I_k such that i has the smallest demand d_i among all such tasks. There cannot be multiple such tasks because T' is demand-class independent.*

If no task in T' spans some node in I_k, then simply say $F_{T'}(I_k) = G_{T'}(I_k) = nil$.

Now, we explore why $T' \cup \{i\}$ can fail to be either conflict-free or demand class independent.

Lemma 2.3.16 *If $T' \subseteq T_j$ and $i = i_{j'}$ for some $j' < j$ is such that $T' \cup \{i\}$ is not conflict-free, then i conflicts with $F_{T'}(I_k)$ for some interval I_k labelled S with $F_{T'}(I_k) \neq nil$.*

Proof. Suppose that edge e and task $i' \in T'$ are such that i and i' conflict across e. First, since neither i nor i' have a bottleneck edge in an interval labelled E, then e is in an interval labelled S and say this interval is I_k. If $s_i \notin I_k$ then since s_i appears before I_k and s_i spans some edge in I_k, then it must span all edges in I_k. Namely, it spans the edge $e' \in I_k$ that has the least remaining capacity $r(F_{T'}(I_k), I_K)$ across all edges in I_k in the singleton solution $\{F_{T'}(I_k)\}$. Now, since i and i' conflict across some edge in I_k and since the least residual capacity left by $F_{T'}(I_k)$ across I_k is no more than $c_e - d_{i'}$ (by definition of $F_{T'}(I_k)$), then surely i and $F_{T'}(I_k)$ also conflict.

On the other hand, if $s_i \in I_k$ (that is, $e \in l(i)$) then we still have $s_i \leq s_{F_{T'}(I_k)}$ by our ordering of tasks. Again, it must then be that i and $F_{T'}(I_k)$ conflict in this case because the least residual capacity across I_k is no more than $c_e - d_{i'}$. See Figure 2.11 for an illustration. □

Lemma 2.3.17 *If $T' \subseteq T_j$ and $i = i_{j'}$ for some $j' < j$ is such that $T' \cup \{i\}$ is not demand class independent, then $\{i, G_{T'}(I_k)\}$ is not demand class independent for some interval I_k labelled S with $G_{T'}(I_k) \neq nil$.*

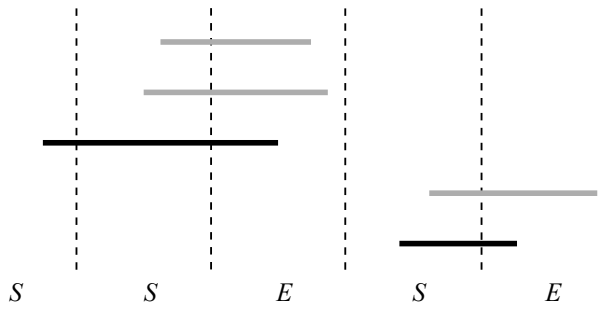

Figure 2.12: An illustration of why we only need to keep track of at most d tasks to detect violations to demand class independence. Tasks with larger demands are drawn higher and two tasks in the same demand class are drawn at the same level. The two dark tasks are in the smallest demand class for some interval labelled S. Notice that it is impossible for any task i with $s_i \preceq s_{i'}$ for each drawn task i' to both be in the same demand class as one of the grey intervals and to share a common point with that same grey interval. For example, if such a task was in the same demand class as the rightmost grey interval and shared a point with the grey interval, then it would have to span the first edge (which is a bottleneck) of the rightmost dark interval which is contradicts feasibility of each task by itself.

Proof. Suppose $i' \in T'$ is such that $\{i, i'\}$ is not demand class independent. That is, i and i' are in the same demand class and share a common node v. Since $s_i \preceq s_{i'}$, then we actually have that $s_i \preceq s_{i'} \preceq t_i$. Say $s_{i'}$ lies in interval I_k (which is an interval labelled S). The claim is that $G_{T'}(I_k) = i'$.

Otherwise, there would be a task i'' containing a node of I_k with $d_{i''} \leq d_{i'}$. Whether $s_{i'} \preceq s_{i''}$ or $s_{i''} \prec s_{i'}$, we have that both i' and i'' share a common node, namely the last node in I_k since I_k is labelled S. Now, since T' is demand class independent then it must be that i'' is in a lower demand class than i', so $d_{i''} \leq d_{i'}/2$. Also, since i is in the same demand class as i' then $d_{i''} \leq d_i/2$ as well. Since I_k is labelled S, both i and i'' span some node in I_k, and $s_i \preceq s_{i''}$, then i must also span the first edge spanned by i''. Because the tasks are left-tight, the first edge of i'' is a bottleneck for i''. But then we have $d_i/2 \geq d_{i''} > c_{s_{i''}}/2$ which contradicts the fact that task i is feasible on its own. This is illustrated in Figure 2.12. □

From these lemmas, we develop a dynamic programming approach that builds a solution in a "right-to-left" manner. To avoid conflicts or violations of demand class independence, it is sufficient to keep track of two tasks in each interval I_k with $X(I_k) = S$: the one that leaves the least residual capacity across that interval as well as the one in the least demand class that has a node in that interval (if there are any such tasks). This means a subproblem will be described by $O(d)$ integers and each such integer can take one of $n + 1$ values (to indicate one of the n tasks or to indicate no task at all), so the table has size $n^{O(d)}$. The time it takes to compute an entry given previous entries is also $n^{O(d)}$, so the total running time of the dynamic programming phase is $n^{O(d)}$.

Let X_S denote the set of intervals labelled S. Subproblems corresponding to subsets of T_j in

the dynamic programming phase are described by triples (F, G, p) where F and G are mappings $X_S \to \hat{T}$ and p is a target profit. For $1 \leq j \leq n$ and for mappings F, G from X_S to $T_j = \{i_j, \ldots, i_n\}$, let $A_j(F, G, p)$ be a boolean variable that is true if and only if there is some subset T' of T_j with profit $p(T') = p$ such that $F_{T'}(I_k) = F(I_k)$ and $G_{T'}(I_k) = G(I_k)$ for all $I_k \in X_S$. Then the $A_j(F, G, p)$ values are related through the following recurrence. Extend this notation to allow values of $A_j(F, G, P)$ with $j = n + 1$ where we say $T_{n+1} = \emptyset$. For the base case $j = n + 1$, we have:

$$A_{n+1}(F, G, p) = \begin{cases} \text{true} & \text{if } F(I_k) = G(I_k) = nil \ \ \forall I_k \in X_S \text{ and } p = 0 \\ \text{false} & \text{otherwise} \end{cases}$$

Before introducing the recurrence for $1 \leq j \leq n$, the following concepts will be useful. If we are given mappings F, G from X_S to $\hat{T}_{j+1}$ then we say that i_j is *compatible* with F, G if i_j does not conflict with any of the $F(I_k)$ values and if i_j is not in the same demand class as any $G(I_k)$ for which i_j has a point in $I_k \in X_S$. In other words, we can add i_j to any set T' with $F_{T'} = F$ and $G_{T'} = G$ without introducing a conflict or violating demand class independence. Then let F^{+i_j}, G^{+i_j} denote the mappings obtained from the set $T' \cup \{i_j\}$. That is, for each $I_k \in X_S$ for which i_j has a point in I_k, if either $F(I_k) = nil$ or $r(i_j, I_k) < r(F(I_K), I_k)$, then $F^{+i_j}(I_k) = i_j$, otherwise $F^{+i_j}(I_k) = F(I_k)$. Similarly, if i_j has a point in $I_k \in X_S$ then we set $G^{+i_j}(I_K) = i_j$ if either $G(I_k) = nil$ or $G(I_k)$ is in a higher demand class than i_j. Otherwise, we let $G^{+i_j}(I_k) = G(I_k)$.

Inductively, for $1 \leq j \leq n$ we have the following relation for computing $A_j(F, G, p)$.

1. If $A_{j+1}(F, G, p)$ is true, then $A_j(F, G, p)$ is true.

2. Otherwise, if $p_{i_j} \leq p$ and $A_{j+1}(F', G', p - p_i)$ is true for some pair (F', G') that is compatible with i_j such that with $F'^{+i_j} = F$ and $G'^{+i_j} = G$, then $A_p(F, G, p)$ is true.

3. Otherwise, $A_j(F, G, p)$ is false.

Lemma 2.3.18 *The recurrence correctly relates the values of* $A_j(F, G, p)$.

Proof. Surely it is true for $j = n + 1$ since the only conflict-free and demand class independent subset of $\emptyset$ has profit 0 and $F_\emptyset(I_k) = G_\emptyset(I_k) = nil$ for all $I_k \in X_S$.

For $1 \leq j \leq n$, first suppose that $A_p(F, G, p)$ is true and suppose that T' is a conflict-free and demand class independent subset of $\hat{T}_j$ with $F_{T'} = F$, $G_{T'} = G$, and $p(T') = p$. If $i_j \notin T'$, then $T' \subseteq \hat{T}_{j+1}$ as well so $A_{j+1}(F, G, p)$ is also true. Otherwise, if $i_j \in T'$ then $p_{i_j} \leq p$ and it must be that $A_{j+1}(F_{T'-\{i_j\}}, G_{T'-\{i_j\}}, p - p_i)$ is also true. This is found as (F', G') range over all pairs of mappings that are compatible with i_j in the second rule of the recurrence.

On the other hand, if $A_j(F, G, p)$ is false then surely $A_{j+1}(F, G, p)$ is also false since any subset of $\hat{T}_{j+1}$ of profit p associated to the mappings F, G would also be such a subset of $\hat{T}_j$. Finally, we argued above that if there is a conflict-free and demand class indepenendent subset T' of $\hat{T}_{j+1}$ with $p(T') = p - p_{i_j}$ and $F_{T'}^{+i_j} = F$ and $G_{T'}^{+i_j} = G$, then the subset $T' \cup \{i_j\}$ has profit p and is

also conflict-free and demand class independent. So, it must be that $A_{j+1}(F', G', p)$ is false for all possible pairs of mappings (F', G') that are compatible with i_j and have $F'^{+i_j} = F$ and $G'^{+i_j} = G$. □

The answer is then the largest p for which $A_1(F, G, p)$ is true for some F, G mappings. There are $n + 1$ choices for j and $n^{O(d)}$ choices for each of the functions F, G. Thus, the total size of the table is $n^{O(d)}$. Calculating a particular entry takes $n^{O(d)}$ time so the overall running time is $n^{O(d)}$. To construct a conflict-free and demand class independent subset with optimum profit, then we would simply maintain any particular set T' associated to each true $A_j(F, G, p)$ and construct these sets according to rules of the recurrence.

To summarize the algorithm in this section, recall that we assumed the instances were simplified according to Definition 2.1.3. Then, we reduced the problem to d-intersecting instances while losing another $O(\log_d m)$ factor. We lost a factor of 4 while pruning the instance according to some label function on the d intervals. Then, we tried each of the $O(n^d)$ guesses for the canopy. For each guess, we discarded the tasks that are not compatible with the canopy and classified the remaining tasks according to which end intervals have a bottleneck. We described a polynomial-time exact algorithm for the optimum profit of a feasible subset of tasks with both endpoints tight. For instances with only the left endpoints tight or only the right endpoints tight, we described an $n^{O(d)}$ exact algorithm for finding the maximum profit conflict-free and demand class independent subset of tasks. Lemma 2.3.14 shows how to prune such a subset to obtain a feasible subset whose profit is within a constant-factor of the optimum solution using these tasks. We take the better of the three solutions found for the tasks with left endpoints tight, right endpoints tight, or both endpoints tight and lost an additional factor of 3. Overall, this is an $O(\log_d n)$-approximation because we lost an $O(\log_d n)$ factor when reducing to d-intersecting instances and all other steps only lost a constant factor . The total running time is $n^{O(d)}$.

2.4 Approximating q-Conflicting Instances

There is a trivial q-approximation when each edge is spanned by at most q tasks based on the fact that interval graphs with maximum clique size q can be coloured with q colours [48]. Let's consider a more general setting. Say that a subset of tasks T' is *q-conflicting* if for any task i and any edge e spanned by i, the number of other tasks i' that also span e for which $d_i + d_{i'} > c_e$ is at most q. We exhibit an $O(q)$-approximation for q-conflicting instances. This definition admits the possibility of having much more than q tasks span each edge e in the input.

Similar instances were considered in [64] for the Maximum Independent Set of Rectangles (MISR) problem. In MISR, we are given a collection of axis-parallel rectangles in the plane and the goal is to find the largest subset of rectangles such that no two share a common point. They showed that if every point in the plane is touched by at most q rectangles, then there is an $O(q)$-

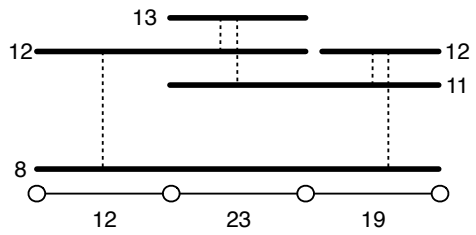

Figure 2.13: An instance with the "conflict-implies-contain" property that is not perfect. The numbers on the path are the edge capacities and the demands of the tasks are written next to the task. The endpoints of the dashed lines connect two tasks that conflict. The graph corresponding to conflicting pairs is then a cycle on 5 nodes so it is not perfect. It is easy to verify that all tasks are tight and that they are all in demand class D_3 so even simplified instances might not be perfect.

approximation algorithm that also bounds the integrality gap of a certain LP relaxation by the same ratio. Their algorithm proceeds in two phases. In the first phase, they solve an LP-relaxation that has constraints similar to the constraints in the LP 2.2 we consider soon. Say the optimum value of this LP is OPT_f. They round the solution to the LP to obtain $\Omega(OPT_f)$ (not necessarily independent) rectangles R with an additional property. No corner of any rectangle in R is touched by any other rectangle in R. That is, two rectangles in R conflict only by "overlapping" across their middles. In the second step, they exhibit that the intersection graph G_R whose nodes are R with edges between two conflicting rectangles is in fact a perfect graph (more specifically, a comparability graph). Since the size of the maximum clique is q, they can then efficiently find an independent set of size at least $|R|/q = \Omega(OPT_f/q)$ by q-colouring G_R.

Our algorithm also proceeds in two phases. First, we formulate and solve an LP relaxation that is stronger than the standard LP relaxation 2.1 that is reminiscent of the LP formed for UFP in [26]. We then use a rounding technique analogous to the technique that shares some similarities with the techniques [64] for the Maximum Independent Set of Rectangles problem. Specifically, we find a collection T' of demand class independent tasks (*cf.* Definition 2.3.2) whose total profit is within a constant factor of the LP optimum that also satisfies another strong property: if $i, i' \in T'$ conflict, then either $span(i) \subseteq span(i')$ or $span(i') \subseteq span(i)$ (analogous to the "overlapping rectangles" property considered in [64]). Simply put, if two tasks in T' conflict, then one is contained in the other: "conflict implies contain". Unfortunately, the graph obtained from conflicting pairs is not necessarily perfect meaning they may not be q-colourable (see, *e.g.*, Figure 2.13). However, there is still enough structure to the remaining tasks that allows us to use a relatively simple randomized algorithm that finds a conflict-free subset with total profit $\Omega(p(T')/q)$.

2.4.1 Initial LP Rounding

For each task i and each edge $e \in span(i)$ let $S_{i,e}$ be the tasks i' spanning e with $d_{i'} \geq d_i$ and $d_i + d_{i'} > c_{e'}$ for some $e' \preceq e$ that is spanned by both i and i'. In words, $S_{i,e}$ is the collection of

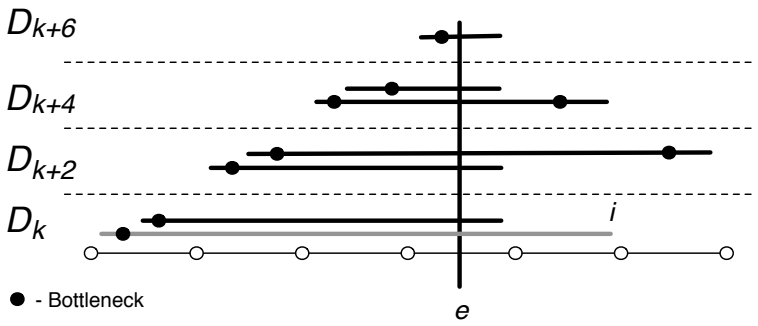

Figure 2.14: All tasks are in $S_{i,e}$ for task i and edge e in the picture, task i is drawn in gray only to help distinguish it from the other tasks in $S_{i,e}$. The height of the task corresponds to the value of its demand. The dots on the tasks indicate that the corresponding edge on the path is a bottleneck edge.

tasks that span e and conflict with i on some edge $e' \preceq e$. Similarly, define $T_{i,e}$ the same way as $S_{i,e}$ except consider edges $e' \succ e$. Figure 2.14 illustrates a set $S_{i,e}$.

Clearly any $i' \neq i$ in $S_{i,e}$ conflicts with i. Furthermore, for any two $i', i'' \in S_{i,e}$ we also have that i' and i'' conflict for the following reason. We have that i conflicts with i' across an edge $e' \preceq e$ and that i conflicts with i'' across an edge $e'' \preceq e$. Suppose that $e' \preceq e''$. Then $e'' \in span(i')$ as well. We have $d_{i''} + d_{i'} \geq d_{i''} + d_i > c_{e''}$ so i' and i'' also conflict across e''. So, at most one task from $S_{i,e}$ can appear in a feasible solution. Similarly, at most one task from $T_{i,e}$ can appear in a feasible solution. This leads us to consider the following LP relaxation for UFP.

$$\begin{aligned}
\text{maximize :} \quad & \sum_i x_i p_i & & (2.2)\\
\text{such that :} \quad & \sum_{i: e \in span(i)} x_i d_i \leq c_e & \forall \text{ edges } e\\
& \sum_{i' \in S_{i,e}} x_{i'} \leq 1 & \forall i, e \in span(i)\\
& \sum_{i' \in T_{i,e}} x_{i'} \leq 1 & \forall i, e \in span(i)\\
& x_i \geq 0 & \forall i
\end{aligned}$$

We just argued in the previous paragraph that feasible UFP solutions map to feasible integer solutions of the LP having the same value, so the optimum value of the LP is at least the optimum profit of the UFP instance. This is similar to the LP presented in [26], except we have replaced the rank constraints by the weaker constraints over sets $S_{i,e}$. Careful inspection of the results in [26] shows that the integrality gap of LP 2.2 and the LP in [26] differ only by a constant (*i.e.* the new constraints presented in LP 2.2 are the only possible constraints that can be generated by the approximate separation oracle in [26]). Since the LP in [26] has an integrality gap of $O(\log n)$, then the integrality gap of LP 2.2 is $O(\log n)$ in general UFP instances. For the specific case of q-conflicting instances, we prove that LP 2.2 actually has an $O(q)$ integrality gap. It is also interesting to note that LP 2.2 has an optimum value of 1 (as opposed to $\Omega(n)$) in the bad instance for LP 2.1 from Figure 2.1. To

see this, note that all tasks are in $T_{n,1}$ so $\sum_i x_i = \sum_{i \in T_{n,1}} x_i \leq 1$ holds.

Let x^* now denote an optimum solution to the LP with value OPT_f. Consider Algorithm 1.

Lemma 2.4.1 *The expected profit of X returned by algorithm 1 when initially called with R is at least $\frac{OPT_f}{40}$. Furthermore, X is demand class independent and if any two $i, i' \in X$ conflict then either $span(i) \subseteq span(i')$ or $span(i') \subseteq span(i)$.*

Proof. At the recursive call when i was selected and X was returned from recursively calling $T' - \{i\}$, i was selected from $T' \cap D_k$ to minimize $length(i)$. So, any other $i' \in T' \cap D_k$ that shares a common edge with i must also span either s_i or t_i. Denote by L the collection tasks in $T' \cap D_k$ at this recursive call that also span node s_i. The claim is that $\sum_{i \in L} x_i^* \leq 4$ and we assume otherwise to find a contradiction. Each $i \in L$ has a bottleneck $e \prec s_i$ or a bottleneck $e \succ s_i$. So, more than half of $\sum_{i \in L} x_i^*$ is represented by tasks with bottlenecks to the left of s_i or by tasks with bottlenecks to the right of s_i. Suppose it is the former (the latter is similar) and call such tasks L'. Let $i' \in L'$ have the rightmost bottleneck $e \prec s_i$ so every task in L' spans e. Then the total fractional demand that spans e is at least

$$\sum_{i'' \in L'} x_{i''}^* d_{i''} \geq \sum_{i'' \in L'} x_{i''}^* d_{i'}/2 > d_{i'}/2 \sum_{i'' \in L'} x_{i''}^* > 2d_{i'} > c_e$$

which contradicts the fact that x^* is feasible.

This contradiction establishes that $\sum_{i \in L} x_i^* \leq 4$ and a similar statement holds for tasks in $T' \cap D_k$ that span t_i. Let $e_{s,i}, e_{t,i}$ denote the first and last edges spanned by task i (they may be equal if i spans a single edge). Let Z denote the tasks in $T' - \{i\}$ that are in either $T_{i,e_{s,i}}$ or $S_{i,e_{t,i}}$ or are in D_k and share a common edge with i. Now, the total of the $x_{i'}^*$ values of tasks in $T_{i,e_{s,i}}$ is at most 1 as is the total of the $x_{i'}^*$ values of tasks in $S_{i,e_{t,i}}$ by the LP constraints. Therefore, the total of all the $x_{i'}^*$ values for tasks in Z is at most 10. Since any individual task i' is added to X with probability at most $x_{i'}^*/20$, then by union bound the probability that $Z \cap X \neq \emptyset$ is at most $10/20 = 1/2$. So:

$$\Pr[i \in X] = \Pr[i \in X | Z \cap X = \emptyset] \cdot \Pr[Z \cap X = \emptyset] \geq \frac{x_i^*}{20} \cdot \frac{1}{2} = \frac{x_i^*}{40}$$

By linearity of expectation, the expected profit of tasks in the set X returned by RoundLP(T) where T is the set of all tasks is then at least $\frac{OPT_f}{40}$. That X is demand class independent and satisfies $span(i) \subseteq span(i')$ or $span(i') \subset span(i)$ for every conflicting $i, i' \in X$ follows by construction. □

2.4.2 Picking a Feasible Subset

There is a simple randomized rounding procedure that returns a $\frac{1}{16q}$-fraction of the total value of all of the tasks in the set X returned from algorithm 1 when called with the set of all tasks R. In the following algorithm, for any task $i \in X$, let C_i be the collection of tasks i' that conflict with i and have $length(i') \geq length(i)$.

Algorithm 1 RoundLP(T')

if $T' = \emptyset$ **then return** $\emptyset$
Let k be minimum such that $T' \cap D_k \neq \emptyset$
Let $i \in T' \cap D_k$ be such that $length(i)$ is minimum
$X \leftarrow RoundLP(T' - \{i\})$ ▷ recursively call this algorithm
Let $e_{s,i}$ and $e_{t,i}$ denote, respectively, the first and last edges spanned by task i
if $T_{i,e_{s,i}} \cap X = S_{i,e_{t,i}} \cap X = \{i\}$ **and** no task $i' \in X \cap D_k$ has $span(i) \cap span(i') \neq \emptyset$ **then** add i to X with probability $\frac{x_i^*}{20}$
return X

Algorithm 2 Prune(X)

Order the tasks in X as $i_1, i_2, \ldots, i_{|X|}$ in decreasing order of $t_i - s_i$.
$F \leftarrow \emptyset$
for $l = 1 \ldots |X|$ **do**
    if $F \cap C_{i_l} = \emptyset$ **then** add i to F with probability $\frac{1}{2q}$
end for
Use Lemma 2.3.3 to find a feasible subset F' of F of total profit at least 1/4 the total profit of F.
return F'

Lemma 2.4.2 *Suppose the tasks in X are q-conflicting, demand class independent, and satisfy the property that among conflicting tasks we have the span of one is contained in the span of the other. Then Algorithm 2 returns a feasible subset of X whose total profit is at least a $\frac{p(X)}{16q}$.*

Proof. Consider a task i and let e have the least capacity among edges in $span(i)$. For every $i' \in C_i$, it must be that $span(i) \subset span(i')$ so i' also spans e. Since e has the least capacity among edges spanned by i, then it must be that $d_i + d_{i'} > c_e$. Because X is q-conflicting, we must have $|C_i| \leq q$. Since the probability that any task is added to F is at most $\frac{1}{2q}$, then by the union bound we have that the probability that F contains some task in $C_i - \{i\}$ is at most $\frac{1}{2}$. Task i is then added to F with probability at least $\frac{1}{2} \cdot \frac{1}{2q} = \frac{1}{4q}$.

Now, F, being a subset of X, is demand class independent. By construction, F is also conflict-free. Thus, by Lemma 2.3.3, we may find a feasible subset F' of F whose total profit is at least $\frac{1}{4}$ of the total profit in F. That is, the total profit of F' is at least $\frac{1}{16q}$ times the total profit of X. □

By composing Lemmas 2.4.1 and 2.4.2, we arrive at the main result of this section.

Theorem 2.4.3 *There is a polynomial time algorithm that finds a feasible subset of demands whose expected total profit is at least $\frac{OPT_f}{640q}$.*

Since linear program 2.2 is an LP-relaxation for UFP, then this proves Theorem 2.0.2.

2.5 Recent Developments

Since the result of this chapter were obtained, Bonsma, Schulz, and Wiese discovered a polynomial-time constant factor approximation algorithm for UFP [19] that does not require any extra assumptions. Specifically, for any constant $\epsilon > 0$, they present a polynomial-time approximation algorithm

with approximation ratio $7 + \epsilon$. They also showed strong NP-hardness for UFP which rules out the possibility of an FPTAS assuming $\mathrm{P} \neq \mathrm{NP}$.

In their approximation, they partition the tasks into slack and tight tasks. However, they have two levels of slack tasks: for some constant $0 < \delta \leq 1/2$ they call a task i *small* if $d_i \leq \delta \cdot cap(i)$, *medium* if $\delta \cdot cap(i) < d_i \leq cap(i)/2$, and *large* if $cap(i)/2 < d_i$. Using the result of Chekuri *et al.* in [26] on approximating small tasks, the optimum solutions to the small and medium tasks can be approximated within a constant factor. Bonsma *et al.* refine this approach to improve the constants by introducing slightly different constraints in the LP relaxation to approximate small tasks. For the medium tasks, they use a dynamic programming routine similar to the dynamic programming used in by Chakrabarti *et al.* [23] for approximating UFP under the no bottleneck assumption. The slack in the capacities is used in a manner analogous to how [23] exploited the no bottleneck assumption. These two approaches for small and medium tasks are combined to provide a $3 + \epsilon$ approximation for the small and medium tasks collectively for any constant $\epsilon > 0$.

Since the the small and medium tasks in [19] already had a constant-factor approximation from [26], the greatest contribution of [19] is a constant-factor approximation for the large tasks (which we called tight tasks in this chapter). There are similarities between our basic approach and the basic approach in [19], we both shifted focus to approximating a different structure whose optimum solution is close to the optimum solution of the UFP instance and we both used dynamic programming to achieve this. In [19], they define the notion of a "top-drawn" subset which is subset of tasks $T' \subseteq T$ that satisfy the following propertiy. If we view each task i as an open rectangle in the plane $R(i) = (s_i, t_i) \times (cap(i) - d_i, cap(i))$, then a subset of tasks T' is said to be top-drawn if for any two $i, j \in T'$, we have $R(i) \cap R(j) = \emptyset$. It's not too hard to see that a top-drawn subset of tasks is feasible. Conversely, they show that every feasible subset of tasks can be partitioned into 4 top-drawn subsets so an algorithm for finding the optimum top-drawn subset of tasks is, in turn, a 4-approximation for large instances of UFP. They then show that the optimum top-drawn subset can be approximated within a factor $(1 + \epsilon)$ for any constant $\epsilon > 0$ using clever dynamic programming that exploits the geometry of top-drawn sets.

In our approach, we reduce the problem of approximating an optimum UFP solution to approximating an optimum conflict-free and demand class independent subset of tasks. Our approach relied heavily on the fact that the tasks were intersecting or d-intersecting which required us to lose a logarithmic or $O(\log_d m)$ factor. The only way to ensure that the $O(\log_d n)$ is a constant factor loss is to choose $d = n^\delta$ for a small constant $\delta > 0$. However, the running time, while sub-exponential, grows much faster than polynomial. So, while our approaches are similar in spirit, the approach in [19] enjoys the fact that top-drawn instances can be found efficiently without losing an extra logarithmic factor in the approximation guarantee to find them.

Chapter 3

Traveling Salesman Paths in Asymmetric Metrics

The classic Traveling Salesman problem (TSP) is the problem of finding the cheapest Hamiltonian cycle in a symmetric metric. One well-studied variant is the Asymmetric Traveling Salesman problem (ATSP) where the goal is to find the cheapest Hamiltonian cycle in an asymmetric metric. Analogously, one may consider problems concerning cheap Hamiltonian path in either symmetric or asymmetric metrics which we generically call Traveling Salesman Path problems. We can consider variants of Traveling Salesmen Path problems where some of the endpoints are fixed and some are not. The four basic variants are when no endpoints are specified (so any Hamiltonian path will do), where two nodes s and t are specified and we require the Hamiltonian path start at s and end at t, or where only the start node s or only the end node t is specified.

It is also natural to consider variants where multiple salesmen are available. For example, large companies often have more than one salesman at their disposal and they want to schedule a route for each salesman so that every client is visited by some salesman. It is not necessary to visit a client with more than one salesman. In terms of graphs, given a positive integer k, we want to find a collection of k paths in a metric graph so each node of the graph lies on at least one of the k paths. The goal is to minimize the total cost of all paths (*e.g.* minimize the total travel cost of the salesmen). There are multiple variants of this problem because we can specify start and/or end locations of each of the k paths in advance. There is one additional variant that is interesting. Suppose we are given a set of k start locations $S = \{s_1, s_2, \ldots, s_k\}$ and a set of k end locations $\{t_1, t_2, \ldots, t_k\}$. The goal is to find a set of k paths where the start and end points of the paths establish a bijection between S and T. That is, each of the start locations is the start of precisely one path and each of the end locations is the end of precisely one path, but it may be that a path starting at s_i ends at t_j for some $j \neq i$. For example, if $S = T$, one application is to a company that maintains a fleet of identical vehicles. It does not matter where the vehicles end up after driving their route, it only matters that each depot has the same number of vehicles before and after the routes are followed. This is related to some vehicle routing problems we discuss when we mention previous work in this area.

Traveling Salesman problems have been studied extensively from the perspective of approximation algorithms. A well-known algorithm (*e.g.* [88]) approximates the classic Traveling Salesman problem within a factor 2 by simply observing that the minimum spanning tree T of the metric is a lower bound on the optimum solution (since a Hamiltonian cycle is connected) and that a depth-first search traversal of T visits all nodes and crosses each edge twice. If we report the first time we visit a node, then by the triangle inequality the resulting cycle has cost no more than the total cost of the edges traversed in the depth-first search. Christofides [32] improved on this approach by presenting a $\frac{3}{2}$-approximation for TSP. To date, this remains the best approximation algorithm for general instances of TSP. Regarding lower bounds, Papadimitriou and Yannakakis [74] first proved that TSP was APX-hard. Later, Papadimitriou and Vempala provided an explicit constant lower bound. Specifically, they proved that unless P = NP, then there is no $\left(\frac{220}{219} - \epsilon\right)$-approximation for TSP for any constant $\epsilon > 0$.

Held and Karp [52] introduced an LP relaxation for TSP (LP 1.1). Wolsey [90] and Williamson & Shmoys [85] showed that the integrailty gap of this LP is also at most $\frac{3}{2}$. There are examples for which the integrality gap of LP 1.1 is arbitrarily close to $\frac{4}{3}$ and it is a major open problem to determine if this lower bound is tight. It should also be noted that this LP can be solved in polynomial time since separating over the cut constraints can be accomplished using a minimum cut algorithm.

Special cases of TSP have also been considered. Bermen and Karpinski [16] show that the problem admits an $\frac{8}{7}$-approximation when all distances are 1 or 2 (as in the strong NP-hardness proof in Section 1.2.8), improving on a previous bound of $\frac{7}{6}$ by Papadimitriou and Yannakakis [74]. When the nodes in the metric $G = (V, E)$ are points in the Euclidean plane and the distance between any two points is their Euclidean distance, both Arora [6] and Mitchell [67] show the problem has a PTAS.

Another special case that has been considered is when the distance d_{uv} is equal to the length of the shortest path from u to v in a connected, unweighted graph (sometimes called *Graphic TSP*). Very recently, Oveis Gharan, Saberi, and Singh [72] showed that the integrality gap of LP 1.1 is $\frac{3}{2} - c$ for some very small constant $c > 0$. Even more recently, Mömke and Svensson [69] improved this bound to 1.461. When we further restrict the input of Graphic TSP to metrics obtained from the shortest path metrics of 3-edge connected and cubic graphs, Aggarwal *et al.* [1] show that the integrality gap of the LP relaxation is at most $\frac{4}{3}$.

A variant of TSP where our goal is to find a Hamiltonian path has also been studied. Hoogeveen [54] showed that if no endpoints are specified or if at most one endpoint is specified, then the problem can still be approximated within $\frac{3}{2}$. In the case that both endpoints are specified, a $\frac{5}{3}$-approximation is shown. Later, An and Shmoys [3] analyzed his algorithm in a different way and bounded the integrality gap of a variant of LP 1.1 to the case of Hamiltonian paths by $\frac{5}{3}$ as well. When the metric is graphic, they also show that analysis similar to [72] demonstrates that the integrality gap of this LP is bounded by $\frac{3}{2} - c'$ for some very small constant $c' > 0$. Mömke and Svensonn [69] also show that

their analysis of the integrality gap of LP 1.1 for TSP generalizes to the case of Hamiltonian paths and demonstrates that the corresponding LP has an integrality gap of at most 1.586. The integrality gap is at least $\frac{3}{2}$ for the case of TSP Path when both endpoints are fixed.

The Asymmetric Traveling Salesman problem has also received a lot of attention. Frieze *et al.* [42] gave the first approximation algorithm for the problem with an approximation ratio of $\log_2 n$. Williamson [89] showed that the solutions produced by this algorithm also bound the integrality gap of LP relaxation 3.1 (to be introduced later) by $\log_2 n$. Bläser [17] modified this algorithm to obtain an approximation ratio of $0.999 \log_2 n$, followed by an improvement to $\frac{4}{3} \log_3 n \approx 0.842 \log_2 n$ by Kaplan *et al.* [58]. Feige and Singh [38] provided one more constant-factor improvement to $\frac{2}{3} \log_2 n$. Then, in a breakthrough paper, Asadpour *et al.* [9] finally improved the approximability by more than a constant factor and presented an $O(\log n / \log \log n)$-approximation for ATSP that also bounds the integrality gap of LP 3.1 by the same amount. One intersting special case is when the asymmetric metric comes from the shortest paths of a (perhaps weighted) directed graph that can be drawn on an orientable surface of genus γ. Oveis Gharan and Saberi [71] show that the integrality gap of LP relaxation 3.1 is $O(\sqrt{\gamma} \log \gamma)$. In particular, they also show that the integrality gap in metrics obtained from shortest paths in planar graphs (which can be embedded on the sphere, an orientable surface of genus 0) is at most 22.5.

Similar to TSP, Papadimitriou and Vempala [73] show ATSP cannot be approximated within a factor $\frac{117}{116} - \epsilon$ for any constant $\epsilon > 0$ unless P = NP. The best known lower bound for the integrality gap of LP 3.1 is 2, as shown by Charikar *et al.* [24]. It is also an important problem to determine if the upper bound on the integrality gap (or, more generally, the best polynomial-time approximation ratio) is also constant.

The first approximation algorithms for the Asymmetric Traveling Salesman Path problem appeared much later than the first approximation algorithm for ATSP [42]. Lam and Newman [62] first showed that the problem can be approximated within an $O(\sqrt{n})$-factor. The first logarithmic approximation was by Chekuri and Pal [30] with an approximation ratio of $4H_n = 4 \ln n - o(1)$. Later, Feige and Singh [38] demonstrated that the approximability of ATSP and ATSPP were within $2 + \epsilon$ for any constant $\epsilon > 0$. Combined with their ATSP approximation, this implied ATSPP can be approximated within $\left(\frac{4}{3} + \epsilon\right) \log_2 n$. None of the aforementioned algorithms bound the integrality gap of any LP relaxations. The first such bound was proven by Nagarajan and Ravi [70] for the LP relaxation we consider later, namely LP 3.3, where they proved the gap was at most $O(\sqrt{n})$. Until our work, this was the best known bound on the integrality gap.

Another variant that is studied in literature is the following (sometimes under the guise of *vehicle routing* rather than traveling salesmen). Given a symmetric metric $G = (V, E)$ and a collection of k nodes $r_1, \ldots, r_k$, the goal is to find a cycle C_i containing r_i for each $1 \leq i \leq k$ such that every node in V lies on one such cycle. The objective in this case is to minimize the total length. A 2-approximation is known for this problem which is based on the doubling-tree principle. Namely,

one can easily find a minimum cost forest where each component contains a unique root r_i. Since the optimum solution contains such a forest as a subgraph (by deleting one edge per cycle), this minimum cost forest has cost at most the optimum value OPT. After doubling the edges of this forest, we obtain an Eulerian graph for each connected component and all nodes can then be visited by traversing the edges of these graphs with cost at most $2OPT$ (see, *e.g.* [65, 80]). A similar algorithm also applies to finding paths $P_1, \ldots, P_k$ where P_i starts at r_i such that every node is on one such path.

More recently a $\frac{3}{2}$-approximation was presented by Rodrigues and Xu [83] when the number of roots k is constant. They also point out that previous technical reports that claim a $\frac{3}{2}$-approximation for this problem [77] for arbitrary k and a $\frac{5}{3}$-approximation for a path variant [78] of this problem are incorrect. For the special case $k = 2$ in the Hamiltonian path variant of the problem, Rathinam and Sengupta [79] also demonstrate a $\frac{3}{2}$-approximation.

Yet another variant is when the metric $G = (V, E)$ is symmetric and we have two subsets of nodes S, T, both of equal size (say k). Furthermore, we can assume that S and T are disjoint (there is a simple reduction from the problem with $S \cap T \neq \emptyset$ to this case). The goal is to find k paths, each of which starts at some node in S and ends at some node in T, so that all nodes in V (including S and T) are on exactly one such path. In this way, the start and end nodes of the paths establish a bijection between S and T. Rathinama and Sengupta [76] consider this problem (with the additional constraint that each path also visits some node in $V - (S \cup T)$). Using ideas from their paper, it is possible to find a minimum cost forest where each connected component contains exactly one node in S and exactly one node in T (by using matroid intersection techniques over two R-rooted spanning forest matroids from Section 3.4.2, one with $R = S$ and the other with $R = T$). By doubling each edge in this forest except edges that lie on some $s_i - t_j$ paths, we obtain an Eulerian path from s_i to t_j in each forest which is a 2-approximation for the problem.

As far as we know, all of the variants we consider in this Chapter concerning multiple traveling salesmen in asymmetric metrics, as well as the variant in symmetric metrics covered in Section 3.4.2, have not been studied before.

In this chapter, we present a variety of approximation algorithms for some variants of the Traveling Salesman problem. Unless it needs to be stated explicitly otherwise, throughout this chapter we use n to denote the number of nodes in the metric graph being considered. Primarily, we discuss Traveling Salesman Path problems in asymmetric metrics, but we also consider some other versions in the last section. Many of the algorithms can be seen as extensions of the $O(\log n)$-approximation by Frieze *et al.* [42] the for well-studied Asymmetric Traveling Salesman problem [42], so we begin by reviewing their algorithm and its analysis. We also review Williamson's analysis [89] that the integrality gap of a natural LP relaxation for ATSP is also bound by $O(\log n)$ and, moreover, that the solutions found in the $O(\log n)$-approximation by Frieze *et al.* [42] are at most $O(\log n)$ times the optimum cost of this LP relaxation. Some of the ideas in this analysis are used to prove analogous

bounds for the LP relaxations we consider for the TSP variants studied in Section 3.2 and Section 3.3. These bounds are used later in Chapter 4.

The first new contribution in this chapter is another $O(\log n)$-approximation for the Asymmetric Traveling Salesman Path problem (ATSPP). As mentioned, before Feige and Singh already demonstrated that ATSPP can be approximated within a factor $O(\log n)$ [38] (which has since been improved to $O(\log n/\log\log n)$ due to results in [9]), but their result did not bound the integrality gap of a natural LP relaxation. We show that our algorithm also bounds the integrality gap of the LP relaxation for ATSPP by $O(\log n)$, improving on the bound $O(\sqrt{n})$ by Nagarajan and Ravi [70]. This is also a crucial ingredient for approximating the Minimum Directed Latency problem we consider in Chapter 4.

Next, we consider the variant of ATSPP where we are given two nodes s and t and integer k and the goal is to find k paths from s to t whose union includes all the nodes. Our ATSPP algorithm is modified to provide an $O(k \log n)$-approximation for this problem. While it is unfortunate that the approximation ratio depends on k linearly, it still demonstrates a logarithmic approximation algorithm for the interesting case when two salesmen are available. More generally, we exhibit the following: for any integer $b \geq 1$ there is a polynomial-time algorithm that finds at most $(1 + \frac{1}{b}) \cdot k$ paths whose total cost is at most $O(b \log n)$ times the optimum value of a linear programming relaxation. To the best of our knowledge, these results are the first approximation algorithms for Traveling Salesman Path problems in asymmetric metrics with multiple salesmen.

We discuss how to extend these results to algorithms when none of the endpoints are fixed or where the start or endpoints may vary. In almost all cases (including the case of the paths establishing a bijection between start nodes S and end nodes T), the approximation guarantees translate almost identically. The most notable exception is when we have specific start and end locations s_i, t_i for each path and we require that the path starting at s_i to end at t_i. In this case, we show that no polynomial-time algorithm can guarantee any bounded-ratio approximation unless P = NP. This is also interesting since the same variant in symmetric metrics has a very simple 3-approximation and the same variant in asymmetric metrics with $s_i = t_i$ for every $1 \leq i \leq k$ (so the paths can be thought of as "rooted cycles") has a logarithmic approximation. We also present these algorithms in the last section.

The bound on the integrality gap of LP relaxation 3.3 for ATSPP appeared in [43]. The bound of $O(k^2 \log n)$ on the integrality gap of LP relaxation 3.5 for k-ATSPP we develop at the end of Subsection 3.3.3 will appear in the full version of [43].

3.1 Warmup: The Asymmetric Traveling Salesman Problem

In this section, we review a classic approximation algorithm for finding Hamiltonian cycles in directed metrics by Frieze, *et al.* [42]. This variant is simply called the Asymmetric Traveling Salesman Problem (ATSP). The algorithm is simpler than our algorithm for paths but is similar enough to

provide some familiarity with our ideas. We use flows and circulations extensively throughout this chapter. The reader is referred to Section 1.2.1 for notation considering flows and circulations.

We are given a directed graph $G = (V, A)$ where each arc $e = uv \in A$ has a cost d_{uv}. First, we note that for any $W \subseteq V$ we have that the cost of the optimum Hamiltonian cycle on the metric induced by W is at most the cost of the optimum Hamiltonian cycle on V. We dub this the *monotonicity property*. To see this, suppose $v_1, v_2, \ldots, v_n$ is a Hamiltonian cycle C on V. Now, suppose $v \in V - W$ and arcs uv, vw are used in the optimum solution on V. If we replace uv, vw with uw then we obtain a cycle on $V - \{v\}$. By the triangle inequlity, $d_{uw} \leq d_{uv} + d_{vw}$ so the cost of this cycle is no more expensive than the cost of the original cycle. Repeat this for all nodes in $V - W$ (processing them in any order) to obtain a cycle on W of no greater cost than the optimum cycle on V.

A *cycle cover* of a subset of nodes W is an integral circulation C such that $C(\delta^+(v)) = C(\delta^-(v)) = 1$ (so each arc uv has $C_{uv} \in \{0, 1\}$). If one considers the subset of arcs $D = \{uv \in A : C_{uv} = 1\}$, then the indegree and outdegree of each node in the graph $G = (V, D)$ is exactly 1. That is, D is a collection of vertex-disjoint cycles $C_1, \ldots, C_k$ where each node lies on some cycle C_i and each C_i includes at least 2 nodes. Viewing a cycle cover as a subset of arcs, a cycle cover is then a base of the intersection of two partition matroids, one which bounds the indegree of each node in V by 1 and the other which bounds the outdegree of each node in V by 1. More formally, consider the two matroids $\mathcal{M}^+ = (A, I^+), \mathcal{M}^- = (A, I^-)$ where:

$$D \in I^+ \quad \Leftrightarrow \quad \forall\, v \in V,\ |\delta^+(v) \cap D| \leq 1$$
$$D \in I^- \quad \Leftrightarrow \quad \forall\, v \in V,\ |\delta^-(v) \cap D| \leq 1$$

A base in $\mathcal{M}^+$ (resp. $\mathcal{M}^-$) is a subset of arcs D where every node has precisely one outgoing (resp. incoming) arc in F. Thus, the minimum cost cycle cover of V can be found in polynomial time using matroid intersection techniques to find base that is common to both $\mathcal{M}^+$ and $\mathcal{M}^-$.

We begin with a high-level overview of the approximation algorithm for ATSP by Frieze *et al.* [42]. Begin by computing an optimum cycle cover C on V. Discard all but one node from each cycle in the support of C and repeat with the remaining nodes until only a single node remains. The number of nodes is at least halved at each step since each cycle has at least two nodes, so the overall number of iterations is at most $\log_2 n$. Once there is one node remaining, then the graph on V using only the arcs in the support of the cycle covers we found throughout the algorithms execution is Eulerian. Follow an Eulerian circuit, but bypass some arcs of this circuit to ensure each node is only visited once. The cost is bounded by at most $\log_2 n$ times the cost of an optimum solution since the Eulerian circuit is the union of the support of at most $\log_2 n$ cycle covers and since a cycle cover on a subset of nodes costs no more than the optimum Hamiltonian cycle on all of V. The full description is found in Algorithm 3.

Algorithm 3 An $O(\log n)$-approximation for ATSP [42]

1: Let $D_v \leftarrow 0,\ \forall v \in V$
2: Let $W \leftarrow V$
3: **while** $|W| > 1$ **do**
4: Let D' be a minimum cost cycle cover of W
5: Let $C_1, C_2, \ldots, C_k$ be the cycles in the support of D'
6: Choose any single node v_{C_i} from each cycle C_i to "represent" that cycle
7: Update $D \leftarrow D + D'$ ▷ $D \cap D' = \emptyset$ before this step
8: Update $W \leftarrow \{v_{C_1}, v_{C_2}, \ldots, v_{C_k}\}$
9: **end while**
10: Let $G' = (V, \text{support}(D))$
11: Find an Eulerian circuit C of G' that visits all nodes
12: Remove all but one occurence of each node $v \in V$ from C to get a Hamiltonian cycle C'
13: **return** C'

Lemma 3.1.1 *The graph $G' = (V, D)$ computed on line 10 of Algorithm 3 has an Eulerian circuit that visits all nodes.*

Proof. Each cycle cover D' satsifies $D'(\delta^+(v)) = D'(\delta^-(v)) = 1$ for every $v \in V$. By induction on the number of iterations and since D is initially 0 on all $v \in V$ we see that $D(\delta^+(v)) = D(\delta^-(v))$ at the end of the main loop. Now, each arc $uv \in A$ has $D_{uv} \in \{0, 1\}$ since if $D'_{uv} = 1$ for any iteration, then either u or v is discarded from W and uv is not used in any future circulation. In other words, $uv \in \text{support}(D)$ if and only if $D_{uv} = 1$. Thus, the degree requirements for G' are satisfied. It is a commonly-known fact that a directed graph with equal indegree and outdegree at each node has an Eulerian circuit that includes all the nodes if and only if it is weakly connected (*i.e.* connected if the directed edges are replaced with undirected edges).

It is sufficient to prove that there is always a path to w where w is the sole node in W when the loop terminates. Let $W_0 = V$ and denote the set W just after the j'th iteration of the loop by W_j. If the number of iterations is p, then $W_p = \{w\}$. We prove by induction on $p - j$ for $0 \leq j \leq p$ that there is a path from any node in W_j to w using only arcs in the support of D. It is clearly true for $j = p$ since $W_p = \{w\}$. Now, suppose that every node in W_{j+1} has a path to w using only arcs in p. Every node $v \in W_j$ is in some cycle C_i in the support of the cycle cover D' found in iteration $j + 1$. Thus, there is a path from v to the "representative node" v_{C_i} by following arcs in the cycle C_i (which are also arcs in D). From here, there is a path to w using only arcs in D by induction since $v_{C_i} \in W_{j+1}$. Thus, for $j = 0$ we see that every node in $W_0 = V$ has a path to w using only arcs in D. □

Theorem 3.1.2 *The cycle C' returned by Algorithm 3 has cost at most $\log_2 n \cdot OPT$ where OPT is the minimum cost of a Hamiltonian cycle on V.*

Proof. The cost of each cycle cover D' found in each iteration is at most OPT because D' is a minimum cost cycle cover on W, because the optimum Hamiltonian cycle on W in each iteration is

trivially a cycle cover (when viewed as an integral circulation), and because the optimum Hamiltonian cycle on W has cost no more than the optimum Hamiltonian cycle on V. The number of nodes of W kept in each iteration is at most $|W|/2$ since each cycle C_i in the cycle cover involves at least 2 nodes and we discard all but one node in each cycle from W at the end of each iteration. Thus, the the number of iterations is at most $\log_2 n$.

Since D is simply the sum (as a circulation) of the at most $\log_2 n$ cycle covers found in each iteration and each cycle cover has cost at most OPT, then the total cost of the circuit C is at most $\log_2 n \cdot OPT$. The final cycle C' is obtained from the circuit C by shortcutting past some nodes, so the triangle inequality yields that the cost of C' is at most the cost of C. □

Now, consider the following integer programming formulation for ATSP. We have a variable x_{uv} for each arc $uv \in A$ and constrain x_{uv} to take only values 0 and 1. The idea is that $x_{uv} = 1$ if we use uv in the Hamiltonian cycle and $x_{uv} = 0$ if we do not use uv in the Hamiltonian cycle. For a subset of arcs D, we let $x(D) = \sum_{uv \in D} x_{uv}$ (similar to our notation $F(D) = \sum_{uv \in D} F_{uv}$ for a flow or circulation F). Then the integer program is:

$$\text{minimize :} \quad \sum_{e \in A} d_{uv} x_{uv} \tag{3.1}$$

$$\begin{aligned} \text{subject to :} \quad & x(\delta^+(v)) = 1 && \forall v \in V \\ & x(\delta^-(v)) = 1 && \forall v \in V \\ & x(\delta^+(S)) \geq 1 && \forall \emptyset \subsetneq S \subsetneq V \\ & x_{uv} \in \{0,1\} && \forall uv \in A \end{aligned} \tag{3.2}$$

The equality constraints ensure that integer point x is a cycle cover. If one considers the subset of edges $D = \{uv \in A : x_{uv} = 1\}$, then we already discussed that D is a collection of cycles where each node $v \in V$ is on exactly one of these cycles. Then, Constraints (3.2) tell us that D cannot have more than one cycle otherwise the nodes S in such a cycle would have $x(\delta^+(S)) = 0$. So, D is actually a Hamiltonian cycle whose cost is the same as x. Conversely, if D is any Hamiltonian cycle on G, then the point x with $x_{uv} = 1$ for $uv \in D$ and $x_{uv} = 0$ for $uv \notin D$ is a feasible point in the integer program with the same cost as D.

If we replace the integer constraints $x_{uv} \in \{0,1\}, \forall uv \in A$ with $0 \leq x_{uv} \leq 1, \forall uv \in A$, we obtain a linear program known as the *Held-Karp* relaxation for ATSP [52]. Not only do we have that Algorithm 3 is a $\log_2 n$ approximation, we also see that it bounds the integrality gap of this relaxation. Before proving this, we recall a result of Frank and Jackson. Actually, the stated theorem is not quite as strong as their original result, but it is sufficient for our needs and is simpler to state. It says that for any non-isolated node v of a directed Eulerian graph, there is one incoming arc and one outgoing arc such that if we bypass v by shortcutting past these arcs, the number of outgoing arcs of any subset S not containing v remains the same. It is often called the *splitting off lemma for*

digraphs. An application of the lemma is then referred to as *splitting off*.

Theorem 3.1.3 (Frank [40] and Jackson [56]) *Let $G = (V, A)$ be a directed multigraph with the indegree and outdegree being equal at every node. For any non-isolated node w, there are two arcs uw, wv such that if we let A' denote the collection of arcs obtained by removing uw, wv from A and adding uv to A, then $|\delta^+_{A'}(S)| = |\delta^+_{A'}(S)|$ for any $\emptyset \subsetneq S \subsetneq V - \{w\}$.*

Williamson analyzed the ATSP approximation algorithm in [42] to show that it also bounds the integrality gap of LP relaxation 3.1. Specifically, he proved the following.

Theorem 3.1.4 (Williamson [89]) *Let OPT_f denote the optimum value of the linear programming relaxation of the integer program. The cost of the cycle C' returned by Algorithm 3 is at most $\log_2 n \cdot OPT_f$.*

Proof. For any $W \subseteq V$, let $LP(W)$ denote the instance of LP 3.1 on the graph induced by W. Similarly, let $OPT_f(W)$ denote the optimum value of $LP(W)$. The first step is to show that $OPT_f(W) \leq OPT_f(V)$ for any $W \subseteq V$. This follows by induction if we show that $OPT_f(W) \leq OPT_f(W \cup \{v\})$ for any $v \in V - W$.

Let x be an optimum solution to $LP(W \cup \{v\})$. We know that each x_{uv} is rational so we define $\Delta = \text{lcm}_{uv \in A}\{d_{uv}\}$ where d_{uv} is the denominator in the reduced form of x_{uv}. Then Δx_{uv} is an integer for each $uv \in A$. Consider the multigraph G_Δ on $W \cup \{v\}$ with Δx_{uv} copies of each arc uv. This graph has indegree and outdegree exactly Δ at each node v and each cut $\emptyset \subsetneq s \subsetneq W \cup \{v\}$ has $|\delta^+(S)| \geq \Delta$.

While $\delta^+(v) \neq \emptyset$, use Theorem 3.1.3 to find arcs uv, vw such that removing a copy of uv, vw and adding a copy of uw does not decrease the $|\delta^+(S)|$ for any $\emptyset \subsetneq S \subsetneq W$. Note that the total cost of all arcs does not increase by the triangle inequality. Let A' denote the resulting multiset of arcs when we finally have $\delta^+(v) = \delta^-(v) = \emptyset$ and let $\#uv$ denote the number of copies of uv in A'. We form a solution x' to $LP(W)$ by setting $x'_{uv} = \frac{\#uv}{\Delta}$. Since the indegree and outdegree of each node in W does not change throughout the splitting off operation, then $x'(\delta^+(u)) = x'(\delta^-(u)) = 1$ for every $u \in W$. Furthermore, by our choice of arcs uv, vw in each step, we also have that $x'(\delta^+(S)) \geq 1$ for each $\emptyset \subsetneq S \subsetneq W$. Thus, x' is a feasible solution to $LP(W)$.

The cost of x' is exactly the total cost of A' scaled by Δ^{-1}. This, in turn, is at most the total cost of A scaled by Δ^{-1}. Finally, the total cost of A is exactly $\Delta OPT_f(W \cup \{v\})$ since x is an optimum solution for $LP(W \cup \{v\})$. Since x' is feasible for $LP(W)$, then $OPT_f(W)$ is at most the cost of x'. However, we just saw that the cost of x' is at most $OPT_f(W \cup \{v\})$, so $OPT_f(W) \leq OPT_f(W \cup \{v\})$.

The second step is to show that the cost of the optimum cycle cover on W is at most $OPT_f(W)$. The result then follows because we already saw that the cost of C' is bounded by the total cost of at most $\log_2 n$ minimum cost cycle covers on subsets of V. To see this, remove the cut constraints $x(\delta^+(S))$ from $LP(W)$. The resulting polytope is exactly the polytope for the intersection of the

base polytopes for the two partition matroids that bound the indegree and outdegree of a node by 1. This is an integral polytope (*cf.* Section 1.2.3) and the integer solutions correspond to cycle covers of W, so the minimum cost cycle cover has cost exactly equal to the optimum value of this polytope which, in turn, is at most $OPT_f(W)$ since a feasible solution to $LP(W)$ remains feasible when constraints are removed. □

3.2 The Asymmetric Traveling Salesman Path Problem

The Asymmetric Traveling Salesman Path problem (ATSPP) is similar to ATSP except we are looking for a Hamiltonian path rather than a Hamiltonian cycle. In this section, we demonstrate an $O(\log n)$-approximation algorithm for ATSPP. What distinguishes this algorithm from previous approximation algorithms is that we can also bound the integrality gap of a natural LP relaxation of ATSPP by $O(\log n)$ using our algorithm.

Our algorithm is also similar in spirit to the $O(\log n)$-approximation for ATSP by Frieze *et al.* [42] that was presented in the previous section. We build on this algorithm by first describing a structure similar to the cycle covers in the ATSP algorithm called a path/cycle cover. As before, we choose a representative from each cycle and discard all but the representative node from each cycle. However, for our algorithm to work we cannot arbitrarily choose any representative from each cycle. We introduce a potential function called a "label" for each node and choose a representative based on this function. After finding sufficiently many path/cycle covers, we prove that we can find a cheap path that includes all the nodes that were not discarded in some iteration. Then, using the cycles found across the iterations, we graft the discarded nodes onto the path.

The rest of the section is organized as follows. First, we precisely define what is meant by a path/cycle cover and discuss how to find one optimally. We also demonstrate why a simple generalization of the ATSP algorithm to ATSPP using path/cycle covers may not succeed. We can then present the algorithm which is similar to the ATSP algorithm, except representatives for the cycles are chosen differently and the final path is constructed a bit differently. Properties concerning the node labels are then proven to demonstrate the correctness and approximation ratio of the algorithm. Finally, we concluded this section by presenting a natural LP relaxation and prove how the algorithm bounds the integrality gap of this relaxation by $O(\log n)$.

3.2.1 Path/Cycle Covers

Suppose $G = (V, A)$ is a directed metric and $s, t \in V$ are given in advance with $s \neq t$. The goal is to find a Hamiltonian path that starts at s and ends at t. A good starting point is to define the notion of a path/cycle cover.

Definition 3.2.1 *A path/cycle cover of $G = (V, A)$ with a specified start node s and a specified end node t is an integral $s - t$ flow F such that:*

- $F(\delta^+(v)) = F(\delta^-(v)) = 1$ *for each* $v \in V - \{s, t\}$
- $F(\delta^+(s)) = F(\delta^-(t)) = 1$
- $F(\delta^-(s)) = F(\delta^+(t)) = 0$

If F is a path/cycle cover, then let $D = \{uv \in A : F_{uv} = 1\}$ be the support of F. Similar to the support of cycle covers in Section 3.1, D consists of an $s - t$ path P and cycles $C_1, \ldots, C_k$ such that every node $v \in V$ is on precisely one of $P, C_1, \ldots, C_k$.

The support of a path/cycle cover is a common base of two partition matroids; one which bounds the indegree of each $v \neq s$ by 1 and the indegree of s by 0, and another which bounds the outdegree of each $v \neq t$ by 1 and the outdegree of t by 0. As with cycle covers in Section 3.1, this means an optimum path/cycle cover of a directed graph can be found in polynomial time using matroid intersection techniques. Note that a Hamiltonian $s - t$ path is trivially the support of a path/cycle cover (with no cycles), so the cost of an optimum path/cycle cover is at most OPT.

A first approach to this problem would be to find a path/cycle cover F, remove all but one node from each cycle in F, and repeat until the support of the path/cycle cover consists of only a path from s to t with no additional cycles. From this, we could incorporate the discarded nodes using the cycles from the supports of path/cycle covers found across all iterations. For sure, such an algorithm arrives at this case because the number of remaining nodes strictly decreases in each iteration before the final iteration. However, it is not so apparent that the number of iterations is $O(\log n)$.

One might hope that when a node v becomes part of the $s - t$ path, it then remains on the path in subsequent iterations. If this were the case, we could still argue that the number of nodes that are not on the path is halved at each step. Unfortunately, this is not always the case; it is possible that a node is part of the path in one iteration and is then included in a cycle in the next iteration. See Figure 3.1 for one such example. The metric is the shortest paths metric of the graph in the first image. The first path/cycle cover requires 4 edges and the 4 edges shown in the second image have cost 1, so it is an optimum path/cycle cover. Suppose the node that was discarded from the cycle is the grayed out node in the third image. It is easy to verify that the path/cycle cover shown in the last image has cost 5 while all other path/cycle covers have cost 6, so the path/cycle cover shown in the last image is optimum.

Still, the first part (the main loop) of our algorithm is similar to Algorithm 3 for Hamiltonian cycles. One of the main differences is that we no longer choose an arbitrary representative of a cycle in each iteration; we will be more careful. Another major difference is that instead of simply maintaining a set of nodes W across the iterations, we also maintain an acyclic $s - t$ flow F on W that is composed of parts of previous path/cycle covers. Denoting the set of nodes that remain after all iterations by W, the following holds after the main loop because of how we choose the representatives. For any two $u, v \in W$, there is either a path from u to v or a path from v to u in the

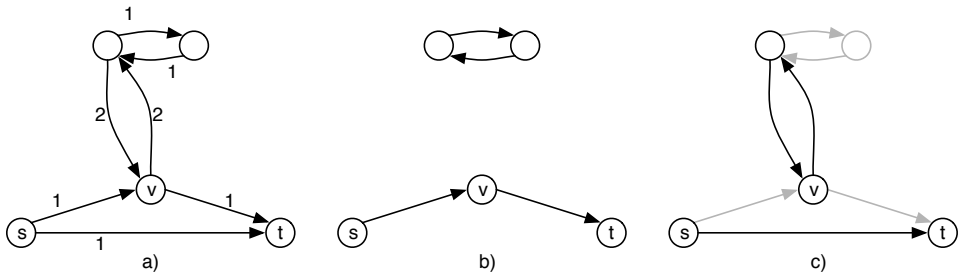

Figure 3.1: a) The graph whose shortest paths defines the metric. b) The support of the first path/cycle cover. c) The support of the second path/cycle cover (the first is grayed out).

support of the acyclic flow F. This helps us reconstruct a Hamiltonian path using only arcs found in some path/cycle cover.

3.2.2 A Logarithmic Approximation for ATSPP

In the algorithm we use H to denote a circulation. A connected component A of the support of H is a strongly connected component in the support of H that includes at least two nodes; we do not consider isolated nodes in the support of H. Finally, if F is an integral flow over nodes V, then we may form a multigraph $G = (V, F)$ where we have F_e copies of each arc e.

The algorithm for approximating ATSPP is Algorithm 4. As mentioned earlier, the algorithm maintains a set W of nodes that always contains s and t and an acyclic flow F from s to t involving only nodes in W. At each step of the algorithm, a path/cycle cover F' of the nodes in W is found which is then added to F. The innermost loop starting at Step 10 chooses a representative for each connected component A in the support of circulation H in a particular manner and discards all other node in A from W. It might be that some nodes in A also support some of the flow in the acyclic flow F, so we modify the flow F to bypass such nodes by shortcutting. This can be naively (*i.e.* without using the splitting off lemma). The sets S_v are only to simplify notation in the proof. We ultimately use the sets S_w to show that every node $w \in W$ at the end of the main loop supports a lot of flow in F which helps us construct the final path.

First, we prove that we can always find an Eulerian walk from s to t in the multigraph $(V, A_P + C)$.

Lemma 3.2.2 *The multigraph $(V, A_P + C)$ in Step 23 has an Eulerian walk from s to t that includes all nodes in V.*

Proof. Since C is a circulation, then the indegree and outdegree of each node $v \in V$ using the arcs from C are equal. Since P is a path then the indegree and outdegree of each node $v \in V - \{s, t\}$ using the arcs A_P are equal. Finally, the indegree of s is one less than its outdegree and the outdegree of t is one less than its indegree since P starts at s and ends at t. Therefore, the degree requirements for such a walk are satisfied.

Algorithm 4 An $O(\log n)$-approximation for ATSPP

1: Let $W \leftarrow V$
2: Let $S_v \leftarrow \{v\}, \forall v \in V$ ▷ only needed for the correctness proof
3: Let $l_v \leftarrow 0, \forall v \in V$
4: Let $F_{uv} \leftarrow 0, \forall u, v \in V$
5: Let $C_{uv} \leftarrow 0, \forall uv \in V$
6: **for** $2\lfloor \log_2 n \rfloor + 1$ iterations **do**
7: Find a minimum-cost path/cycle cover F' on W
8: $F \leftarrow F + F'$
9: Subtract a circulation H from F so F is acyclic again
10: **for** each connected component A in the support of H **do**
11: For each vertex $u \in A$, let d_u denote the total flow in H entering u (*i.e.* $H(\delta^-(u))$).
12: Let $v_A \leftarrow \arg\min_{u \in A} l_u + d_u$ ▷ breaking ties arbitrarily
13: **for** each node $w \in A - \{v_A\}$ **do**
14: Shortcut the flow in F over w so w supports no flow in F
15: **end for**
16: $W \leftarrow W - (A - \{v_A\})$
17: $l_{v_A} \leftarrow l_{v_A} + d_{v_A}$
18: $S_{v_A} \leftarrow \cup_{v \in A} S_v$
19: **end for**
20: $C \leftarrow C + H$
21: **end for**
22: Let P be a topological ordering of the nodes in the multigraph (W, F) formed from flow F
23: Viewing P as an acyclic $s - t$ flow, let P' be an Eulerian walk from s to t in the multigraph $(V, P + C)$
24: Shortcut past repeated nodes in P' to get a Hamiltonian path X
25: **return** X

To prove the connectivity requirement, it suffices to prove that the graph $(V, A_P + C)$ is weakly connected. This follows from showing each node $v \in V$ has a path to t using only edges in $A_P + C$. The argument is near identical to the analogous argument in Lemma 3.1.1. The idea is that all nodes in W trivially have a path to t by following only A_P. The other nodes can reach t by first following the cycle found when they were removed to the representative of that cycle which, inductively, has a path to t. □

The next two lemmas collectively say that each node in W at the end of the main loop supports more than half of the flow from s to t in F.

Lemma 3.2.3 *We have $F(\delta^+(v)) = F(\delta^-(v)) = 2\lfloor \log_2 n \rfloor + 1 - l_v$ for each $v \in W - \{s, t\}$ after the main loop.*

Proof. For each $v \in W - \{s, t\}$, both $F(\delta^+(v))$ and $F(\delta^-(v))$ increase by 1 in each of the $2\lfloor \log_2 n \rfloor$ iterations when the new path/cycle cover is added. Whenever $v \in W - \{s, t\}$ is chosen as a representative node, both $F(\delta^+(v))$ and $F(\delta^-(v))$ decrease by the amount of flow d_v entering v from the circulation H because we removed this circulation from the flow. However, this decrease is precisely the amount that l_v increases in that iteration. Finally, we note that when a node is bypassed in step 14, it is discarded from W and the $F(\delta^+(v))$ and $F(\delta^-(v))$ values for any other $w \in W$ do

not change during this bypass. □

Lemma 3.2.4 *Each node $v \in W - \{s,t\}$ has $l_v \leq \lfloor \log_2 n \rfloor$ throughout the algorithm.*

Proof. We do this by proving, in each iteration of the loop, that $2^{l_v} \leq |S_v|$ for each $v \in W$. We also maintain the invariant $S_u \cap S_v = \emptyset$ for any $u, v \in W$. Both statements are initially true because $S_v = \{v\}$ and $l_v = 0$ before the first iteration of the outer loop. Inductively, consider a step in the algorithm when l_v is updated because v was chosen as the representative of some circulation A.

Let $\overline{F}$ be the flow F at the start of the current iteration of the outer loop. That is, $\overline{F}$ is the acyclic flow before the path/cycle cover F' is added. Consider some connected component A of the circulation H removed from $F = \overline{F} + F'$. First, we claim that there are distinct nodes a, b such that $d_a = d_b = 1$.

Consider a topological ordering of the nodes based on the acyclic flow $\overline{F}$. Let a be the first node in this ordering that appears in A and let b be the last node in this ordering that appears in A. Note that $a \neq b$ since we only considered connected components A of the support of H that involve at least two nodes. We have $d_a, d_b \geq 1$ since both a and b support some flow in A. Since all other nodes in A appear after a in this topological ordering, then there is no arc ua for $u \in A$ with $\overline{F}_{ua} > 0$. Thus, the only flow that can enter a from H is from the path/cycle cover F' that was introduced in this iteration. However, $F'_{uv} \leq 1$ for any arc uv so it must be that $H(\delta^+(a)) = 1$. In a similar manner, $d_b = 1$ because no node $v \in A$ can have $\overline{F}_{bv} > 0$ since b appears latest in the topological ordering of nodes in A and because only one unit of flow from F' can exit b in the path/cycle cover F'. The result for b then follows because

$$1 = H(\delta^+(b)) = H(\delta^-(b)) = d_b$$

where the middle equality follows from flow conservation at b.

We know that just before the value of l_v is updated we have

$$l_v + d_v = \min_{u \in A} l_u + d_u.$$

Consider the nodes a, b described above (it may be that $a = v$ or $b = v$). Then we have $l_v + d_v \leq l_a + 1$ and $l_v + d_v \leq l_b + 1$. By the induction hypothesis, we have both

$$2^{l_v + d_v - 1} \leq 2^{l_a} \leq |S_a| \quad \text{and} \quad 2^{l_v + d_v - 1} \leq 2^{l_b} \leq |S_b|.$$

Since S_a and S_b are disjoint, then after the update we have $|S_v| \geq |S_a| + |S_b| \geq 2^{l_v}$ where we now refer to the new values of S_v and l_v.

That $S_u \cap S_w = \emptyset$ for any $u, w \in W$ after updating l_v and S_v follows simply because the new set S_v is formed as the disjoint union of sets S_u for $u \in A$ and the nodes $u \in A - \{v\}$ are then discarded from W. □

We are now equipped to prove the approximation ratio.

Theorem 3.2.5 *The path X returned by Algorithm 4 has cost at most $(2\lfloor \log_2 n \rfloor + 1) \cdot OPT$.*

Proof. Let P^* be an optimum Hamiltonian path on V of cost OPT. Consider a subset W of V. The optimum Hamiltonian path on W costs no more than the cost of P^* because we can shortcut past nodes in $V - W$ to transform P^* into a feasible Hamiltonian path on W of cost at most OPT. A Hamiltonian path on W corresponds to an (acyclic) path/cycle cover on W so the optimum path/cycle cover on W has cost at most OPT. Since the flow F plus the circulation C we find after the outer loop are obtained by combining $2\lfloor \log_2 n \rfloor + 1$ minimum cost path/cycle covers on subsets of W and, perhaps, shortcutting around some nodes, then the cost of F plus the cost of C is at most $(2\lfloor \log_2 n \rfloor + 1) \cdot OPT$.

Next, we claim that any arc uv used in the path P found by topologically ordering W satisfies $F_{uv} \geq 1$. If so, then the cost of P is at most the cost of F. To see this, recall that F, being an acyclic integer flow sending $2\lfloor \log_2 n \rfloor + 1$ units of flow from s to t, can be decomposed as the union of as many paths from s to t. By Lemmas 3.2.3 and 3.2.4, both u and v appear on at least $\lfloor \log_2 n \rfloor + 1$ path. By the pigeonhole principle, there then must be some path that includes both u and v. Now, v cannot appear before u on this path since u comes before v in the topological ordering of W based on F. Similarly, there cannot be another node $w \in W$ that appears between u and v on this path since, otherwise, v could not have immediately followed u in the topological ordering. Thus, the edge uv appears on a path in the path decomposition of F meaning $F_{uv} \geq 1$.

So, the cost of the path P is at most the cost of the acyclic flow F. Incorporating the circulation C, we then get an graph G with an Eulerian walk from s to t that spans all the nodes whose cost is at most the cost of F plus the cost of C. Since bypassing nodes on this walk does not increase the cost of the walk, we finally see that the cost the final Hamiltonian path X is at most $(2\lfloor \log_2 n \rfloor + 1)OPT$. □

3.2.3 A Logarithmic Bound on the Integrality Gap for ATSPP

As with ATSP, the cost of the path X returned by Algorithm 4 is at most a logarithmic factor of the optimum cost of a natural linear programming relaxation of the problem. Consider the following LP relaxation of ATSPP.

$$\text{minimize :} \quad \sum_{uv \in A} d_{uv} x_{uv} \tag{3.3}$$

$$\begin{aligned}
\text{subject to :} \quad & x(\delta^+(v)) = x(\delta^-(v)) = 1 && \forall v \in V - \{s,t\} \\
& x(\delta^+(s)) = x(\delta^-(t)) = 1 && \\
& x(\delta^-(s)) = x(\delta^+(t)) = 0 && \\
& x(\delta^+(S)) \geq 1 && \forall \{s\} \subseteq S \subsetneq V \qquad (3.4) \\
& 0 \leq x_{uv} \leq 1 && \forall uv \in A
\end{aligned}$$

If x is a feasible point with integer components, then the degree constraints say that x must be a path/cycle cover. The cut constraints then ensure that there are no cycles (otherwise the nodes of the cycle would be such a set S with outgoing cut size 0) so the path/cycle cover would indeed be a Hamiltonian path from s to t. Let OPT_f denote the optimum fractional solution to the LP relaxation above. Conversely, it is easy to see that a Hamiltonian path corresponds to a feasible integer solution to LP 3.3. Then we can bound the integrality gap in the following way.

Theorem 3.2.6 *The cost of X is at most $(2\lfloor \log_2 n \rfloor + 1) \cdot OPT_f$.*

Proof. As in the proof of Theorem 3.1.4, the result follows if we can show that the cost of every path/cycle cover found on any subset W of V is at most OPT_f. For $W \subseteq V$ with $s, t \in W$, let $LP(W)$ denote the linear programming relaxation for ATSPP on the instance induced by the nodes in W with optimum value $OPT(W)$. If we remove the cut constraints $x(\delta^+(S))$ from $LP(W)$, then we once again have the integral polytope corresponding to the intersection of the two matroids that define the minimum path/cycle cover problem. Since the optimum value of $LP(W)$ does not increase if these constraints are removed, then we only need to show that $OPT(W) \leq OPT(V)$ to establish the theorem. Again, this follows if we show $OPT(W) \leq OPT(W \cup \{v\})$ for any $v \in V - W$ where we recall that $s, t \in W$.

Essentially, this is the same argument as in Theorem 3.1.4. We just have to be careful because $1 = x(\delta^+(s)) \neq x(\delta^-(S)) = 0$ and $0 = x(\delta^+(t)) \neq x(\delta^-(t)) = 1$ so the multigraph obtained from multiplying x by Δ is not Eulerian. However, if we add precisely Δ edges from t to s in the multigraph, then we get an Eulerian graph and we can split off the arcs around v while preserving global connectivity among the nodes in W. Note that we never split off any ts arcs nor do we introduce any new ts arcs because $v \notin \{s, t\}$ and there are no tv or vs arcs. After splitting v off, we remove the Δ arcs from t to s and form the solution x' for $LP(W)$ in the same way. Removing the ts arcs does not decrease $|\delta^+(S)|$ for any S containing s because $ts \notin \delta^+(S)$. □

3.3 Multiple Traveling Salesmen

More generally, we may consider the problem of finding k paths from s to t whose union covers all of the nodes. This is called the *k-Asymmetric Traveling Salesmen Path problem* (k-ATSPP). In symmetric metrics, we usually do not save too much by using k salesmen instead of 1 for the following reason. If k is odd, then we can cover all the nodes using a single salesman at no greater cost than if we use k salesmen. The salesman could travel between s and t exactly k times by following each of the k paths in the optimum solution using multiple salesmen. The process is illustrated in Figure 3.2. If k is even, then the optimum solution using a single salesman has cost at most $(1 + 1/k)$ times the optimum solution using k salesmen for essentially the same reason. The main difference is that after following the last of the k paths in an optimum solution using k

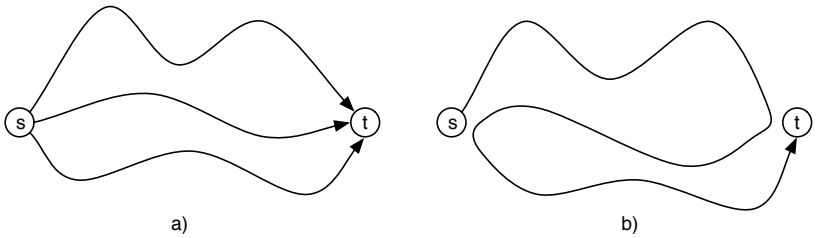

Figure 3.2: a) A sketch of a solution using 3 salesmen. b) A sketch of a similar solution of no greater cost using only one salesman.

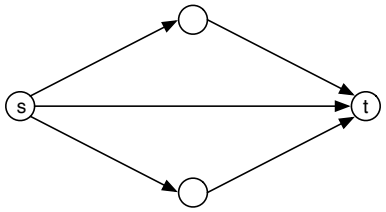

Figure 3.3: All shown edges have distance 1 and all omitted edges have distance D for arbitrarily large values D. Using one salesman requires cost at least D while the optimum solution using two salesmen is only 4.

salesmen, we are still at s. We can take one final step to t which has cost OPT/k where OPT is the optimum solution using k salesmen since surely the single step from s to t has no greater cost than the cheapest of these k paths. However, in asymmetric metrics the cost between using even only one path or two paths can be dramatically different. Consider the example in Figure 3.3 that shows the ratio of the cost between using one or two salesmen can be unbounded.

In this section, we develop a bicriteria approximation algorithm that finds approximately k paths whose total cost is within some bounded ratio of the optimum solution which uses exactly k paths. This bicriteria approximation is parameterized by a positive integer b and different bicriteria approximation guarantees result from different choices of b. On one extreme, setting $b = 1$ results in an $O(\log n)$ approximation that uses at most twice the number of given paths and, on the other extreme, setting b to be $k+1$ results in a true (not bicriteria) $O(k \log n)$-approximation using exactly k paths.

3.3.1 Preliminary Discussions and Results

The use of partition matroids allows us to consider the following useful structure:

Definition 3.3.1 *A k-path/cycle cover from s to t is an integral flow F such that $F(\delta^+(v)) = F(\delta^-(v)) = 1$ for each $v \in V - \{s,t\}$, $F(\delta^+(s)) = F(\delta^-(t)) = k$, and $F(\delta^-(s)) = F(\delta^+(t)) = 0$.*

A k-path/cycle cover can be decomposed into a collection of k paths from s to t and a collection of cycles where the cycles are disjoint from each other and from the paths and where any two paths

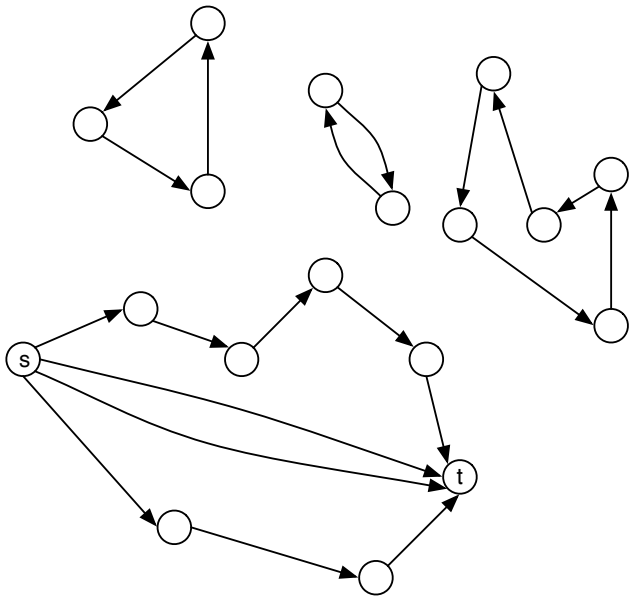

Figure 3.4: An illustration of a k-Path/cycle cover with $k = 4$.

only have s and t in common. Again, we may find the minimum cost k-path/cycle cover since these are precisely the bases common to two partition matroids (if we have k copies of the st arc). A k-path/cycle cover is illustrated in Figure 3.4

An LP relaxation for k-ATSPP in the same spirit as LP 3.3 for ATSPP is as follows. Note that we allow the st edge to be used multiple times since the optimum k-ATSPP solution may have some paths travel directly to t without visiting any other nodes.

$$\text{minimize :} \quad \sum_{e \in A} d_{uv} x_{uv} \tag{3.5}$$

$$\begin{aligned}
\text{subject to :} \quad & x(\delta^+(v)) = x(\delta^-(v) = 1 && \forall v \in V - \{s,t\} \\
& x(\delta^+(s)) = x(\delta^-(t)) = k \\
& x(\delta^-(s)) = x(\delta^+(t)) = 0 \\
& x(\delta^+(S)) \geq 1 && \forall \{s\} \subseteq S \subsetneq V \\
& 0 \leq x_{uv} \leq 1 && \forall\, uv \in A, u \neq s \text{ or } v \neq t \\
& 0 \leq x_{st} \leq k
\end{aligned} \tag{3.6}$$

We consider more general approximation algorithms for the case of k Traveling Salesmen.

Definition 3.3.2 *An (α, β)-bicriteria approximation algorithm for k-ATSPP is a polynomial-time algorithm which finds between k and $\beta \cdot k$ paths from s to t whose union spans all nodes where the total cost of these paths is at most $\alpha \cdot OPT$ where OPT is the cost of the optimum solution using exactly k paths.*

In particular, an $(\alpha, 1)$-bicriteria approximation is simply an α-approximation since it uses no more than k paths.

The main result of this section is the following:

Theorem 3.3.3 *For any integer $b \geq 1$, there is a polynomial time (in n and b) $(O(b \log n), (1 + \frac{1}{b}))$-bicriteria approximation for k-ATSPP. In particular, the cost of the paths found by this algorithm are at most $((b+1)\log_2 n)OPT_f$ where OPT_f is the optimum value of the linear programming relaxation 3.5.*

Of interest are the following two special cases:

Corollary 3.3.4 *There is an $(O(\log n), 2)$-bicriteria approximation for k-ATSPP.*

Proof. Choose $b = 1$. □

Corollary 3.3.5 *There is an $O(k \log n)$-approximation for k-ATSPP.*

Proof. Choose $b = k + 1$ and notice that $k \cdot (1 + \frac{1}{k+1}) < k + 1$. Since the number of paths is an integer, then there are precisely k paths. □

We break the presentation of the algorithm into the two phases. The first phase of the algorithm is very similar to the first phase of Algorithm 4 for ATSPP. A modification of the second phase of Algorithm 4 can be used to obtain an $O(k^2 \log n)$-approximation for k-ATSPP, but getting the ratio down to $O(k \log n)$ requires a different approach to this second phase. So, the second phase is presented first as a simple modification of the steps after the outer loop in Algorithm 4 to get an $O(k^2 \log n)$ approximation in the special case $b = k + 1$. Then we present the final version of phase 2 that works for any value $b \geq 1$.

3.3.2 Phase 1

Let $b \geq 1$ be an integer, this is the b in the statement of Theorem 3.3.3. For notational convenience, we let L be $(b+1)\lfloor \log_2 n \rfloor$ for the remainder of this section. Consider Algorithm 5, a variant of Algorithm 4.

The statements and proofs of Lemmas 3.2.3 and 3.2.4 carry over essentially word for word. One argument in Lemma 3.2.4 required $F'(uv) \leq 1$ which is no longer the case for every arc $uv \in A$. However, it is still true that $F_{uv} \leq 1$ for any arc $uv \neq st$. Since uv has $v \neq t$ in the proof, the argument still works. The fact that the circulation can be incorporated into the paths P and then shortcut is also similar because the indegrees and outdegrees of each node (except s and t) using arcs in C are equal and the graph using arcs in C is weakly connected. Finally, we similarly have that the cost of F plus the cost of C is at most $L \cdot OPT$ since k paths spanning V can be shortcut to k paths spanning a subset W of V and that k paths spanning W trivially form a k-path/cycle cover of W.

Algorithm 5 An $(O(b \log n), (1 + \frac{1}{b})k)$-bicriteria approximation for k-ATSPP

Let $L \leftarrow (b+1)\lfloor \log_2 n \rfloor$
Let $W \leftarrow V$
Let $S_v \leftarrow \{v\}, \forall v \in V$
Let $l_v \leftarrow 0, \forall v \in V$
Let $F_{uv} \leftarrow 0, \forall u, v \in V$
Let $C_{uv} \leftarrow 0, \forall uv \in V$
for L iterations **do**
    Find a minimum-cost k-path/cycle cover F' on W
    $F \leftarrow F + F'$
    Remove a circulation H from F so F is acyclic again
    for each connected component A in the support of H **do**
        For each vertex $u \in A$, let d_u denote the total flow in H entering u.
        Let $v_A \leftarrow \arg\min_{u \in A} l_u + d_u$ ▷ breaking ties arbitrarily
        for each $w \in A - \{v_A\}$ **do**
            Shortcut the flow in F over w so w supports no flow in F
        end for
        $W \leftarrow W - (A - \{v_A\})$
        $l_{v_A} \leftarrow l_{v_A} + d_{v_A}$
        $S_{v_A} \cup_{v \in A} S_v$
    end for
    $C \leftarrow C + H$
end for
Use phase 2 to find a collection P of at most $k \cdot (1 + \frac{1}{b})$ paths whose union spans W
Add the circulation C to the arcs used by P to get a multiset of edges C'
Find at most $k \cdot (1 + \frac{1}{b})$ walks P' whose union covers each arc uv exacly $C'(uv)$ times
Bypass repeated nodes in P' to get at most $k \cdot (1 + \frac{1}{b})$ paths X whose union spans all nodes
return X

In the warm up to phase 2, we only consider the case $b = k + 1$. We prove that the cost of the k paths P found is at most $O(k)$ times the cost of F which implies the cost of P plus the cost of C is at most $O(kL) \cdot OPT = O(k^2 \log n) \cdot OPT$. In the actual phase 2, we consider any integer $b \geq 1$ and show that we can find between k and $(1 + \frac{1}{b})k$ paths that spann all nodes in W whose cost is at most the cost of F (without losing an $O(k)$ factor).

3.3.3 Warmup To Phase 2

Consider the specific case $b = k + 1$. As this is only a warm up to the main result, some details are only sketched. After the outer loop is complete, F is an acyclic integer flow of kL units from s to t. Form the auxiliary graph $G' = (W, A')$ where $uv \in A'$ if and only if there is a path from u to v using only edges in the support of F. Since F is acyclic and since the relation "u has a path to v" is transitive, then G' is a partially-ordered set. An antichain in this partially-ordered set is a subset T of W such that for any $u, v \in T$ we do not have $uv \in A'$. A chain cover is a collection of paths in G' whose union spans all nodes. We use the following classic result.

Theorem 3.3.6 (Dilworth's Theorem, *e.g.* [48]) *Let T be any antichain and P be a chain cover using l chains of a partially-ordered set. Then $|T| \leq l$ and we have equality if T is a maximum*

antichain and P is a covering using the fewest paths possible.

This dual relationship between antichains and chain covers allows us to prove the following.

Theorem 3.3.7 *G' can be covered by k chains.*

Proof. Suppose this is not true. Then Dilworth's Theorem says that there is an antichain I with $|I| \geq k+1$. Consider a decomposition of F into kL paths from s to t. By the analogs of Lemmas 3.2.3 and 3.2.4 for Algorithm 5, each node is involved in at least $L - \lfloor \log_2 n \rfloor$ paths. Since I is an antichain, then no two paths may pass through a common node in I so the total number of paths is at least

$$|I|(L - \lfloor \log_2 n \rfloor) \geq (k+1)((k+2)\lfloor \log_2 n \rfloor - \lfloor \log_2 n \rfloor) > k(k+2)\lfloor \log_2 n \rfloor = kL$$

which is a contradiction. □

It is also possible to find these k paths efficiently. The total cost of these k paths is bounded by k times the cost of the flow F. The reason for this weaker bound is that the edges used in these paths are edges in A' which correspond to paths in F. So, each edge of F could potentially be used in all k paths we just found which is why we can only prove the weaker bound.

3.3.4 Phase 2

Rather than relying on the transitive closure of the acyclic flow F, we can find fewer edge-disjoint paths that span all of the nodes in W in the support of F. Currently, we can partition F into $kL = k(b+1)\lfloor \log_2 n \rfloor$ paths from s to t whose union spans W, but we want to reduce the number of paths significantly using only edges in the support of F while still including every node in W. We see that this is possible because each $w \in W$ supports a significant amount of this flow. For now, let D be some fixed (perhaps non-integer) value that we specify later. The main object of concern in this phase is the following polytope $\mathcal{P}(D)$ with a variable z_{uv} over $\mathbb{R}$ for every ordered pair of distinct nodes.

$$\begin{aligned}
\mathcal{P}(D): \quad z(\delta^+(w)) = z(\delta^-(w)) &= 1 && \forall w \in W - \{s,t\} \qquad (3.7)\\
z(\delta^+(s)) = z(\delta^-(t)) &= D \\
z(\delta^-(s)) = z(\delta^+(t)) &= 0 \\
0 \leq z_{uv} &\leq F_{uv} && \forall \text{ ordered pairs } u, v \in V
\end{aligned}$$

Note that any point z in this polytope with integer coordinates corresponds to a D-path/cycle cover (which is only possible if D is an integer). Since the support of F is acyclic and the support of z is required to be a subset of the support of F, then any integer point z is actually a collection of D paths from s to t whose union spans all nodes. Thus, our goal is to find the smallest integer D for which $\mathcal{P}(D)$ has an integer point. The following property is key for phase 2.

Lemma 3.3.8 *The polytope $\mathcal{P}(D)$ is integral when D is an integer.*

Proof. There is a simple correspondence between points in $\mathcal{P}(D)$ and points in the intersection of the base polytopes for two partition matroids over the arcs where we have F_{uv} copies of each arc uv. □

So, to prove $\mathcal{P}(D)$ has an integer point for a given integer D, it suffices to prove that $\mathcal{P}(D)$ contains *any* point. That is, if there is some point z with, perhaps, rational coordinates, then there is certainly a point z' with integer coordinates since $\mathcal{P}(D)$ is integral. Note also that the bounds $z_{uv} \leq F_{uv}$ would also imply that the cost of such an integer point (*i.e.* a D-path cover) is at most the cost of F.

We require that each node in $W - \{s, t\}$ supports the same amount of flow in F, which is possible through the following lemma.

Lemma 3.3.9 *There is an acyclic integral flow F' that sends kL units from s to t whose cost is no more than the cost of F such that every node $v \in W - \{s, t\}$ supports the same amount of flow $\alpha \geq L - \lfloor \log_2 n \rfloor$ in F'.*

Proof. Let $\gamma = \max_{w \in W} l_w$ and recall that $\gamma \leq \lfloor \log_2 n \rfloor$. Now, modify the acyclic flow F in the following way. While there is some $v \in W - \{s, t\}$ with $F(\delta^+(v)) > L - \gamma$, then bypass some flow around v. More formally, choose any two arcs $uv \in \delta^-(v), vw \in \delta^+(v)$ with $F_{uv}, F_{vw} > 0$ and let $\epsilon := \min\{F_{uv}, F_{vw}, F(\delta^+(v)) - (L - \gamma)\}$. Subtract ϵ from F_{uv} and F_{vw} and add ϵ to F_{uw}. The flow F remains acyclic after this operation (since there was already a path from v to w in the support of the acyclic flow F before this operation) and the total flow incident to every other node does not change. Repeat this procedure until $F(\delta^+(v)) = L - \gamma$.

We see that F_{uv} is an integer for every edge after each modification. It is true initially since F is the sum of paths found in phase 1. When choosing ϵ at each step we have that both flow values F_{uv} and F_{vw} as well as $F(\delta^+(v))$ are integers by induction. Since both L and γ are also integers, then ϵ is an integer. The modification then adjusts the flow across each edge by an integer so the resulting flow is also integral. Finally, the resulting flow is also acyclic because the flow is acyclic before each update and since we only add edges uw for which there was already a path uv, vw. □

From now on, we assume that F has this uniformity condition among nodes in $W - \{s, t\}$. The following lemma is the first step to finding a good integer D for which $\mathcal{P}(D) \neq \emptyset$. The value D may be fractional, but we deal with that problem later.

Lemma 3.3.10 *There is some value (perhaps fractional) D between k and $\frac{kL}{L - \lfloor \log_2 n \rfloor}$ such that $\mathcal{P}(D)$ is non-empty.*

Proof. Every node $v \in W - \{s, t\}$ has the same amount of flow (say $L - \gamma$) in F passing through it. Let $D = \frac{kL}{L - \gamma}$ and note that $k \leq D \leq \frac{kL}{L - \lfloor \log_2 n \rfloor}$ by Lemma 3.3.9. We now construct a (possibly rational) point z in $\mathcal{P}(D)$ to exhibit that $\mathcal{P}(D) \neq \emptyset$.

This is simple, let $z_{uv} = \frac{F_{uv}}{L-\gamma}$. We have $1 \leq L - \gamma$ because $\gamma \leq \lfloor \log_2 n \rfloor$ and $L = (b+1)\lfloor \log_2 n \rfloor$ where $b \geq 1$. So, $0 \leq z_{uv} \leq F_{uv}$ is satisfied. Since each $v \in W - \{s,t\}$ has $F(\delta^+(v)) = F(\delta^-(v)) = L-\gamma$, then $z(\delta^+(v)) = z(\delta^-(v)) = 1$. Similarly, we see that $z(\delta^+(s)) = z(\delta^-(t)) = D$ and $z(\delta^-(s)) = z(\delta^+(t)) = 0$. □

The main problem is that the value D in the lemma may not be an integer. This is remedied with the following lemma.

Lemma 3.3.11 *If $\mathcal{P}(D) \neq \emptyset$, then $\mathcal{P}(\lfloor D \rfloor) \neq \emptyset$.*

Proof. Suppose D is not already an integer, otherwise we are done. Let z be any point in $\mathcal{P}(D)$. Form an undirected and weighted bipartite graph $H = (W_L \cup W_R, E')$ where both W_L and W_R are disjoint copies of W. For each arc uv, add an edge from $u \in W_L$ to $v \in W_R$ with weight z_{uv}. In fact, we use z_{uv} to denote both the weight of the uv arc in the original directed graph G and the weight of the edge in our new bipartite graph H connecting $u \in W_L$ to $v \in W_R$. By the degree constraints, we have that $z(\delta(v))$ (the total z-value of all edges in E' incident to v) is an integer for every $v \in L \cup W_R$ except for $s \in W_L$ and $t \in W_R$ which have z-degree D.

First, we claim that there is a path from $s \in W_L$ to $t \in W_R$ in H that only uses edges uv with $z_{uv} > 0$. Note that these paths are allowed to take a step from W_R to W_L since H is undirected. Such a step corresponds to following an arc uv in the reverse direction in the original directed graph G.

Suppose, for the sake of contradiction, that there is no path from $s \in W_L$ to $t \in W_R$ using only edges uv with positive weight. Let S be the collection of all nodes in H that can be reached from s using only edges with positive weight; our assumptions means the copy of t in W_R is not in S. We count the weight of the edges with both endpoints in S in two different ways and arrive at a contradiction.

On one hand, since every node $v \in W_R - \{t\}$ has $z(\delta(v)) = 1$ and since $t \notin S$, then

$$z(E(S)) = \sum_{v \in W_R \cap S} z(\delta(v)) = |W_R \cap S|.$$

On the other hand, since every node $v \in W_L - \{s\}$ has $z(\delta(v)) = 1$ and since $s \in S$, then

$$z(E(S)) = \sum_{v \in W_L \cap S} z(\delta(v)) = |(W_L - \{s\}) \cap S| + z(\delta(s)) = |(W_L - \{s\}) \cap S| + D.$$

But $|R \cap S| = |(W_L - \{s\}) \cap S| + D$ contradicts our assumption that D is not an integer. So, it must be that there is a path from $s \in W_L$ to $t \in W_R$ in H using only edges e with $z_e > 0$.

Suppose that such a path followed a sequence of edges $e_1, e_2, \ldots, e_c$. Since H is bipartite and $s \in W_L, t \in W_R$ are on different sides, then c must be odd. Let

$$\sigma = \min \left\{ D - \lfloor D \rfloor, \min_{1 \leq i \leq c : i \text{ odd}} z_{e_i} \right\}$$

be the minimum z-value of the edges that are followed from W_L to W_R when walking along this path (or the difference between D and $\lfloor D \rfloor$ if this is smaller). Update the z values of the edges on this path by:

$$z_{e_i} \leftarrow \begin{cases} z_{e_i} - \sigma & i \text{ odd} \\ z_{e_i} + \sigma & i \text{ even} \end{cases}$$

We now argue that the resulting z-values fit in the polytope $P(D - \sigma)$. First, notice that both $z(\delta(s)), s \in W_L$ and $z(\delta(t)), t \in W_R$, which were originally D, decrease by exactly σ. Any other node v not on this path does not have the z-value of any incident edge changed. Finally, if v is an internal node on this path then the z-value of one incident edge decreases by σ and the z-value of another incident edge increases by σ. Therefore, we have $z(\delta(v)) = 1$ after this update for every node v apart from $s \in W_L$ and $t \in W_R$.

By our choice of σ, we continue to have $z_e \geq 0$ for every edge e. Now, if the path was a single edge st, then no edge had its z-value increased so the bounds $z_e \leq F_e$ continue to hold for every edge e. Otherwise, every edge $e = uv$ on this path has either $z(\delta(u)) = 1$ or $z(\delta(v)) = 1$ so $z_e \leq 1$ must hold after an update. Since the only z_e values that are increased are those in the support of F, then $z_e \leq 1 \leq F_e$.

The above process maps a point from $P(D)$ to a point in $P(D - \sigma)$ when D is not an integer. If σ was chosen to be $D - \lfloor D \rfloor > 0$, then $z(\delta(s)) = z(\delta(t)) = \lfloor D \rfloor$ after this process and we are done. Otherwise, we can repeat the process to map a point from $P(D - \sigma)$ to $P(D - \sigma')$ for some $\sigma' > \sigma$ with $D - \sigma' \geq \lfloor D \rfloor$ and so on. Each such step that does not result in a point in the polytope $P(\lfloor D \rfloor)$ has us remove at least one edge from the support of z. Since no edges are introduced to the support of z, then this process is repeated at most $|W_L| \cdot |W_R| = |W|^2$ times. After finitely many iterations, we arrive at a point in $P(\lfloor D \rfloor)$. □

Lemma 3.3.11 shows that for $D = \lfloor \frac{kL}{L-\gamma} \rfloor$ we have $P(D) \neq \emptyset$. Since D is an integer, then any basic point z in $P(D) \neq \emptyset$ has each z_e value being integer. Find and return such a point to obtain D paths of cost at most the cost of F whose union spans all of W.

The number of paths is bound as:

$$D \leq \frac{kL}{L-\gamma} \leq \frac{k(b+1)\lfloor \log_2 n \rfloor}{(b+1)\lfloor \log_2 n \rfloor - \lfloor \log_2 n \rfloor} = k\left(1 + \frac{1}{b}\right)$$

This proves Theorem 3.3.3.

3.4 Approximating Other Multiple Salesmen Variants

Section 3.3 presented a bicriteria approximation algorithm for the problem of finding k paths from a given node s to a given node t in an asymmetric metric. Here, we consider other variants, mainly on how the start and/or end nodes are specified. For most cases presented, we discuss how the algorithm in Section 3.3 can be used to approximate these variants (after we define what a bicriteria

approximation is for these variants). However, one case that cannot be approximated with these techniques is when we are given pairs of nodes $(s_1, t_1), \ldots, (s_k, t_k)$ and we want to find an $s_i - t_i$ path for each $1 \leq i \leq k$ so that each node is included on at least one such path. We call this case *General k-ATSPP* and discuss this case further.

In symmetric metrics (*i.e.* General k-TSPP), we have a simple constant-factor approximation. In the special case in asymmetric metrics where $s_i = t_i$ for each $1 \leq k \leq n$, the problem is to find cycles where each cycle contains some "root" $r_i := s_i (= t_i)$ so that each node is on one of the cycles. In this case, we show how an $O(\log n)$-approximation follows from a relatively simple modification of the $O(\log n)$-approximation for ATSP reviewed in Section 3.1. Finally, we conclude this section by demonstrating that General k-ATSPP cannot be approximated in polynomial time within any bounded ratio unless P = NP.

3.4.1 Varying the Endpoints in k-ATSPP

We first present a couple of simple transformations between instances of k-ATSPP to show how some variants of k-ATSPP can be modelled as an instance of k-ATSPP when a common start node s and a common end node t are fixed in advance. Then, we can apply the techniques from Section 3.3 to approximate these instances. First, let's assign names to describe the different instances. If a common source s is specified and all paths are required to start at s, then we shall call this case a *single source* case. If no sources are specified and the k paths are allowed to start at any (perhaps different) nodes, then we shall call this case the *no source* case. Finally, if distinct nodes $s_1, \ldots, s_k$ and we require that each s_i be the start of precisely one path, then we shall call this case the *multiple source* case. Note that the case where some of the s_i are the same can easily be reduced to the case where the s_i are different: simply create a separate node s'_i for each $1 \leq i \leq k$ with cost 0 arcs to and from s_i. Let the new instance be defined with distinct start locations s'_i whose underlying metric is the shortest paths metric for the new graph.

Similarly, we can define the terms *single sink*, *no sink*, and *multiple sink* to describe how the endpoints of the paths are described. We can combine these terms to describe both the start and end locations of an instance. For example, the problem considered in Section 3.3 is the single source, single sink case of k-ATSPP. The only real ambiguity is the multiple source, multiple sink case. If nodes $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ are given then there are two variants of k-ATSPP where each s_i is required to be the start of a path and each t_j is required to be the end of a path. If a path that starts at s_i may end at any t_j, then we simply call this variant the multiple source, multiple sink case. However, if we require the path that starts at s_i to end at the respective node t_i, then we call this problem *General k-ATSPP*.

Theorem 3.4.1 *Suppose there is an $(\alpha(n), \beta(n))$-bicriteria approximation algorithm for single source, single sink instances of k-ATSPP. Then there is an $(\alpha(n+1), \beta(n+1))$-bicriteria approximation algorithm for single source, no sink instances of k-ATSPP.*

Proof. Let $G = (V, A)$ be an instance of k-ATSPP with distances $d_{uv}, u, v \in V$ and a specified start node s. Also say that OPT is the cost of the optimum k-ATSPP solution in G and that paths $P_1^*, \ldots, P_k^*$ are the paths used in such an optimum solution. Create a new vertex, say t, and let $M := \max_{u,v \in V} d_{uv}$. Consider the asymmetric metric graph $G' = (V \cup \{t\}, A')$ where the distances d'_{uv} are defined as:

$$d'_{uv} = \begin{cases} d_{uv} & \text{if } u, v \in V \\ 0 & \text{if } v = t \\ M & \text{if } u = t \end{cases}$$

First, we verify that the distances d'_{uv} satisfy the directed triangle inequality. Let $u, v, w \in V'$ with $u \neq v, v \neq w$.

- if $u, v, w \in V$ then $d'_{uw} = d_{uw} \leq d_{uv} + d_{vw} = d'_{uv} + d'_{vw}$
- if $u = t$ and $v, w \in V$ then $d'_{uw} = M \leq M + d_{vw} = d'_{uv} + d'_{vw}$
- if $v = t$ and $u, w \in V$ then $d'_{uw} = d_{uw} \leq 0 + M = d'_{uv} + d'_{vw}$
- if $w = t$ and $u, v \in V$ then $d'_{uw} = 0 \leq d'_{uv} + d'_{vw}$
- finally, if $u = w = t$ and $v \in V$ then $d'_{uw} = 0$ as well

Now, view G' as a instance of k-ATSPP with start node $s \in V$ and end node t being specified. Let $P_1, \ldots, P_{k'}$ be the paths found by the bicriteria approximation with $k \leq k' \leq \beta k$. Note that G' has a solution of cost OPT which is obtained by appending t to the end of each of the paths $P_i^*, 1 \leq i \leq k$ used in the optimum solution for G. So, the total cost of the paths $P_1, \ldots, P_{k'}$ is at most αOPT. By shortcutting, we may assume that no path $P_i, 1 \leq i \leq k'$ contains t as an internal node. Deleting t from the end of each of these paths, we obtain at most βk paths $P'_1, \ldots, P'_{k'}$ that start at s and visit every node in $V = V' - \{t\}$ whose total cost is at most αOPT. □

Corollary 3.4.2 *For any integer $b \geq 1$, there is an $(O(b \log n), (1 + \frac{1}{b})k)$-bicriteria approximation for single source, no sink instances of k-ATSPP.*

Proof. It is immediate from Theorems 3.3.3 and 3.4.1 and the fact that $(b+1)\lfloor \log_2(n+1) \rfloor = O(b \log n)$. □

Now, consider the variant where a common start node s is given and there are k distinct end nodes $t_1, \ldots, t_k$. The goal is to find an $s - t_i$ path for each $1 \leq i \leq k$ so that every node is on at least one such path while minimizing the total cost of these paths. A clarification needs to be made regarding bicriteria approximation algorithms for such instances. We say an (α, β)-bicriteria approximation algorithm is one that finds between k and βk paths where each starts at s and ends at some t_i whose total cost is at most αOPT. Furthermore, for each $1 \leq i \leq k$, there must be at least one $s - t_i$ path among the at most βk paths.

Theorem 3.4.3 *Suppose there is an $(\alpha(n), \beta(n))$-bicriteria approximation algorithm for single source, single sink instances of k-ATSPP. Then there is an $(\alpha(n+k+1), \beta(n+k+1))$-approximation for single source, multiple sink instances of k-ATSPP.*

Proof. Say that $G = (V, A)$ is an instance of k-ATSPP with distances d_{uv} where a common start node s and varying end nodes $t_1, \ldots, t_k$ are specified. Also say that OPT is the cost of the optimum k-ATSPP solution in G and that paths $P_1^*, \ldots, P_k^*$ are the paths used in such an optimum solution where P_i^* ends at t_i. The reduction is similar to the one in Theorem 3.4.1 except we have to deal with one potential issue. If we were to simply add a node t and a cost 0 arc from each t_i to t, then a path may visit t_i before passing through a different t_j to reach t. This would allow multiple paths to reach t through some t_j while having no path reach t through some t_i. We deal with this by adding an extra node between each t_i and t that can only be reached through t_i to force the paths to reach t through distinct nodes t_i.

Create $k + 1$ new vertices $t, r_1, \ldots, r_k$ and let $M := \max_{u,v \in V} d_{uv}$. We first create a (not complete) directed graph G'' before obtaining our final metric graph. Let $G'' = (V', A'')$ where $V' = V \cup \{r_1, \ldots, r_k, t\}$ and $A'' = A \cup \{t_i r_i, r_i t : 1 \leq i \leq k\} \cup \{tv : v \in V\}$. Define costs d''_{uv} as

$$d''_{uv} = \begin{cases} d_{uv} & \text{if } u, v \in V \\ 0 & \text{if } uv = t_i r_i \text{ or } uv = r_i t \text{ for some } 1 \leq i \leq k \\ 1 + \alpha(n+k)M & \text{if } u = t, v \in V \end{cases}$$

Notice that for any $u, v \in V'$ there is a path from u to v in G'' and that the shortest path from t to any other node has cost at least $1 + \alpha(n + k)M$ since that is the cost of any outgoing arc from t. Finally, let $G' = (V', A')$ be the asymmetric metric graph whose distances are obtained from the shortest path distances in G''. View G' as an instance of k-ATSPP where both the start node $s \in V$ and end node t are fixed. Let $P_1, \ldots, P_{k'}$ be paths obtained from running the (α, β)-bicriteria approximation on G' where $k \leq k' \leq \beta k$.

There is a solution of cost at OPT in G' that is obtained by adding r_i and then t to the end of each path P_i^* in the optimum solution for the k-ATSPP instance on G. Since P_i^* ends at t_i, then the total cost of the steps added to the paths is 0. So, the total cost of the paths $P_1, \ldots, P_{k'}$ is at most αOPT. In particular, this means no path P_i contains t as an internal node since the distance from t to any other node exceeds $\alpha(n + k)M \geq \alpha OPT$ (notice that the optimum solution for G contains at most $n + k$ arcs, each of cost at most M). For the same reason, no path contains any of the new r_i nodes except, perhaps, as the second last node.

We can now obtain paths $P'_1, \ldots, P'_{k'}$ in G from the paths $P_1, \ldots, P_{k'}$. For each i, we have that P_i stays entirely within V except for either the last node, which is t, or the last two nodes which visit some r_j before ending at t. In the first case, say v is the second last node on path P_i. Then $d'_{vt} = \min_{1 \leq j \leq k} d_{vt_j}$ since the shortest path from v to t in G'' first visits the node of the form t_j that is nearest to v. For such a node t_j, we form a path P'_i in G that follows P_i to v, then finishes at t_j. In the second case where P_i visits some r_j and then t, we let the path P'_i in G follow P until

the node just prior to r_j, then take one final step to t_j if the node appearing before r_j on P_i is not already t_j. In either case, the cost of P_i' is the same as the cost of P_i so the total cost of the paths P_i' is at most αOPT. Finally, since each node r_j was visited at least once, then for each $1 \leq j \leq k$ there is some path P_i' that ends at t_j. □

Corollary 3.4.4 *There is an $(O(b \log n), (1 + \frac{1}{b})k)$-bicriteria approximation for single source, multiple sink instances of k-ATSPP where. t_i are specified.*

Proof. This follows from Theorems 3.3.3 and 3.4.3 and the fact that $(b+1)\lfloor \log_2(n+k+1) \rfloor = O(b \log n)$ since $k \leq n$. □

If the start points are subject to the same variations (*i.e.* no source or multiple source instances with a single sink), then the analogs of Theorems 3.4.1 and 3.4.3 hold by minor adjustments to the proofs to create a common start locations s. In fact, any combination of ways to specify the start and endpoints can be approximated using similar means (*e.g.* no source and multiple sinks would have an approximation ratio of $\alpha(n+k+2)$).

Again, we have to clarify what happens in the multiple source, multiple sink case with sources $S = \{s_1, \ldots, s_k\}$ and sinks $T = \{t_1, \ldots, t_k\}$. In such a case, the reduction to the case with a common start s and a common node t and using a bicriteria approximation for this case guarantees only that each path starts at some s_i and ends at some t_j but does not guarantee $i = j$. In particular, if the approximation algorithm guarantees that only k paths are used (so it is a true approximation, not a bicriteria approximation), then the only thing we can say about the structure is that the association of a start node s_i to the end node t_j of the path starting at s_i establishes a bijection between S and T. As mentioned before, this is interesting in cases where the "salesmen" that travel from S to T are identical so it does not matter which one is received by a node in T.

Finally, we come to the case where we require the path that starts at s_i to end at t_i which we call *General k-ATSPP*. A bicriteria approximation that uses at most βk paths s all paths start and end at some input pair (s_i, t_i) (*i.e.* no paths can start at some s_i and end at some t_j for $i \neq j$) and has at least one $s_i - t_i$ path for each $1 \leq i \leq k$. Another way to think of this is that each (s_i, t_i) pair has at least path and the average number of times an (s_i, t_i)-path is used is at most β. Before presenting the inapproximability of this general problem, we mention how certain interesting cases of this problem can be approximated well.

3.4.2 A Constant Factor Approximation for General k-TSPP in Symmetric Metrics

Recall the General k-TSPP problem is the following. We are given a set of k ordered pairs of nodes $(s_1, t_1), \ldots, (s_k, t_k)$ in a symmetric metric graph $G = (V, E)$ with distances d_{uv}. The goal is to find an $s_i - t_i$ path for each pair whose union spans all nodes. Here, we see that the problem can be approximated within a constant factor if the metric is symmetric (*i.e.* $d_{uv} = d_{vu}$ for all $u, v \in V$).

We may assume the $2k$ nodes $s_1, t_1, \ldots, s_k, t_k$ are distinct by creating multiple copies of each start and/or end location in the following way. For each v that is some start node s_i or some end node t_j, let n_v be the number of times that v appears among $s_1, t_1, \ldots, s_k, t_k$. Replace v with n_v different nodes $v_1, \ldots, v_{n_v}$ where we define $d(v_i, u) := d(v, u)$ for $1 \leq i \leq n_v$ and $d(v_i, v_j) = 0$ for $1 \leq i, j \leq n_v$. It is trivial to verify that these new distances also satisfy the triangle inequality. Let $R = \{s_1, t_1, \ldots, s_k, t_k\}$ be the collection of $2k$ nodes that are the start or end nodes from some pair.

Definition 3.4.5 *An R-rooted spanning forest F is an acyclic collection of edges F such that every node $v \in V$ is connected to some node in R by edges in F.*

The cheapest R-rooted spanning forest is a lower-bound on the optimum solution to k-TSPP. To see this, suppose P is the set of edges used in the optimum k-TSPP solution. Since the nodes $s_1, t_1, \ldots, s_k, t_k$ are all distinct and the distances are metric, then no node $v \in V$ is visited by more than one path (which also implies each edge is used at most once). We can then obtain an R-rooted spanning forest by deleting one edge from each $s_i - t_i$ path in P, which has cost no less than the cheapest R-rooted spanning forest.

Let $I \subseteq \mathcal{P}(E)$ be the collection of subsets of E such that $F \in I$ if and only if the graph $G_F = (V, F)$ is acyclic and no two nodes in R are in the same connected component of G_F. Then the maximal subsets in I are preciesly the R-rooted spanning forests of G. The following observation on the structure of I allows us to compute the cheapest R-rooted spanning forest in polynomial time.

Theorem 3.4.6 (e.g. **Cerdeira [22])** *The pair $\mathcal{M} = (E, I)$ where I is defined as in the previous paragraph is a matroid.*

Given this, we use the following algorithm to compute an approximate k-TSPP solution.

Algorithm 6 A 3-approximation for General k-TSPP in symmetric metrics

```
1: Let T_1 be a minimum cost R-rooted spanning forest
2: Let T_2 be the multi-set of edges obtained by doubling each edge in T_1
3: T_3 ← T_2 ∪_{i=1}^{k} {s_i t_i}
4: P ← ∅
5: for each connected component F of T_3 do
6:     Let (s_i, t_i) be the unique input pair contained in F
7:     Compute an Eulerian walk P_F from s_i to t_i in F
8:     Shortcut over repeated nodes in P_F to obtain an s_i − t_i path P'_F
9:     P ← P ∪ {P'_F}
10: end for
11: return P
```

Theorem 3.4.7 *Algorithm 6 returns a collection of $s_i - t_i$ paths whose union spans all nodes. Furthermore, the cost of these paths is at most $3 \cdot OPT$.*

Proof. By assumption that all $s_1, t_1, \ldots, s_k, t_k$ are distinct, then each connected component of T_1 (and, thus, T_2) has precisely one node in S. Then each connected component in T_3 contains precisely two nodes in S and these correspond to the endpoints of some input pair (s_i, t_i). Furthermore, since T_1 is a subset of T_3, then in T_3 every node $v \in V$ is connected to both nodes in some input pair (s_i, t_i).

Consider a connected component F of T_3 containing a pair, say, (s_i, t_i). Since the degree of every node in T_2 is even, then the degree of every node in $F - \{s_i, t_i\}$ is even and the degrees of s_i and t_i are odd. Thus, there is an Eulerian walk from s_i to t_i in F.

We argued that the cost of T is at most OPT just before Theorem 3.4.6. It follows that the cost of T_2 is at most $2 \cdot OPT$. Since any feasible solution consists of paths from s_i to t_i for each $1 \leq i \leq k$ and since following the single edge $s_i t_i$ is the shortest $s_i - t_i$ path by the triangle inequality, then the cost of $\cup_{i=1}^{k} \{s_i t_i\}$ is also at most OPT. That is, the cost of T_3 is at most $3 \cdot OPT$. Now, P_F is an Eulerian walk of F meaning its cost is exactly the cost of the edges of F. Shortcutting does not increase the overall cost, so the cost of P'_F is at most the cost of F. Thus, the final set of paths returned by Algorithm 6 costs no more than $3 \cdot OPT$. □

3.4.3 A Logarithmic Approximation for General k-ATSPP with $s_i = t_i$.

Now consider the variant of General k-ATSPP (back in asymmetric metrics) where $s_i = t_i$ for every $1 \leq i \leq k$. A solution looks like a collection of cycles where each cycle contains a root and each node lies on one such cycle. Note that there may be "singleton cycles" consisting of a single root node r_i whose corresponding salesman does not visit any other nodes. From now on, we view the problem as the following. Given an asymmetric metric graph $G = (V, A)$ with distances d_{uv} and a subset of nodes $R = \{r_1, \ldots, r_k\}$, we want to find a collection of cycles of minimum total cost where each cycle contains one node in R and every node in $V - R$ is on one such cycle.

The classic $O(\log n)$-approximation by Freize *et al.* [42] for ATSP can be modified in a simple way to approximate this variant. Define a *R-rooted cycle cover* as a collection of cycles where each node in $V - R$ is on one such cycle but where a node in R is not necessarily on any of these cycles (which may be thought of as a "trivial cycle" containing only one node). For any subset $W \subseteq V$ including R, the cheapest R-rooted cycle cover on the graph induced by W is a lower-bound for OPT since we can obtain a feasible R-rooted cycle cover on the graph induced by W by simply shortcutting past nodes in $V - W$ in the optimum General k-ATSPP solution for G.

Lemma 3.4.8 *The cheapest R-rooted cycle cover of G can be computed in polynomial time.*

Proof. Let $A' = A \cup \{l_1, \ldots, l_k\}$ where l_i is a self-loop at r_i with cost 0 (*i.e.* an arc from r_i to itself). The R-rooted cycle covers are precisely the common bases between the two partition matroids over A' that bound, respectively, the indegree and outdegree of a node by 1. □

The algorithm then proceeds in the natural way. Find a minimum cost R-rooted cycle cover. If a cycle C has more than one root r_i, then iteratively shortcut C around some roots (but do not discard the roots) until exactly one root is on C. By the triangle inequality, the cost does not increase which, by optimality of the R-rooted cycle cover, means the resulting R-rooted cycle cover is also a minimum cost R-rooted cycle cover. Now, for each cycle with no root discard all but one node (choosing this node arbitrarily). For a cycle containing a root r_i, discard all nodes but r_i from this cycle. Repeat until only R remains, add the cycles found in previous iterations, and shortcut.

Algorithm 7 An $O(\log(|V-R|)$-approximation for General k-ATSPP when $s_i = t_i$ for all i

```
Let D ← ∅
Let W ← V
while W ≠ R do
    Let D' be a minimum cost R-rooted cycle cover of W
    Let C_1, C_2, ..., C_l be the cycles of D'
    for each cycle C_i of D' do
        if C_i contains more than one root then
            Shortcut past all but one root in C_i
        end if
        if C_i contains one root r_j then
            Let v_{C_i} ← r_j
        else
            Let v_{C_i} be any node on C_i
        end if
        W ← {v_{C_1}, v_{C_2}, ..., v_{C_l}}
        D ← D + D'
    end for
end while
for each i from 1 to k do
    Find an Eulerian circuit X_i in the component containint r_i
    Shortcut X_i past repeated nodes to obtain a cycle Y_i
end for
return {Y_i}_{i=1}^k
```

The proof of correctness is near-identical to the proof of correctness for Algorithm 3. The main difference is that we must deal with multiple components at the end of the main loop. All cycles found have length at least 2, all but at most 1 node in each cycle is discarded in each step, and all nodes in $W - R$ appear on at least one such cycle so the number of iterations is bounded by $\lfloor \log_2 |V - R| \rfloor + 1$. The extra $+1$ is because it takes at most $\log_2 |V - R|$ iterations to reduce the set of nodes not in R to a single node. One more iteration then guarantees this node is connected to some root. So, we have just argued the following.

Theorem 3.4.9 *There is an $O(\log |V| - k)$-approximation for instances of General k-ATSPP where $s_i = t_i$ for each $1 \leq i \leq k$.*

While this is an $O(\log n)$-approximation where $n = |V|$, for large values of k (*i.e.* $l = n - 2^{o(\log n)}$) it is actually a $o(\log n)$-approximation.

3.4.4 Inapproximability of General k-ATSPP

We recall the General k-ATSPP problem. We have a collection of start nodes $s_1, \ldots, s_k$ and a collection of end nodes $t_1, \ldots, t_k$ in an asymmetric metric $G = (V, A)$ with non-negative distances $d_{uv}, uv \in A$ satisfying the directed triangle inequality. We must find a path P_i for each $1 \leq i \leq k$ where P_i starts at s_i and ends at t_i and such that every node in V lies on some P_i. The objective is to minimize the total cost of all paths P_i.

The main result of this section is proving that General k-ATSPP cannot be approximated within any multiplicative factor $f(n)$ where $f(n)$ is a polynomial-time computable function (*e.g.* 2^n or $n!$) unless P = NP. The reduction is from the following problem.

Definition 3.4.10 *In the* tripartite triangle packing problem*, we are given a tripartite graph $G = (U \cup V \cup W, E)$ with $|U| = |V| = |W| = n$ where no edge in E has both endpoints in a common partition U, V, or W. A triangle is a subset of 3 nodes for which any two are adjacent in G. The problem is to determine if it is possible to find n vertex-disjoint triangles in G.*

The following can be found in [45].

Theorem 3.4.11 *The triangle packing problem in tripartite graphs is NP-complete.*

We have the following lower-bound on the approximability of General k-ATSPP. We recall that a *polynomial-time computable function* is a function $f : \mathbb{Z} \to \mathbb{Q}$ such that $f(n)$ can be computed in time $O(\text{polylog}(n))$ (*i.e.* polynomial time in the number of bits used to represent the input).

Theorem 3.4.12 *General k-ATSPP cannot be approximated better than any polynomial-time computable ratio $f(n)$ unless P = NP.*

Proof. Let $G = (U \cup V \cup W, E)$ be an instance of triangle packing in a tripartite graph with $|U| = |V| = |W| = n$. Create a directed graph H with four layers of nodes X_1, X_2, X_3, X_4 where X_1 and X_4 are disjoint copies of U, X_2 is a copy of V, and X_3 is a copy of W. For every uv edge in G with $u \in U, v \in V$, add an arc from $u \in X_1$ to $v \in X_2$ in H. For every vw edge in G with $v \in V, w \in W$, add an arc from $v \in X_2$ to $w \in X_3$ in G. Finally, for every uw edge in G with $u \in U, w \in W$, add an arc from $w \in X_3$ to $u \in X_4$. Since every arc is oriented from X_i to X_{i+1} for some $i = 1, 2, 3$ then the graph is acyclic. All of these arcs should have distance 1. Finally, for every $u \in X_4$ and every $u' \in X_1$, add an arc directed from u to u' of cost $3nf(4n)$. Next, let H' denote the asymmetric metric on the same nodes as H where the distance between nodes in H' is their distance in H. For each $u \in U$, add a salesman that starts at the copy of u in X_1 and ends at the copy of u in X_4. So, there are n salesmen in total. This instance of General k-ATSPP can be computed in polynomial time because $f(n)$ is polynomial-time computable. An example of this reduction is illustrated in figure 3.5.

The claim is that if there is a triangle packing including n triangles, then there is a k-ATSPP solution with cost $3n$. Otherwise, any k-ATSPP solution must use an edge with cost $\geq 3nf(4n)$,

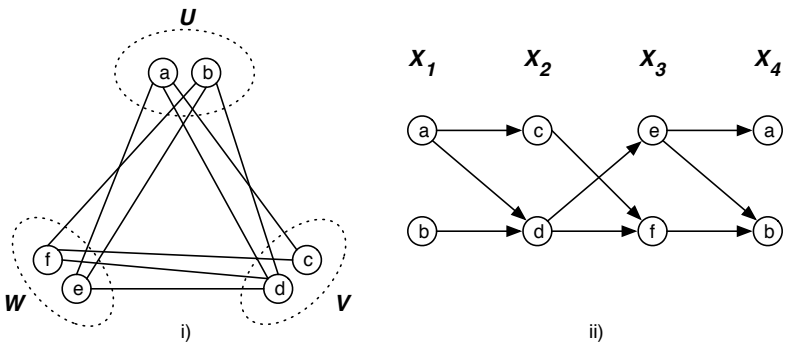

Figure 3.5: i) An instance of tripartite triangle packing with $n = 2$. ii) The graph H with all cost 1 arcs drawn. The "back arcs" of cost $3nf(4n)$ are not pictured. The final metric H' (not pictured) is shortest paths metric form H. The path b, d, e, b corresponds (in the sense of the proof) to triangle $\{b, d, e\}$ in the first graph. Also, one can see that the graph in image i) does not have a triangle packing nor does the graph in image ii) have a General k-ATSPP solution using only cost 1 arcs.

so the gap between "yes" and "no" instances is at least $f(4n)$. Suppose $T = \{(u_i, v_i, w_i)\}_{i=1}^{n}$ is a collection of n vertex-disjoint triangles with each $u_i \in U, v_i \in V$ and $w_i \in W$. For each such triangle (u_i, v_i, w_i), we have the salesman starting at $u_i \in X_1$ to travel first to $v_i \in X_2$, then to $w_i \in X_3$, and finally to $u_i \in X_4$. Every node in H' is visited since every node is included in some triangle in T. By construction, every step taken by a salesman traverses an edge of cost 1. Each of the n salesmen then moves a total distance of 3 so the total cost is $3n$.

Conversely, suppose there is a k-ATSPP solution that avoids using any edge of cost $3nf(4n)$. Then each of the edges followed by the salesmen must be in increasing order of the level X_i. Since there are only n salesmen and since each salesman can visit only two nodes in addition to their endpoints, then each salesman visits every layer X_i and each node is visited by preciseley one salesman. Finally, by construction the cost of an edge from layer X_i to layer X_{i+1} is either 1 or at least $3nf(4n)$. Since we assumed that no edges of cost $3nf(4n)$ were used, then every salesman must use a weight 1 edge. Thus, the nodes visited by each salesman correspond to a triangle in G and these triangles partition the nodes of G. Since the graph H' has $4n$ nodes, then first statement of the theorem holds. □

It may still be possible to devise a bicriteria approximation algorithm for General k-ATSPP with a bounded approximation ratio α using at most βk paths. We have simply shown that such an algorithm would necessarily have $\beta k \geq k + 1$ if P $\neq$ NP.

The instances of General k-ATSPP produced in the previous reduction have $k = \frac{n}{4}$. We can slightly modify the reduction in Theorem 3.4.12 to prove a similar hardness for smaller values of k relative to n.

Theorem 3.4.13 *For any constant $\epsilon > 0$ and any polynomial-time computable function $f(\cdot)$, instances of General k-ATSPP with $k \geq n^{\epsilon}$ cannot be approximated within $f(n)$ unless P = NP.*

Proof. Form the same auxiliary graph H as in the reduction in Theorem 3.4.12. Then, append a directed path of length at least $n^{\frac{1}{\epsilon}} - 4n$ to the copy of u_1 in X_4 and say this path ends at node u_1'. Call this new graph H'' and note that total number of nodes N in H'' is at least $n^{\frac{1}{\epsilon}}$. We also set the weight of each backward arc uv with $u \in X_4$ and $v \in X_1$ to be $(N-n) \cdot f(N)$. The metric in the General k-ATSPP instance is then the shortest-path metric completion of H. The salesmen are similarly defined: for each $2 \leq i \leq n$ we define a salesman that starts at the copy of u_i in X_1 and ends at the copy of u_i in X_4. The only difference is that the first salesman starts at the copy of u_1 in X_1 and ends at the new node u_1' at the end of the new path we appended to the copy of u_1 in X_4.

If there is a triangle packing, then follow the same paths in the k-ATSPP solution as in the proof of Theorem 3.4.12 except the first salesman proceeds to u_1' along the newly-appended path after reaching u_1. The total cost of this solution is $N - n$. Conversely, any k-ATSPP solution that avoids using an arc of cost at least $(N-n) \cdot f(N)$ corresponds naturally to a triangle packing. We note that the first salesman must visit u_1 before u_1' if no arcs of cost at least $(N-n) \cdot f(N)$ are followed, so their subpath from $u_1 \in X_1$ to $u_1 \in X_4$ also corresponds to a triangle.

The gap between "yes" and "no" instances is then $f(N)$. Since $f(N)$ can be computed in polynomial time in n, then the reduction takes polynomial time. □

One may ask about even smaller values for k. Currently, it is not known if the canonical NP-complete problem 3SAT (instances of SAT with at most three variables per clause) has a sub-exponential time algorithm (*i.e.* $2^{o(n)}$) where n is the number of variables in the SAT instance and m is the number of clauses. Note that a $2^{O(n)}$ algorithm for 3SAT is trivial since we can try all 2^n possible truth assignments and, in time that is polynomial in n and m, check each such truth assignment to see if it satisfies the instance. The contribution of m to the running time is suppressed since $m = O(n^3)$ in an instance of 3SAT. We can show that approximating General k-ATSPP on N nodes when k is polylogarithmic in N would imply a sub-exponential time algorithm for 3SAT.

Theorem 3.4.14 *There is a constant $d > 0$ such that for any polynomial-time computable function $f(\cdot)$ the following holds. Suppose there is a polynomial-time approximation algorithm for instances of k-ATSPP on N nodes with $k \geq (\log_2 N)^d$ with ratio better than $f(N)$. Then there is a sub-exponential time algorithm for 3SAT.*

Proof. Consider the reduction from 3SAT to tripartite triangle packing from [45]. For some constant $c > 0$, the running time is $O(n^c)$ where n is the number of variables in the original SAT instance. By slightly increasing c we may assume that there is a constant n_0 such that for all $n \geq n_0$, instances of SAT on n variables are reduced to instances of tripartite triangle packing in time at most n^c. If we denote the size of a partition in the tripartite triangle packing instance by k, then $k \leq n^c$ since the size of the resulting instance can be no larger than the running time of the reduction.

Now consider the following reduction from 3SAT to General k-ATSPP. Reduce an instance Φ of 3SAT first to tripartite triangle packing as in [45] (with each partition having size $k \leq n^c$) and then

to k-ATSPP using the reduction from Theorem 3.4.13, except the path length is $\lfloor 2^{k^{\frac{1}{c(c+1)}}} \rfloor - 4k$. The length of the backward arcs uv with $u \in X_4$ and $v \in X_1$ is still $(N-n) \cdot f(N)$ where N is the number of nodes in this modified version of H with the sub-exponentially long path. The resulting metric then has $N \leq 2^{k^{\frac{1}{c(c+1)}}} \leq 2^{n^{\frac{1}{c+1}}}$ nodes. Since the number of bits used to represent N is polynomial in n, then $f(N)$ can be computed in time that is polynomial in n. Thus, the entire reduction takes time that is polynomial in $2^{n^{\frac{1}{c+1}}}$. As in Theorem 3.4.13, the gap between "yes" and "no" instances is at least $f(N)$. Finally, the number of paths in the instance k is at least $(\log_2 N)^{c(c+1)}$. We let the constant d in the statement of the theorem be $c(c+1)$.

Now, suppose there is a polynomial-time approximation algorithm for General k-ATSPP when $k \geq (\log_2 N)^d$ with ratio better than $f(N)$. We show how to use this to get a sub-exponential time algorithm for 3SAT. Let Φ be a 3SAT instance on n nodes. If $n \leq n_0$, we solve Φ by brute force over all of the possible truth assignments. As $n \leq n_0$, there are a constant number of truth assignments that must be checked. Otherwise, run the reduction outlined in the previous two paragraphs and then run the approximation algorithm for General k-ATSPP on this instance. The running time of the reduction to tripartite triangle packing found in [45] is polynomial in n, the reduction to General k-ATSPP is polynomial in $2^{n^{\frac{1}{c+1}}}$, and the running time of the algorithm is polynomial in $N \leq 2^{n^{\frac{1}{c+1}}}$ and $\log f(N)$. Since $\log f(N)$ is bound by a polynomial in n, then the total running time of this sequence of steps is $O((2^{n^{\frac{1}{c+1}}})^b) = O(2^{bn^{\frac{1}{c+1}}})$ for some constant b. That is, the total running time is $2^{o(n)}$ since $c > 0$. $\square$

Chapter 4

Minimum Latency in Asymmetric Metrics

Minimum Latency problems are similar to Traveling Salesmen problems except we want to minimize the average time a node waits to be reached, rather than minimize the total travel cost. Minimum Latency problems are also referred to as Traveling Repairman problems because they model the following situation. Imagine you are a repairman and you have a list of clients that need your services. Since these clients are likely frustrated that they need repairs, it is more important to your business that you minimize the average time a client waits to be served rather than minimize your total cost of travel.

Recall the formal description of the Minimum Latency problem in asymmetric metrics. We are given an asymmetric metric $G = (V, A)$ with distances d on arcs in A. Furthermore, a start node $s \in V$ is specified. The goal is to find a Hamiltonian path P in G starting at s to minimize the following objective. Suppose there are n nodes (including s) and the distance from s to the k'th node on this path is D_k (with $D_1 = 0$ since the first node visited is s). Then the objective is to minimize the average waiting time $\frac{1}{n}\sum_{k=1}^{n} D_k$. Note that we do not actually care about the distance from the last node in P to s since the time it takes the repairman to travel home does not affect client satisfaction. We often say that the distance from s to v along P is the *latency* of node v in path P. Also, notice that the average latency of nodes on a path P is exactly a factor n from the total latency of all nodes. For the sake of simplicity, from now on we suppose that the cost of a solution is measured by the total latency of all nodes rather than their average latency. Note that an α-approximation for minimizing the total latency is also an α-approximation for minimizing the average latency.

In general, the minimum latency path P may look very different from the optimum Hamiltonian path starting from s (*i.e.* a TSP Path solution starting at s) even if the metric is symmetric (*i.e.* $d_{uv} = d_{vu}$ for all $u, v \in V$). Consider the following example from [18]. The metric M is simply the subset of $n + 1$ integer points (with n even) $\{0, 1, -2, 4, -8, \ldots, -2^{n-1}\} = \{0\} \cup \{(-2)^i : i = 0, 1, \ldots, n-1\}$ on the real line $\mathbb{R}$ with distances between points x, y simply being $|x - y|$. The

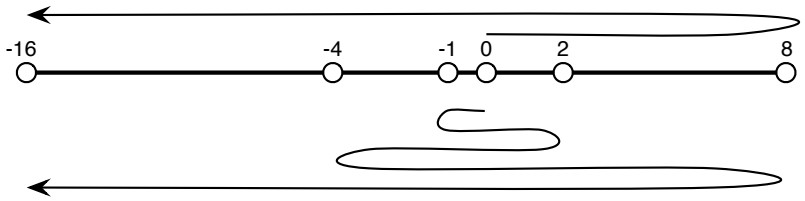

Figure 4.1: An instance of the Minimum Latency problem on a subset of points on the real line. The optimum TSP Path solution that starts at s is picured above the line and the optimum latency solution is pictured below. The latency of the path above is an $\Omega(n)$ factor larger than the latency of the bottom path.

starting point is $s = 0$. Then the optimum TSP Path solution travels from 0 to 2^{n-2} and then to -2^{n-1}. Notice that this path passes by all other client locations and that the total distance of this path is 2^n. Any location of the form -2^{2i+1} has latency at least 2^{n-1} since this is the time it takes the path to travel from 0 to location 2^{n-2} and back to 0 again. There are $n/2$ such points so the average latency is then $\Omega(2^n)$.

However, consider the solution that visits locations in the order $1, -2, 4, -8, 16, \ldots, -2^{n-1}$. Letting $\ell(i)$ denote the latency of the i'th node in this list, we have $\ell(1) = 1$ and $\ell(i) = \ell(i-1) + 2^{i-2} + 2^{i-1} = \ell(i-1) + 3 \cdot 2^{i-2}$ for $i > 1$. Inductively, we then have $\ell(i) = 3 \cdot 2^{i-1} - 2$. Summing over all i, the total latency is then $O(2^n)$ so the average latency is $O(2^n/n)$.

This is an $\Omega(n)$-factor smaller than the average waiting time when following the optimum TSP Path solution described above. See Figure 4.1 for an illustration of these two solutions. In [18], it is also noted that some instances of the Minimum Latency problem over points in $\mathbb{R}^2$ (with distances between points being Euclidean distance) have optimum solutions that use crossing edges. On the other hand, there are no optimum TSP Path solutions that use crossing edges for any instance of TSPP over points in $\mathbb{R}^2$ (*e.g.*, [63]).

The main result of this chapter is an approximation algorithm for the Minimum Latency problem in asymmetric metrics. Our algorithm also bounds the integrality gap of a particular LP relaxation that we introduce in sec 4.3.

Theorem 4.0.15 *There is an $O(\log n)$-approximation algorithm for the Minimum Latency problem in asymmetric metrics. Furthermore, the integrality gap of linear program 4.4 is bounded by $O(\log n)$.*

In some sense, the linear programming relaxation we consider has variables for each node v_i that look like variables in LP 3.3 for ATSPP with start node s and end node v_i. Variables for different i are then related through some extra "ordering" constraints that attempt to order the locations. Along the way, we require Theorem 3.3.3 and another modification of Theorem 3.2.6. Specifically, it is not be sufficient to simply use any $O(k \log n)$-approximation for k-ATSPP, we need the fact that the LP presented for ATSPP has an $O(k \log n)$ bound on the integrality gap.

A generalization of the Minimum Latency problem also considers repair times. Say each node v has a repair time r_v meaning the repairman must take r_v time to service node v before moving to the next node. We say that the latency of a node is the time it takes to reach the node plus the time it takes to complete the repair. For simplicity, we say that the start node s does not have any repair time. The special case $r_v = 0$ for all $v \in V$ is the original Minimum Latency problem. It is quite easy to incorporate repair times in asymmetric metrics.

Theorem 4.0.16 *If there is an $\alpha(n)$-approximation algorithm for the Minimum Latency problem in asymmetric metrics, then there is also an $\alpha(n)$-approximation algorithm for the Minimum Latency problem in asymmetric metrics with repair times.*

Proof. Suppose $G = (V, A)$ is an instance of the Minimum Latency problem in asymmetric metrics with start node $s \in V$, distances d, and repair times r_v. Form a new (nonmetric) graph $G' = (V', A')$ where V' consists of V plus a copy v' of each node $v \in V - \{s\}$. For each new copy v' of some node v, add an arc from v to v' with cost r_v (the repair time of node v) and add an arc from v' to v with cost 0. Finally, the asymmetric metric H is defined as follows. The nodes of H are $\{s\} \cup (V' - V)$ (the start s plus the copy v' of each $v \in V - \{s\}$) and an arc uv of H have distance equal to the length of the shortest path from u to v in the graph G'. This asymmetric metric H is the instance of the Minimum Latency problem *without* repair times. Note that H and G have the same number of nodes.

Consider a Hamiltonian path (say P) $s, v_2, v_3, \ldots, v_n$ in G. We claim that the latency of the Hamiltonian path (say P') $s, v'_2, v'_3, \ldots, v'_n$ in H (without repair times) is equal to the latency of the path P in G (with repair times). In P, the latency of node $v_i, i > 1$ is equal to the length of the path from s to v_i plus the repair times of nodes $v_j, 1 < j \leq i$. In G', the shortest path from s to v'_2 is equal to d_{s,v_2} plus the repair time of v_2. Similarly, the length of the shortest path from v'_j to v'_{j+1} is $d_{v_j v_{j+1}}$ plus the repair time of v_{j+1}. So, we see that the length of the path from s to v'_i in P' is equal the length of the path from s to v_i in P plus the repair times of all $v_j, 1 < j \leq i$.

Similarly, given a Hamiltonian path (say P') $s, v'_2, v'_3, \ldots, v'_n$ in H we can consider the Hamiltonian path (say P) $s, v_2, v_3, \ldots, v_n$. Essentially the same arguments show that the latency of P' (without repair times) is the same as the latency of P (with repair times). Since solutions to the Minimum Latency problems in G and H correspond naturally and have the same cost and since both G and H have the same number of nodes, then an $\alpha(n)$-approximation algorithm for the Minimum Latency problem without repair times can be used to approximate the Minimum Latency problem with repair times. □

In symmetric metrics, the first constant factor approximation for the Minimum Latency problem, by Blum *et al.* [18] guaranteed an approximation ratio of 72. Goemans and Kleinberg [47] improved this to 21.55, followed by an improvement to 7.18 by Archer *et al.* [5]. Currently, the best approximation ratio for the Minimum Latency problem in symmetric metrics is 3.59 by Chaudhuri

et al. [25].

A generalization to the setting where we want to find a collection of k paths in a symmetric metric that start at a common node s was considered by Fakcharoenphol *et al.* [36] where they present a constant factor approximation. The constant factor was subsequently improved by Chekuri and Kumar [27]. More generally, Chekuri and Kumar [27] present a constant-factor approximation when the repairmen start at different nodes. Finally, a 6-approxomation was presented by Chaudhuri *et al.* [25] for the setting of k repairmen with possibly different start locations. Jothi and Raghavachari [57] develop a constant-factor approximation for a variant of the problem with k repairmen starting at a common node s where each node also has an associated start time.

In asymmetric metrics the only approximation algorithm known before our work for the Minimum Latency problem was an $O(n^{\frac{1}{2}+\epsilon})$-approximation by Nagarajan and Ravi [70] that does not bound the integrality gap of any LP relaxation. They also demonstrate that an α-approximation for Minimum Latency in asymmetric metrics also implies a 4α-approximation for ATSP. That is, approximating the Minimum Latency problem in asymmetric metrics is, asymptotically, at least as hard as approximating ATSP.

The first section of this chapter reviews the basic ideas behind approximating the Minimum Latency problem in symmetric metrics. Not all proofs are provided here, but some of the ideas that are mirrored in our algorithm are highlighted. The section following this establishes a technical result concerning the integrality gap of ATSPP that is necessary for our Minimum Latency algorithm. The LP relaxation for ATSPP ensures that every cut separating s from any other node has capacity at least 1. We need similar results for a similar LP that has the bound on cut constraints relaxed to 2/3 or 1/2. We show, more generally, that if the cut constraint is replaced by $x(\delta^+(S)) \geq \alpha$ where $\alpha > 1/2$ and if the indegree and outdegree of a node is only forced to be equal (not to both be equal to 1), then the integrality gap is still $O(\alpha^{-1} \log n)$. If the cut constraint is replaced by $x(\delta^+(S)) \geq \frac{1}{k}$ for some integer $k \geq 2$, then we apply the bicriteria approximation result for k-ATSPP from Section 3.3 to find $\left(1 + \frac{1}{b}\right) \cdot k$ paths whose total cost is $O(bk \log n)$ times the optimum value of the LP. Then, we describe the LP relaxation for the Minimum Latency problem and discusses how to round a feasible point in this LP to an integer point while increasing the objective function only by an $O(\log n)$ factor.

The $O(\log n)$-approximation for Minimum Latency in asymmetric metrics in Section 4.3 appeared in [43]. An asymptotically weaker result than Corollary 4.2.3 also appeared in this paper.

4.1 A Review of Minimum Latency in Symmetric Metrics

We review the basic ideas behind the constant-factor approximation for the Minimum Latency problem in symmetric metrics by Blum *et.al.* [18]. Though many improvements to the constant factor in the approximation algorithm have been presented since [18], they are more or less clever refinements of the basic principles we discuss here. The main deviation of our presentation from the presentation

in [18] is that we discuss optimum paths that visit k nodes rather than optimum trees that include k nodes. Since we are only establishing some intuition regarding the problem here, the simplicity of paths over trees is preferred. Also, the intuition gained by discussing paths translates more directly to our algorithm for Minimum Latency in asymmetric metrics in Section 4.3.

Let $G = (V, E)$ be an undirected graph with edge distances d_{uv} satisfying the triangle inequality and let $s \in V$ be the start node. Since we are dealing with symmetric metrics, we have $d_{uv} = d_{vu}$ for all $u, v \in V$. Let OPT denote the total latency of an optimum path that visits nodes in the order $s = v_1^*, v_2^*, \ldots, v_n^*$. Say the distance from s to v_k^* on this path is D_k. For any integer $1 \leq k \leq n$, let P_k denote the optimum cost of a path in G that starts at s and visits $k-1$ other nodes in V (any will do) and say that this path is $\mathcal{P}_k$. Note that $P_k \leq D_k$. For simplicity in this discussion, we assume that we already know the paths $\mathcal{P}_k$ (it is NP-hard to compute them).

Assume, by scaling, that all distances are at least 1. First, coarsely organize the nodes in sets $S_j = \{v_k : 2^j \leq D_k < 2^{j+1}\}$ by their latencies in the optimum solution. Let m_j be the maximum number of nodes that can be covered by a path with cost less than 2^{j+1}. That is, $P_{m_j} < 2^{j+1}$ whereas $P_{m_j+1} \geq 2^{j+1}$. We construct a Hamiltonian path in G starting at s as follows. For $j = 0, 1, \ldots$ in order, follow the path from s to the end of $\mathcal{P}_{m_j}$ and then back to s again. Report the first time each node is visited. So, the final solution is constructed by joining the paths $\mathcal{P}_{m_0}, \mathcal{P}_{m_1}, \ldots$ at the common start s and then performing a depth-first search starting at r that first goes down $\mathcal{P}_{m_0}$, then $\mathcal{P}_{m_1}$, and so on.

Here is how to bound the cost of these paths. For a node v_k^* in, say, S_j, we have that the latency of v_k^* is less than 2^{j+1}. So, $m_j \geq k$ since the path of cost D_k obtained by visiting the first k nodes in the optimum solution has cost less than 2^{j+1}. This means the k'th node in the Hamiltonian path we just constructed is visited before completing the traversal of $\mathcal{P}_{m_j}$. Since the cost of $\mathcal{P}_{m_j}$ is at most $P_{m_j} \leq 2^{j+1}$ and since each edge on $\mathcal{P}_{m_{j'}}, 1 \leq j' \leq j$, is traversed at most twice before reaching the k'th node, then the latency of the k'th node in our solution is at most $2 \sum_{0 \leq j' \leq j} 2^{j'+1} \leq 8 \cdot 2^j$.

Finally, since the latency of v_k^* is at least 2^j where $v_k^* \in S_j$, then the latency of the k'th node visited in our solution is at most 8 times more than the latency of v_k^*. Summing over all k shows that the total latency of our solution is at most 8 times the total latency of the optimum solution. Note that this algorithm assumes we can find optimum paths that start at r and visit $k-1$ other nodes which is impossible to do efficiently unless P = NP, but a variant of this algorithm produces an $8c$-approximation for the Minimum Latency problem if we only have a c-approximation for computing these paths.

Of course, this approach will not work in asymmetric metrics because we cannot simply travel "backwards" along a path and expect to have a reasonable bound on the cost. That is, even if we have a c-approximation to the problem of finding the cheapest path starting at s that visits k nodes, we cannot stitch these paths together for varying values of k as in the undirected case because the cost of returning to s before traversing the next path might be too expensive.

In our algorithm for Minimum Latency in asymmetric metrics, we devise and solve a linear program that determines a "fractional latency" for each node. Then, we group the nodes according to their fractional latencies in the LP and approximate a path that visits most nodes in one particular group using algorithms from Chapter 3. The cost of these paths is bounded by a logarithmic factor of the latency of the last node in the path which is why we require the integrality gap bounds, not just approximation algorithms, for ATSPP problems. The nodes in a group that are not covered by a path are moved into the next group. These paths will be stitched in the naive way: once we reach the end of one path in one group, go to the next unvisited node in the next group. As mentioned before, we need to bound the cost of these "back edges". We do this by using new "ordering" constraints in the LP that have no analog in the linear programming relaxations for the problems considered in Chapter 3. These help us bound the cost of these back edges by a reasonable factor of the cost of the LP optimum.

4.2 Relaxed Cut Constraints for ATSPP

In this section, we define a more general LP relaxation for ATSPP. Given some value $0 \leq \alpha \leq 1$, denote the following LP as LP(α).

$$\begin{aligned}
\text{minimize:} \quad & \sum_{uv \in A} c_{uv} x_{uv} && && (4.1)\\
\text{such that:} \quad & x(\delta^+(v)) = x(\delta^-(v)) && \forall v \in V - \{s,t\} &\\
& x(\delta^+(s)) = x(\delta^-(t)) = 1 &&&\\
& x(\delta^-(s)) = x(\delta^+(t)) = 0 &&&\\
& x(\delta^+(S)) \geq \alpha && \forall \{s\} \subseteq S \subsetneq V & (4.2)\\
& x_{uv} \geq 0 && \forall uv \in A &
\end{aligned}$$

If $\alpha > 0$, it is easy to argue that optimum integer points in this LP correspond to optimum Hamiltonian paths. We want to bound the integrality gap of this LP, but the weaker cut constraints make this difficult. To do this, we recall the path/cycle cover LP.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e \in A} c_{uv} x_{uv} && && (4.3)\\
\text{such that:} \quad & x(\delta^+(v)) = x(\delta^-(v)) = 1 && \forall v \in V - \{s,t\} &\\
& x(\delta^+(s)) = x(\delta^-(t)) = 1 &&&\\
& x(\delta^-(s)) = x(\delta^+(t)) = 0 &&&\\
& x_{uv} \geq 0 && \forall uv \in A &
\end{aligned}$$

If we also added the constraint $x(\delta^+(v)) = 1$ for each $v \in V - \{s,t\}$ to LP 4.1, then it is easy to argue that the optimum path/cycle cover is at most the optimum value of LP 4.1 and then we could

proceed as in Section 3.2. However, without this extra constraint we have to resort to approximate bounds.

Lemma 4.2.1 *Let $W \subseteq V$ be a subset containing s and t and let $\alpha > 1/2$. Given a solution x to LP(α) with cost at most L, a feasible solution to LP 4.3 on W of cost at most $\frac{3}{2\alpha-1}L$ can be found.*

Proof. Multiply x by $1/\alpha$. Now it constitutes a flow F of $1/\alpha$ units from s to t. Constraints (4.2), restricted to sets of size 1, imply that each node u now has at least one unit of flow going through it. Find a flow decomposition of F into paths and cycles, so that the union of the paths is acyclic. Let $F = F_p + F_c$, where F_p is the sum of flows on the paths in our decomposition, and F_c is the sum of flows on the cycles.

Choose some γ such that $\frac{1}{2\alpha} < \gamma < 1$. For any node u such that the amount of F_p flow going through u is less than γ, shortcut any flow decomposition paths that contain u (splitting-off techniques as in Theorem 3.1.3 are not needed), so that there is no more F_p flow going through u. Let $U \subseteq W$ be the set of vertices still participating in the F_p flow. Then each vertex in U has at least γ units of F_p flow going through it, and each vertex in $W - U$ has at least $1 - \gamma$ units of F_c flow going through it.

We find a topological ordering of vertices in U according to F_p (which is acyclic), and let P be an s-t path that visits the nodes of U in this topological order. We claim that the cost of P is within a constant factor of the cost of F_p. The argument for this is similar to one in the proof of Theorem 3.2.6. Out of $1/\alpha$ units of flow going from s to t in F_p, each vertex $u \in U$ carries γ units, which is more than half of the total amount (as $\gamma > 1/2\alpha$). So for any two such vertices u and v, there must be shared flow paths that carry flow of at least $2\gamma - 1/\alpha$ units. In particular, for every two consecutive nodes $u, v \in P$, F_p must contain such shared paths in which v immediately follows u. So the cost of P is at most $\frac{1}{2\gamma-1/\alpha}$ times the cost of F_p.

We now define $\tilde{x}$ as a flow equal to one unit of s-t flow on the path P plus $\frac{1}{1-\gamma}$ times the flow F_c. If any node $v \neq s, t$ has $\tilde{x}(\delta^+(v)) > 1$, then we can bypass flow in $\tilde{x}$ around v (again, without splitting off) until $\tilde{(}\delta^+(v)) = 1$. We claim that $\tilde{x}$ is now a feasible solution to LP 4.3: there is exactly one unit of flow from s to t and no flow enters s or exits t (as F_c consists of cycles not containing s or t); there is flow conservation at all nodes except s and t. Now, before bypassing some flow in $\tilde{x}(\delta^+(v))$ we have that every node in $W - \{s, t\}$ supported at least one unit of flow in $\tilde{x}$ for the following reason. If v is on the path P then it gets at least one unit of flow from the path. Otherwise, v supported at least $1 - \gamma$ flow from F_c which, after scaling by $\frac{1}{1-\gamma}$, means v supported at least 1 unit of flow in $\tilde{x}$. After shortcutting, every $v \in W - \{s, t\}$ then supports exactly one unit of flow in $\tilde{x}$.

The cost of this solution is at most

$$\frac{1}{2\gamma - 1/\alpha} \cdot cost(F_p) + \frac{1}{1-\gamma} \cdot cost(F_c) \quad \leq \quad \max\left(\frac{1}{2\gamma - 1/\alpha}, \frac{1}{1-\gamma}\right) \cdot \frac{1}{\alpha} L.$$

If we set $\gamma = \frac{1}{3} + \frac{1}{3\alpha}$, which satisfies $\frac{1}{2\alpha} < \gamma < 1$, we see that the cost of $\tilde{x}$ is at most $\frac{3}{2\alpha-1} \cdot L$. □

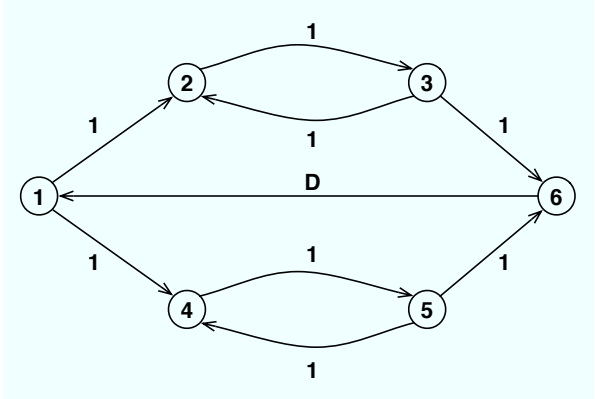

Figure 4.2: Bad gap example for LP (α) with $\alpha = 1/2$. Here, D is an arbitrarily large integer.

Now we can bound the integrality gap of LP(α) by the following.

Corollary 4.2.2 *For $\alpha > \frac{1}{2}$, the integrality gap of LP(α) is at most $\frac{6 \log_2 n + 3}{2\alpha - 1}$.*

Proof. We use Algorithm 4 as in the proof of Theorem 3.2.6 to bound the integrality gap. The only difference is that we cannot bound the cost of the optimum path/cycle of W cover by OPT because we do not have constraints $x(\delta^+(v)) = 1$ and $x(\delta^-(v)) = 1$ for $v \in W - \{s, t\}$ in LP 4.1. However, Lemma 4.2.1 says that the optimum path/cycle cover on a subset W containint s and t is at most $\frac{3 \cdot OPT}{2\alpha - 1}$ so we can proceed as in the proof of Theorem 3.2.6 with this weaker bound. □

We note that for $\alpha \leq 1/2$, the gap between the optimum ATSPP solution and the optimum solution to LP(α) can be unbounded. For example, let D be an arbitrarily large value and consider the shortest path metric obtained from the graph in Figure 4.2. It is easy to check that the following assignment of x-values to the arcs is feasible for LP(α) with $\alpha = 1/2$. Assign a value of 1/2 to arcs (1,2), (3,2), (3,6), (1,4), (5,4), and (5,6) and a value of 1 to arcs (2,3), (4,5). Every other arc is assigned a value of 0. This assignment is feasible for the linear program and has objective function value 5. On the other hand, it is easy to verify that any Hamiltonian path from 1 to 6 has cost at least D.

On the other hand, if we multiply a solution to LP($1/k$) by k (and, perhaps, split off around some nodes so $x(\delta^+(v)) = x(\delta^-(v)) = 1$), it becomes a point in the relaxation for the k-ATSPP LP 3.5. The following is then a simple consequence of Theorem 3.3.3. It is also be used in Section 4.3.

Corollary 4.2.3 *For any integer $b \geq 1$, one can efficiently find a collection of $\left(1 + \frac{1}{b}\right) \cdot k$ paths from s to t of total cost at most $(b+1) \cdot k \cdot \log_2 n$ times the optimum value of LP($1/k$).*

Proof. Multiplying the solution to LP($1/k$) by k increases its cost by k. If necessary, split off flow passing through nodes $v \in V - \{s, t\}$ until they support exactly 1 unit of flow. Theorem 3.3.3 then says we can then find $\left(1 + \frac{1}{b}\right) \cdot k$ paths from s to t whose union spans all nodes while losing at most an additional factor of $(b+1) \log_2 n$. □

We note that a similar result in [43] shows how to find at most $k \log_2 n$ paths whose total cost is at most $k \log_2 n$ times the optimum solution to LP($1/k$). While the number of paths found by this result is much more than the number found in Corollary 4.2.3 (even for $b = 1$), the bound on the cost of these paths is less than the bounds in Corollary 4.2.3 (though, only by a constant factor when using $b = 1$ in the corollary). It's interesting to note that even this result, which uses many more paths, would have been sufficient for the results in Section 4.3 after making minor modifications to Algorithm 8 and some proofs in that section. The interested reader can find the details in [43].

4.3 Approximating Minimum Latency in Asymmetric Metrics

The algorithm in this section for Minimum Latency in asymmetric metrics works when both the start and end nodes are specified. We may suppose that the end location t is specified by guessing all possible endpoints and running the proceeding algorithm. So, our instance of the Minimum Latency problem is to find a Hamiltonian path from the start node s to the end node t such that the total waiting time of all nodes along this path (including t) is minimized.

The LP relaxation we use is quite large. To gain some intuition regarding its formulation, we discuss some of the motivation behind the constraints before presenting the final LP. First, minimizing the average waiting time and the total waiting time is essentially the same goal; the total waiting time is precisely a factor n larger than the average waiting time. So, we say that our goal is to minimize the total waiting time.

Now, if P is the path used in an optimum solution then we can view the total waiting time (not average) as the sum of the costs of each of the subpaths of P that start at s. We formulate the LP to have a collection of variables for each node v that model a path from s to v as a flow of value 1. To add some consistency between flows for different nodes v, we can add constraints to ensure that each such flow is a "subflow" of the flow from s to t. So, for each node v and each edge $e = uw$, we let f^v_{uw} be the total amount of the unit flow from s to v supported by edge uw. Saying that the flow from s to v is a subflow of the flow from s to t amounts to placing the constraint $f^v_{uw} \leq f^t_{uw}$ for each edge uw. The objective function is then the sum of the costs of each of these flows.

In the introduction to this chapter, we mentioned that many paths are generated throughout the course of the rounding algorithm rather than a single path from s to t. Then, we stitch these paths together by traveling from the end of one path to the first unvisited node in another path. To bound the cost of stitching these paths together, we introduce ordering variables x_{uv} where the idea is that u comes before v in the solution when $x_{uv} = 1$. Since the nodes are totally ordered by the optimum path P, we have $x_{uv} + x_{vu} = 1$ for all distinct pairs of node. In other words, it must be that either u comes before v or v comes before u.

We actually require a refinement of the notion of ordering constraints. In an optimum solution, if u comes before v then the subpath from s to v passes through u. More generally, for any triple of distinct nodes u, v, w we use x_{uvw} to indicate that v lies after u and before w in the solution.

Again, for distinct u, v, w we have the following. If u comes before w, then either v appears before u, between u and w, or after w. In an integer solution, this amounts to $x_{vuw} + x_{uvw} + x_{uwv} = 1$. Finally, if u comes after w then $x_{vuw} = x_{uvw} = x_{uwv} = 0$ in an integer solution. Both cases are modelled in the LP by the constraint $x_{vuw} + x_{uvw} + x_{uwv} = x_{uw}$.

Next, we can force the flow from s to v to pass through any u appearing before v in the following way. For any set S containing u but not containing s, we can add a constraint that says the total flow in f^v on arcs exiting S must be at least x_{uv}. In the ideal case where the x values are integers, if u comes before v then this ensures that the unit of flow from s to v passes through u. Otherwise, if u comes before v then no flow is required to pass through u.

One final class of variables we add, for technical reasons to be seen later, is a variable $\ell(v)$ for each v that is the "latency" of node v. In an optimum integer solution, this is simply the cost of the subpath from s to v. However, we also know that if u, v, w appear in this order, then the cost of the path to w is at least $d_{su} + d_{uv} + d_{vw}$. So, we may constrain the latency of w to be at least $(d_{su} + d_{uv} + d_{vw}) \cdot x_{uvw}$; this will be useful in the rounding procedure. LP 4.4 is the linear program we use based on these ideas.

$$\text{minimize: } \sum_{v \neq s} \ell(v) \tag{4.4}$$

$$\begin{aligned}
\text{such that: } \ell(v) &\geq \sum_{uw} d_{uw} f^v_{uw} && \forall v && \\
\ell(v) &\geq [d_{su} + d_{uw} + d_{wv}]\, x_{uwv} && \forall u, w, v : |\{u, w, v\}| = 3 && (4.5)\\
\ell(t) &\geq \ell(v) && \forall v && \\
x_{uw} &= x_{vuw} + x_{uvw} + x_{uwv} && \forall u, w, v : |\{u, w, v\}| = 3 && (4.6)\\
x_{uw} + x_{wu} &= 1 && \forall u, w : u \neq w && (4.7)\\
x_{su} = x_{ut} &= 1 && \forall u \notin \{s, t\} && (4.8)\\
\sum_w f^v_{wu} &= \sum_w f^v_{uw} \forall v, \forall u \notin \{s, v\} && && (4.9)\\
\sum_w f^v_{sw} = \sum_w f^v_{wv} &= 1 && \forall v && (4.10)\\
f^v_{us} = f^v_{vu} &= 0 && \forall u, v && (4.11)\\
\sum_w f^v_{uw} &= x_{uv} && \forall v, u \neq v && (4.12)\\
f^v_{uw} &\leq f^t_{uw} && \forall u, w, v && (4.13)\\
f^v(\delta^+(S)) &\geq x_{yv} && \{s\} \subseteq S \subsetneq V, y \in S && (4.14)\\
x_{uw}, x_{uwv}, f^v_{uw} &\geq 0 && \forall u, w, v &&
\end{aligned}$$

We note that the LP can be solved in polynomial time using the ellipsoid method. The number of variables is $O(n^3)$ and there are only polynomially many instances of each constraint type except Constraints (4.14). However, we can use a max-flow/min-cut algorithm to separate over these con-

straints. In particular, for every v we form a graph G_v on V where the capacity of an edge wx is f^v_{wx}. Then for every y as in the constraint, we check that the minimum y, v cut in G_v has value at least x_{yv}.

First, we prove that integer solutions correspond to feasible paths with the same total latency. Let P be a Hamiltonian path from s to t. Set $x_{uv} = 1$ if u appears before v on P, and 0 otherwise. Similarly, set $x_{uvw} = 1$ if u, v, w appear in this order on P, otherwise set $x_{uvw} = 0$. Let $l(v)$ denote the cost of the subpath of P from s to v. Finally, for a node v, set $f^v_{uw} = 1$ if u appears immediately before w and w appears sometime before v on P. All constraints are easy to check if we remember that variables of the form f^v are simply indicators of the edges of the subpath of P from s to v.

Conversely, suppose x, f are integer points in the polytope. The x variables are clearly only 0 or 1 (*e.g.* Constraints (4.7) and (4.6)). From this and Constraint (4.12) we must have $f^v_{uw} \leq x_{uv}$ for $u, v, w : u \neq v$. For $u = v$, we have $f^v_{v,w} = 0$ by (4.11). Thus, each variable in the LP is assigned a value of only 0 or 1. Let P_v denote the set of edges $\{uw : f^v_{uw} > 0\}$. Then the out-degree of s and in-degree of t is exactly 1 using edges in P_v by Constraint (4.10). By the flow conservation Constraints (4.9), P_v is a path from s to v plus, perhaps, some cycles on nodes not on this path. Now, if $f^v_{uw} = 1$ then f^t_{uw} as well by Constraint (4.13). So, if P_v contains a cycle then P_t must also contain a cycle. But this is impossible since if C were such a cycle, Constraint (4.14) would be violated for $v = t, S = C$ and any node $y \in C$ because $x_{yt} = 1$. So, P_v is just a path from s to v. Since any such path must be a subpath of P_t, then P_v must be a subpath of P_w for v, w with $x_{vw} = 1$. Finally, the cost of P_v is exactly $l(v)$ so the total latency of the path P_t is equal to the value of the LP under this integer solution.

We begin the rounding algorithm with a lemma that allows to group the nodes into only $O(\log n)$ groups based on their latency.

Lemma 4.3.1 *Given a feasible solution to LP (4.4) with objective value L, we can find another solution of value at most $(1 + \frac{1}{n})L$ in which the ratio of the largest to smallest latency $\ell(\cdot)$ is at most n^2.*

Proof. Let (x, ℓ, f) be a feasible solution with value L, with $\ell(t)$ the largest latency value in this solution. Note that $L \geq \ell(t)$. Define a new feasible solution (x, ℓ', f) by $\ell'(v) = \max\{\ell(v), \ell(t)/n^2\}$. The total increase in the objective function is at most $n \cdot \frac{\ell(t)}{n^2} \leq L/n$ as there are n nodes in total. Thus, the objective value of this new solution is at most $(1 + 1/n)L$. □

Using Lemma 4.3.1 and scaling the edge lengths (if needed), we can assume that we have a solution (x, ℓ, f) satisfying the following:

Corollary 4.3.2 *There is a feasible solution (x, ℓ, f) in which the smallest latency is 1 and the largest latency is at most n^2 and whose cost is at most $(1 + \frac{1}{n})$ times the optimum LP solution.*

Let L^* be the value (i.e. total latency) of this solution.

The idea of our algorithm is to construct s-v paths for several nodes v, such that together they cover all vertices of V, and then to "stitch" these paths together to obtain one Hamiltonian path. We use our results for ATSPP to construct these paths. For this, we observe that parts of a solution to the latency LP (4.4) can be transformed to obtain feasible solutions to different instances of LP(α). For example, we can construct a Hamiltonian s-t path of total length $O(\log n) \cdot \ell(t)$ as follows. From a solution to LP (4.4), take the t-flow defined by the variables f^t_{uw}, and notice that it constitutes a feasible solution to LP 3.3. In particular, since $x_{yt} = 1$ for all y, Constraints (4.14) of LP (4.4) for $v = t$ imply that the cut constraints of LP 3.3 are satisfied. The objective function value for LP 3.3 of this solution is at most $\ell(t)$. Thus, by Theorem 3.2.6, we can find the desired path. Of course, this path is not yet a good solution for the latency problem, as even nodes v with $\ell(v) \ll \ell(t)$ can have latency in this path close to $O(\log n) \cdot \ell(t)$. Our algorithm constructs several paths of different lengths, incorporating most nodes v into paths of length $O(\log n) \cdot \ell(v)$, and then combines these paths to obtain the final solution.

Algorithm 8 An $O(\log n)$-Approximation for Minimum Latency in Asymmetric Metrics

1: Let (x, ℓ, f) be a solution to LP (4.4) as described by Corollary 4.3.2. Let S be the path $\{s\}$.
2: Partition the nodes into $g = \lfloor \log_2 \ell(t) + 1 \rfloor$ sets $V_1, \ldots, V_g$ with $v \in V_i$ if $2^{i-1} \leq \ell(v) < 2^i$.
3: **for** $i = 1$ to $g - 1$ **do**
4: **for** $j = 1$ to 2 **do**
5: **if** $V_i \neq \emptyset$ **then**
6: Let $v_i^j = \arg\max_{v \in V_i} |\{u \in V_i : x_{uv} \geq \frac{1}{2}\}|$ ▷ this maximizes the size of B_i^j below
7: Let $A_i^j = \{u \in V : x_{uv_i^j} \geq \frac{2}{3} + \frac{2i-2+j}{24 \log_2 n}\}$
8: Let $B_i^j = \{u \in V_i : x_{uv_i^j} \geq \frac{1}{2}\}$ ▷ $|B_i^j| \geq (|V_i| - 1)/2$
9: Find an s-v_i^j path P_i^j, containing A_i^j, of cost $\delta_1 \log_2 n \cdot 2^i$; append P_i^j to S.
10: Find two s-v_i^j paths $\mathcal{P}_i^j$, containing B_i^j, of cost at most $8 \log_2 n \cdot 2^i$; append $\mathcal{P}_i^j$ to S.
11: $V_i = V_i - (A_i^j \cup B_i^j \cup \{v_i^j\})$ ▷ size of V_i is at least halved
12: **end if**
13: **end for**
14: Let $V_{i+1} = V_{i+1} \cup V_i$ ▷ remaining nodes are carried over to the next set
15: **end for**
16: Construct an s-t path P_g, containing V_g, of cost at most $(2 \log_2 n + 1) \cdot \ell(t)$. Append P_g to S.
17: Shortcut S over the later copies of repeated nodes. Output S.

4.3.1 Constructing the Paths

Algorithm 8 finds an approximate solution to the Minimum Latency problem, and we now explain how some of its steps are performed. The algorithm maintains a path S, initially containing only the start node s, and gradually adds new parts to it. This is done through operation *append* on lines 9, 10, and 16. To append a path P to S means first to extend S to incorporate the nodes of P by going from the last node of S to the first node in P not already covered by s and then following path P. The new path is obtained from this walk by shortcutting past nodes that appeared earlier in the walk. If all nodes in P are already covered by S, then the result of appending P to S is simply S. For

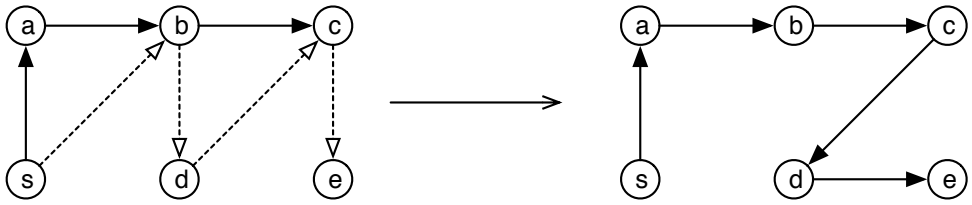

Figure 4.3: Appending the dashed path to the solid path.

example, if $S = sabc$ and $P = sbdce$, the walk that results is $sabcdce$. After shortcutting, the walk is simply $sabcde$. See Figure 4.3 for an illustration. Step 10 appends a set of paths to S. This just means sequentially appending all paths in the set, in arbitrary order, to S.

Next we describe how to build paths P_i^j and $\mathcal{P}_i^j$ in Steps 9 and 10. We described above how to use Corollary 4.2.2 to build a Hamiltonian s-t path P of length $(2\log_2 n+1)\cdot\ell(t)$, which is used on line 16 of the algorithm. The idea behind building paths P_i^j and $\mathcal{P}_i^j$ with their corresponding length guarantees is similar.

To construct P_i^j, we do the following. Since each node $u \in A_i^j$ has $x_{uv_i^j} \geq 2/3$, the amount of v_i^j-flow that goes through u is at least $2/3$. We apply splitting-off on this flow to nodes outside of A_i^j, and obtain a total of one unit of s-v_i^j flow over the nodes in A_i^j, of cost no larger than $\ell(v_i^j) \leq 2^i$. This flow satisfies all the constraints of LP($\alpha = 2/3$), including the set Constraints (4.2), which are implied by the set Constraints (4.14) of the latency LP (4.4), as $x_{uv_i^j} \geq 2/3$ for $u \in A_i^j$. Thus, using Corollary 4.2.2, we can find a path from s to v_i^j, spanning all the nodes of A_i^j, whose cost is at most $\delta_1 \log_2 n \cdot 2^i$ for some constant δ_1.

To obtain the set of paths $\mathcal{P}_i^j$, we look at the v_i^j-flow going through each node of B_i^j, whose amount is at least $\frac{1}{2}$. Apply splitting-off on the v_i^j-flow to nodes outside of B_i^j to, again, obtain a total of one unit of s-v_i^j flow over the nodes in B_i^j of cost nol arger than $\ell(v_i^j) \leq 2^i$. Furthermore, the amount of flow passing through each node in B_i^j remains at least $\frac{1}{2}$, so the resulting flow satisfies all the constraints of LP($\alpha = 1/2$) including the set Constraints (4.2). This time, use Corollary 4.2.3 with $b = 3$ to obtain 2 paths from s to v_i^j which span all nodes in B_i^j with total cost at most $8 \cdot \log_2 n \cdot 2^i$.

4.3.2 Connecting the paths

We now bound the lengths of edges introduced by the append operation in the different cases. The cost of the path obtained by appending path P to the existing path S is at most the cost of P plus the cost of S plus the cost of the edge from the end of S to the first node in P that does not appear in S. Let $app(P)$ denote the cost of this new edge.

Lemma 4.3.3 *For any i, j, and path $P \in \mathcal{P}_i^j$, $app(P) \leq 6 \cdot 2^i$. Also, $app(P_g) \leq 6 \cdot 2^g$.*

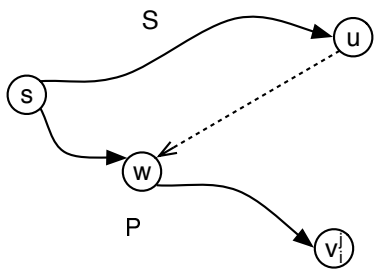

Figure 4.4: An illustration of the nodes u, w and v_i^j on the paths S and P in the proof of Lemma 4.3.3. The dashed edge is the edge that was used to "stitch" the paths together in the append operation.

Proof. Let u be the last node of the path S before the append operation, v_i^j be the last node of P, and w be the first node of P that does not appear in S. If there is no such node, then the append operation simply returns S and there is no cost increase. The paths S, P and nodes u, w and v_i^j are illustrated in Figure 4.4. We need to bound d_{uw}, the distance from u to w.

We observe that $x_{wu} \leq 5/6$. If $u = s$, this is trivial. Otherwise, $u = v_{i'}^{j'}$ is the endpoint of some path constructed in an earlier iteration. Note that $j' \leq 2$ and $i' \leq g - 1 \leq \log_2 \ell(t) \leq 2\log_2 n$ by our assumption that $\ell(t) \leq n^2$, which means that $\frac{5}{6} \geq \frac{2}{3} + \frac{2i'-2+j'}{24\log_2 n}$. So, if we had $x_{wu} > 5/6$, then w would be included in the set $A_{i'}^{j'}$ and in the path $P_{i'}^{j'}$, and thus be already contained in S, which is a contradiction (note it cannot be that w was in another path in $\mathcal{P}_i^j$ since these paths share only s and v_i^j).

Consequently, $x_{uw} = 1 - x_{wu} \geq 1/6$. This means that the amount of w-flow that goes through u is at least $1/6$. Since this flow has to reach w after visiting u, it has to cover a distance of at least d_{uw}, thus adding at least $\frac{1}{6} \cdot d_{uw}$ to $\ell(w)$, the latency of w. Thus, $\ell(w) \geq \frac{1}{6} d_{uw}$, and $d_{uw} \leq 6\ell(w)$. Now, if $w \in \mathcal{P}_i^j$, it must be in B_i^j, which, by definition, means that $w \in V_i$, and therefore $\ell(w) \leq 2^i$. So $app(P) = d_{uw} \leq 6 \cdot 2^i$. If $w \in P_g$, then $app(P_g) \leq 6\ell(w) \leq 6\ell(t) \leq 6 \cdot 2^g$. □

To bound the cost of appending a path P_i^j to S, we need an auxiliary lemma. Informally, it says that if the LP suggests that u, w, v should appear in this order, then the latency $\ell(v)$ of v is a significant fraction of the distance from u to w.

Lemma 4.3.4 *For any $\epsilon > 0$, if $x_{uw} + x_{wv} \geq 1 + \epsilon$, then $\ell(v) \geq \epsilon \cdot d_{uw}$.*

Proof. Using Constraint (4.6) we have:

$$\begin{aligned} 1 + \epsilon &\leq x_{uw} + x_{wv} \\ &= (x_{vuw} + x_{uvw} + x_{uwv}) + (x_{uwv} + x_{wuv} + x_{wvu}) \\ &= 2x_{uwv} + (x_{vuw} + x_{uvw}) + (x_{wuv} + x_{wvu}). \end{aligned}$$

On the other hand, $(x_{vuw} + x_{uvw}) + (x_{wuv} + x_{wvu}) \leq x_{vw} + x_{wu} = 2 - (x_{uw} + x_{wv}) \leq 1 - \epsilon$, using again Constraint (4.6), then Constraint (4.7), and the assumption of the lemma. Therefore,

$2x_{uwv} \geq (1+\epsilon) - (1-\epsilon) = 2\epsilon$, i.e. $x_{uwv} \geq \epsilon$. Then the claim follows using Constraint (4.5). □

Lemma 4.3.5 *For any i and j, $app(P_i^j) \leq 24 \log_2 n \cdot 2^i$.*

Proof. Let u, v_i^j, and w be as in the proof of Lemma 4.3.3. To bound d_{uw}, we consider two cases.

Case 1: If $w \in V_i$, we apply the same proof as for Lemma 4.3.3 and conclude that $app(P_i^j) \leq 6 \cdot 2^i$.

Case 2: If $w \notin V_i$, let (i', j') be an earlier iteration of the algorithm in which node $u = v_{i'}^{j'}$ was added to S. Since $w \notin S$, it must be that $w \notin A_{i'}^{j'}$, and thus $x_{wu} < \frac{2}{3} + \frac{2i'-2+j'}{24 \log_2 n}$. On the other hand, since $w \in A_i^j$, it must be that $x_{wv_i^j} \geq \frac{2}{3} + \frac{2i-2+j}{24 \log_2 n}$. Because $2i' + j' \leq 2i + j - 1$, we have

$$\begin{aligned} x_{uw} + x_{wv_i^j} &= (1 - x_{wu}) + x_{wv_i^j} \\ &\geq 1 - \frac{2i'-2+j'}{24 \log_2 n} + \frac{2i-2+j}{24 \log_2 n} \\ &\geq 1 + \frac{1}{24 \log_2 n}. \end{aligned}$$

Using Lemma 4.3.4, we get that
$app(P_i^j) = d_{uw} \leq 24 \log_2 n \cdot \ell(v_i^j) \leq 24 \log_2 n \cdot 2^i$. □

4.3.3 Bounding the Cost

Now we bound the latencies of the nodes in the path returned by Algorithm 8.

Lemma 4.3.6 *Suppose that a node v is first added to path S in iteration k of the outer loop of the algorithm. Then the latency of v in S is at most $\delta_2 \log_2 n \cdot 2^k$, for some constant $\delta_2 > 0$.*

Proof. Let $len(P)$ denote the length of a path P. The latency of node v on S is at most:

$$\begin{aligned} & \sum_{i=1}^{k} \sum_{j=1}^{2} \left[len(P_i^j) + \sum_{P \in \mathcal{P}_i^j} len(P) + app(P_i^j) + \sum_{P \in \mathcal{P}_i^j} app(P) \right] \\ \leq & \sum_{i=1}^{k} \sum_{j=1}^{2} \left[\delta_1 \log_2 n \cdot 2^i + 8 \log_2 n \cdot 2^i + 24 \log_2 n \cdot 2^i + 2 \cdot 6 \cdot 2^i \right] \\ \leq & \delta_2 \log_2 n \cdot 2^k \end{aligned}$$

□

Suppose that n_i is the number of nodes that are originally placed into the set V_i. Since a node v is originally placed in V_i if $\ell(v) \geq 2^{i-1}$, the value of the LP solution L^* can be bounded by:

$$L^* = \sum_v \ell(v) \geq \sum_{i=1}^{g} n_i \, 2^{i-1}. \tag{4.15}$$

Suppose node v was initially placed in V_i (so $\ell(v) \leq 2^i$). Ideally, we would like to bound the final latency of v by $O(\log n) \cdot 2^i$. If we could do this, then the total latency would be at most

$O(\log n)\sum_{i=1}^{n} n_i 2^i \leq O(\log n)L^*$. However, the algorithm may move some node from V_i to some V_j with $j > i$ and the bound on their latency cannot be described so simply. The key observation is that at most one quarter of the nodes in V_i are moved up to V_{i+1} after iteration i of the outer loop. Since the latencies of the groups double as i increases and since at most one quarter of each group V_i is moved to a higher group, then summing a geometric series shows that the average increase of the latency of a node is still $O(\log n)$.

More formally, let n'_i denote the size of V_i at the beginning of iteration i of the outer loop. Note that n'_i may be larger than n_i since some nodes may have been moved to V_i in Step 14 of the previous iteration.

Claim 4.3.7 *For any i, the size of the set V_i at the end of iteration i is at most $n'_i/4$.*

Proof. Consider the iteration $(i, j = 1)$. Note that the vertex v_i^j is chosen precisely to maximize the number of nodes u in V_i with $x_{uv_i^j} \geq 1/2$, which is the size of the set B_i^j.

Let $B_u = \{v \in V_i : x_{vu} \geq \frac{1}{2}\}$ at the start of iteration i. The size of B_u is the number of ordered pairs (v, u) with $x_{vu} \geq 1/2$. By Constraint (4.7) we have $x_{uv} \geq 1/2$ or $x_{vu} \geq 1/2$ for any two $u \neq v$ in V_i, so the total number of ordered pairs (u, v) with $x_{uv} \geq \frac{1}{2}$ is at least $\frac{n'_i(n'_i-1)}{2}$. Since there are n'_i nodes, then there must be some u such that at least $\frac{n'_i-1}{2}$ other nodes v have $x_{vu} \geq 1/2$. Since B_u consists of these nodes and u itself, then $|B_u| > \frac{n'_1}{2}$.

After iteration $j = 1$, we then have $|V_i| \leq n'_i/2$ since $B_i^1 \cup \{v_i^j\}$ is removed from V_i. Repeating this argument shows that $|V_i| \leq n'_i/4$ after iteration $j = 2$ because the size of V_i is again cut in half. □

Finally, we can prove our main result.

Theorem 4.3.8 *The total latency of path S returned by Algorithm 8 is $O(\log n) \cdot L^*$.*

Proof. From Claim 4.3.7, it follows that at most a $1/4$ fraction of the n'_i nodes that are in V_i at the beginning of iteration i are moved to the set V_{i+1} at the end of this iteration. Thus, for any $1 < i \leq g$, $n'_i \leq n_i + n'_{i-1}/4$. Inductively, this implies that $n'_i \leq \sum_{h=1}^{i} n_h/4^{i-h}$.

Now we claim that the total latency of the solution S is at most $\sum_{i=1}^{g} n'_i \cdot \delta_2 \log_2 n \cdot 2^i$. This is because at most n'_i nodes are added to S in iteration i, and each such node has latency at most $\delta_2 \log_2 n \cdot 2^i$ (using Lemma 4.3.6). Therefore, the total latency of the solution is at most:

$$\begin{aligned}
\sum_{i=1}^{g} n'_i \cdot \delta_2 \log_2 n \cdot 2^i &\leq \sum_{i=1}^{g} \delta_2 \log_2 n \cdot 2^i \cdot \sum_{h=1}^{i} \frac{n_h}{4^{i-h}} \\
&= \delta_2 \log_2 n \sum_{i=1}^{g} \sum_{h=1}^{i} 2^{h-i} \cdot 2^h \, n_h \\
&\leq \delta_2 \log_2 n \sum_{h=1}^{g} 2^h \, n_h \sum_{i=0}^{\infty} \frac{1}{2^i} \\
&\leq O(\log n) \cdot L^*,
\end{aligned}$$

using the bound on n'_i, re-ordering the summation, and using inequality (4.15). Combined with Corollary 4.3.2, this proves the theorem. □

One might wonder if the integrality gap of the LP relaxation 3.3 for ATSPP is $O(\log n / \log\log n)$ since the integrality gap for its ATSP counterpart is bound by the same ratio [9]. While bounding the integrality gap of LP 3.3 by $o(\log n)$ is still an open problem, it is natural to ask if this (and a similar improvement to the k-ATSPP relaxation 3.5) would imply a similar improvement to the approximability of the Minimum Latency problem in asymmetric metrics? This is not immediately true from our algorithm. As noted earlier, we lost an $O(\log n)$-factor for essentially two reasons. One is that the integrality gap for LP 3.3 (and for LP 3.5 with $k = 2$) was shown to be only $O(\log n)$. The other is that we grouped the nodes into $O(\log n)$ groups which affected the cost of the solution by $O(\log n)$ in Lemma 4.3.5.

If we wanted to scale the latencies in Lemma 4.3.1 so that the gap between the smallest and largest latencies was $f(n)$ with $\log_2 f(n) = o(\log N)$, this would require we increase each $\ell(v)$ to be at least $\ell(t)/f(n)$ and the bound on the cost increase would be $\left(1 + \frac{n}{f(n)}\right)$. But any $f(n)$ with $\log_2 f(n) = o(\log n)$ has $f(n) = o(n^\epsilon)$ for all constants $\epsilon > 0$ so the cost increase would be at least $\Omega(n^{1-\epsilon})$ for all constants $\epsilon > 0$. In other words, the cost would increase by far too much to guarantee a $o(\log n)$-approximation ratio. So, if one wanted to improve the approximability of Minimum Latency in asymmetric metrics given improved bounds on integrality gaps for ATSPP and k-ATSPP LP relaxations, then revisions to this approach, or a new approach altogether, are required.

Chapter 5

Conclusion

We conclude by discussing some directions for future work with the problems considered in this thesis and some of their variants. We first discuss directions for the Unsplittable Flow problem on Paths and then discuss directions for the Traveling Salesman variants discussed in this thesis. Directions for Minimum Latency problems will be discussed along with the Traveling Salesman variants.

5.1 Future Directions - Unsplittable Flow Problems on Paths and Trees

There are two open problems regarding UFP that, in the author's mind, seem most interesting. First is the question of a PTAS. While a constant factor approximation has recently been demonstrated for UFP [19], there is still a gap between the strong NP-hardness lower bound (also from [19]) and this constant upper bound. If the input demands are constrained to be integers at most quasi-polynomial in n, then the problem does indeed admit a quasi-PTAS, a $(1+\epsilon)$-approximation runnining in quasi-polynomial time for any constant $\epsilon > 0$ [12]. Can ideas from these two algorithms be combined to achieve a PTAS for the general problem? The constant-factor approximation deals with slack and tight tasks separately, but there are ideas in [12] that combine LP approaches and dynamic programming approaches to obtain the quasi-PTAS in their restricted setting. The LP based parts of the algorithm in [19] also only use the weaker LP relaxation 2.1 and do not utilize the stronger constraints in LP relaxation 2.2 or the strong LP relaxation considered in [26].

The second main problem is whether the integrality gap of LP relaxation 2.2 or, equivalently, the LP relaxation for UFP considered in [26] is bounded by a constant. As mentioned before, the integrality gap of this LP was shown to be $O(\log^2 n)$ which was subsequently improved to $O(\log n)$, but no super constant lower bounds are known. Furthermore, in this thesis we showed that the integrality gap was constant in certain sparse instances, namely q-conflicting instances when q is a fixed constant. The $O(\log n)$ bound on the integrality gap is shown by performing the same decomposition into intersecting instances as we considered in Section 2.2 and then demonstrating

that the integrality gap of the LP relaxation is bounded by a constant in intersecting cases. So, if the integrality gap is super constant then such an instance would have to resist decomposition onto collections of disjoint intersecting instances, much like the example presented at the beginning of Section 2.3.

An interesting place to start bounding the integrality gap of LP 2.2 would be the case with unit profits. Chuzhoy and Chalermsook [34] show that the Maximum Independent Set of Rectangles has an integrality gap that is bounded by $O(\log \log n)$ for the case when all rectangles have unit profit. Their algorithm also uses the constant-factor approximation for instances of Maximum Independent Set of Rectangles where each point in the plane is covered by at most a constant number of rectangles. Our algorithm for q-conflicting instances may prove useful here in a similar way the algorithm of Lewin-Eytan *et al.* [64] was used in [34].

There are also interesting open questions for the more general case of Unsplittable Flow in trees. The current best approximation algorithm has an $O(\log^2 n)$-approximation ratio, but the best lower bound is only constant. Determining the best polynomial-time approximation ratio (assuming some complexity theory barrier) is an interesting question. Is there a constant approximation for Unsplittable Flow in trees or can we prove a super-constant lower bound? In either case, the author believes that an $O(\log n)$-approximation should be obtainable. The results in this thesis critically rely on "left tight", "right tight" and "both tight" properties which does not generalize well to Unsplittable Flow with unbounded degree trees. However, a good start might be to generalize these algorithms (or the algorithm in [19]) to the case of bounded-degree trees.

The LP relaxation for UFP developed in [26] also does not generalize well to a polynomial-time solvable LP relaxation for Unsplittable Flow in trees. Another interesting problem is to devise an efficiently solvable/approximable LP relaxation for Unsplittable Flow in trees that has at most a polylogarithmic integrality gap. Perhaps it is even possible to produce an LP relaxation with a constant integrality gap in trees.

5.2 Future Directions - Asymmetric Traveling Salesman Path and Minimum Latency Problems

There are numerous interesting problems that should be addressed concerning the Traveling Salesman variants discussed in this thesis. Starting with ATSPP, we know from [38] that the approximability of ATSP and ATSPP differs by a multiplicative factor $2 + \epsilon$ for any constant $\epsilon > 0$. However, their result does not extend to integrality gaps. In particular, the author is interested in knowing if the integrality gaps for the Held-Karp relaxations for both ATSP and ATSPP differ by a constant factor.

A possible starting point would be to see if the recent algorithm in [9] that proves the integrality gap of the HK LP relaxation for ATSP is $O(\log n / \log \log n)$ can be adapted to prove the same asymptotic bound for the integrality gap of the HK LP relaxation for ATSPP. The first technical

challenge is that the assignment $z_e := x_{uv} + x_{vu}$ for the undirected graph obtained from a point x in the ATSP polytope already fits in the base polytope for the graphic matroid without scaling by $\frac{n-1}{n}$. The result in [9] relies on this scaling so that the resulting point in the spanning tree polytope is in the *relative interior* of this base polytope. This is not too serious as we could use, say, the *randomized swap rounding* technique by Chekuri *et al.* [31] to sample a random spanning tree T with the required negative correlation properties such that $Pr[e \in T] = z_e$. This works for *any* point in the polytope, not just one in the relative interior. A more serious obstacle to a straight-forward adaptation of the ATSP algorithm in [9] to ATSPP occurs in the last step. There is a technical hurdle in applying Hoffman's circulation theorem because the flow entering s and the flow exiting t are both zero.

For k-ATSPP, the obvious problem is to improve upon the $O(k \log n)$-approximation and integrality gap. Improving the $O(\log n)$ part of this guarantee is at least as challenging as improving the approximation ratio/integrality gap for ATSPP (for $k = 1$). The main question, in the author's mind, is whether the bound can be improved to $O(\log n \cdot \operatorname{polylog} k)$ or even $O(\log n)$. However, it could also be that the integrality gap of the relaxation considered in this thesis is $\Omega(k)$ even if ATSPP has a constant integrality gap.

Though General k-ATSPP seems very difficult, there are still some interesting questions. We demonstrated complexity barriers to approximating General k-ATSPP within any ratio for values of k being as small as polylogarithmic in the number of nodes. Maybe better approximations are possible for a constant number of (s_i, t_i) pairs. In particular, what if there are only two pairs $(s_1, t_1), (s_2, t_2)$? There is potential for this case to be difficult because the somewhat-related problem of finding edge-disjoint paths between two such pairs is NP-complete [39], but there could be a good approximation algorithm. Also, there might also be a good bicriteria approximation algorithm that may use each of the (s_i, t_i) salesman more than once.

Another question relates to bicriteria approximations for multiple traveling salesmen with multiple sources and/or multiple sinks (and even, perhaps, General k-ATSPP). Consider, for example, the single source, multiple sink case. A bicriteria approximation simply says that each sink is the end of β paths *on average*. It may be that all but one sink is used only once while the last sink is used $k + 1$ times for a total of $2k$ paths. What about harder bounds on the number of times a sink may be used as an endpoint? Rather than saying a sink is used on average β times, what if we said each sink can be used *at most* β times. So, for the case $\beta = 2$ as was explored in Sections 3.3 and 3.4 this means each sink can be used at most twice in single source, multiple sink instances. Is there still such a bicriteria approximation algorithm with a logarithmic (or even polylogarithmic) bound on the approximation ratio α?

The approximation algorithm for Minimum Latency in asymmetric metrics lost an $O(\log n)$ factor essentially for two reasons, one is the rounding of the LP for ATSPP and the other is because $O(\log n)$ buckets were used when grouping the nodes by their latency. Care was taken to ensure the

total loss was only additive in these two ratios. As we mentioned at the end of Chapter 4, it might seem natural to assume that an improved bound on the integrality gaps of ATSPP and k-ATSPP would imply a similar improvement for the approximability of Minimum latency. However, the fact that we use $O(\log n)$ buckets prevents such an improvement to ATSPP from immediately implying the same improvement for Minimum Latency.

Two of the problems related to ATSPP covered on in this thesis are Minimum Latency and k-ATSPP. A natural combination of these variants is the k-Traveling Repairmen problem in asymmetric metrics, the problem of finding k paths starting at a fixed node s whose union covers all other nodes exactly once while minimizing the average distance from s to a node in V along these paths. As mentioned before, this has already been studied in symmetric metrics and constant factor approximations are known. It is easy to construct asymmetric metrics where the cost with $k = 1$ is exponentially larger than the cost with $k = 2$ so it may be of practical importance to consider using more than one repairman. The first question is whether this variant admits a good approximation algorithm/integrality gap. If so, what can be said if the k repairmen all start at different locations (with endpoints still not fixed). One would hope that our bounds for k-ATSPP can be applied to the k-Traveling Repairmen problem in a way that is similar to applying ATSPP to Directed Latency, but there are some technical issues. For example Lemma 4.2.1 does not translate directly to the setting with multiple salesmen and it seems difficult to develop meaningful ordering constraints for $k \geq 2$ paths.

Finally, we remark on the General k-TSPP problem in symmetric metrics. We exhibited a 3-approximation that used simple lower bounds, one being similar to the spanning tree lower bound for classic TSP and the other being the simple fact that every $s_i - t_i$ path has cost at least the cost of the edge $s_i t_i$. A glaring omission is the application of any lower-bounds based on minimum cost matchings as in Christofide's algorithm for TSP or Hoogeveen's algorithm for TSP Path. For example, after finding the S-rooted spanning forest and adding the $s_i t_i$ paths, perhaps we can augment the graph to have an Eulerian $s_i t_i$ path for every component using matchings rather than doubling the edges in the forest. The problem with this approach is that a particular component we are trying to augment may contain an odd number of wrong-degree nodes from different paths in some optimum solution, making it difficult to bound the cost of the matchings using edges from the optimum solution. Still, it might be possible to find a c-approximation for some constant $c < 3$ for General k-TSPP in symmetric metrics.

Bibliography

[1] N. Aggarwal, N. Garg, and S. Gupta, *A 4/3-approximation for TSP on cubic 3-edge-connected graphs*, arXiv:1101.5586, available online at `http://arxiv.org/pdf/1101.5586`.

[2] N. Alon, *Ranking tournaments*, SIAM J. Discrete Math, 20(1):137–142, 2006.

[3] H. An and D. B. Shmoys, *LP-based approximation algorithms for traveling salesman path problems*, arXiv:1105.2391, available online at `http://arxiv.org/pdf/1105.2391`.

[4] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang, *Hardness of the undirected edge-disjoint paths problem with congestion*, In Proceedings of FOCS, 2005

[5] A. Archer, A. Levin, and D. P. Williamson, *A faster, better approximation algorithm for the minimum latency problem*, SIAM J. Comput., 37(5):1472-1498, 2008.

[6] S. Arora, *Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems*, Journal of the ACM, 45:753–782, 1998.

[7] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, Journal of the ACM, 45(3):501-555, 1998.

[8] S. Arora and S. Safra. *Probabilistic checking of proofs: A new characterization of NP*, Journal of the ACM, 45(1):70–122, 1998

[9] A. Asadpour, M. X. Goemans, A. Madry, S. Oveis Gharan, and A. Saberi, *An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem*, In Proceedings of SODA, 2010.

[10] G. Ausiello, P. Crescenzi, V. Kann, Marchetti-Spaccamela, Giorgio Gambosi, and Alberto M. Spaccamela, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer, 2003.

[11] Y. Azar and O. Regev, *Strongly polynomial algorithms for the unsplittable flow problem*, In Proceedings of IPCO, 2001.

[12] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber, *A quasi-PTAS for unsplittable flow on line graphs*, In Proceedings of STOC, 2006.

[13] N. Bansal, Z. Friggstad, R. Khandekar, and M. R. Salavatipour, *A logarithmic approximation for unsplittable flow on line graphs*, In Proceedings of SODA, 2009.

[14] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, B. Schieber, *A unified approach to approximating resource allocation and scheduling*, J. ACM 48(5):1069–1090, 2001.

[15] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability - towards tight results*, SIAM J. on Computing, 27:804–915, 1998.

[16] P. Berman and M. Karpinski, *8/7-approximation algorithm for (1,2)-TSP*, In Proceedings of SODA, 2006.

[17] M. Bläser, *A new approximation algorithm for the asymmetric TSP with triangle inequality*, In Proceedings SODA, 2002.

[18] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, In Proceedings of STOC, 1994.

[19] P. Bonsma, J. Schulz, and A. Wiese, *A constant factor approximation algorithm for unsplittable flow on paths*, arXiv:1102.3643, available online at `http://arxiv.org/abs/1102.3643`.

[20] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, *Improved approximation algorithms for resource allocation*, In Proceedings of IPCO, 2001.

[21] G. Carpaneto, S. Martello and P. Toth, *An algorithm for the bottleneck travelling salesman problem*, Operations Research, 32(2):380389, 1984.

[22] J. O. Cerdeira, *Matroids and a forest cover problem*, Mathematical Programming, 66:403–405, 1994.

[23] A. Chakrabarti, C. Chekuri, A. Kumar, and A. Gupta, *Approximation algorithms for the unsplittable flow problem*, Algorithmica, 47(1):53–78, 2007. Preliminary version in APPROX, September 2002.

[24] M. Charikar, M. X. Goemans, and H. Karloff, *On the integrality ratio for asymmetric TSP*, In Proceedings of FOCS, 2004.

[25] K. Chaudhuri, B. Godfrey, S. Rao and K. Talwar, *Paths, trees, and minimum latency tours*, In Proceedings FOCS, 2003.

[26] C. Chekuri, A. Ene, and N. Korula, *UFP in paths and trees and column-restricted packing integer programs*, In Proceedings of APPROX, 2009.

[27] C. Chekuri and A. Kumar, *Maximum coverage problem with group budget constraints*, In Proceedings of APPROX, 2004.

[28] C. Chekuri, S. Khanna, and B. Shepherd, *An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and UFP*, Theory of Computing, 2:137–146, 2006.

[29] C. Chekuri, M. Mydlarz, and F. B. Shepherd, *Multicommodity demand flow in a tree and packing integer programs*, ACM Transactions on Algorithms, 3(3), 2007

[30] C. Chekuri and M. Päl, *An $O(logn)$ approximation ratio for the asymmetric travelling salesman path problem*, In Proceedings of APPROX, 2006.

[31] C. Chekuri, J. Vondrák, and R. Zenklusen, *Dependent randomized rounding via exchange properties of combinatorial structures*, In Proceedings of FOCS, 2010.

[32] N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[33] M. Chudnovsky, N. Robertson, P. Seymour, R. Thomas, *The strong perfect graph theorem*, Annals of Mathematics 164(1):51-229, 2006.

[34] J. Chuzhoy and P. Chalermsook, *The maximum independent set of rectangles problem*, In Proceedings of SODA, 2009.

[35] V. Chvátal, *A greedy heuristic for the set-covering problem*, Mathematics of Operations Research, 4:233–235, 1979.

[36] J. Fakcharoenphol, C. Harrelson, and S. Rao, The k-traveling repairman problem, In Proceedings of SODA, 2003.

[37] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, Journal of the ACM, 45:634–652, 1998.

[38] U. Feige and M. Singh, *Improved approximation ratios for traveling salesman tours and paths in directed graphs*, In Proceedings of APPROX, 2007.

[39] S. Fortune, J. E. Hopcroft, and J. Wylie, *The directed subgraph homeomorphism problem*, Theor. Comput. Sci., 10:111–121, 1980.

[40] A. Frank, *On connectivity properties of Eulerian digraphs*, Annals of Discrete Mathematics, 41:179–194, 1989.

[41] A. Frieze, *Edge disjoint paths in expander graphs*, SIAM Journal on Computing, 30:1790–1801, 2001.

[42] A. Frieze, G. Galbiati and F. Maffioli, *On the worst-case performance of some algorithms for the asymmetric traveling salesman problem*, Networks, 12:23–39, 1982.

[43] Z. Friggstad, M. R. Salavatipour, and Z. Svitkina, *Asymmetric traveling salesman path and directed latency problems*, In Proceedings of SODA, 2010.

[44] I. Gamzu and D. Segev, *A sublogarithmic approximation for highway and tollbooth pricing*, In Proceedings of ICALP, 2010.

[45] M. R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.

[46] N. Garg, V. V. Vazirani, and M. Yannakakis, *Primal-dual approximation algorithms for integral ow and multicut in trees*, Algorithmica, 18(1):3-20, 1997.

[47] M. X. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Math. Program., 82:111-124, 1998.

[48] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57), North-Holland Publishing Co., Amsterdam, The Netherlands, second edition, 2004.

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

[50] M. Grötschel, L. Lovász, and A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer-Verlag, 1988.

[51] V. Guruswami, S. Khanna, B. Shepherd, R. Rajaraman, and M. Yannakakis, *Near optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*, J. of Computer and System Sciences, 67(3):4730496, 2003.

[52] M. Held and R. Karp, *The traveling salesman problem and minimum spanning trees*, Operations Research, 18:1138–1162, 1970.

[53] D. S. Hochbaum, *Approximation algorithms for the set covering and vertex cover problems*, SIAM Journal on Computing, 11:555–556, 1982.

[54] J. A. Hoogeveen, *Analysis of Christodes heuristic: some paths are more difcult than cycles*, Operations Research Letters, 10(5):291-295, 1991.

[55] O. H. Ibarra and C. E. Kim, *Fast approximation algorithms for the knapsack and sum of subset problems*, Journal of the ACM, 22:463–468, 1975.

[56] B. Jackson, *Some remarks on arc-connectivity, vertex splitting, and orientation in digraphs*, Journal of Graph Theory, 12(3):429–436, 1988.

[57] R. Jothi and B. Raghavachari, *Approximating the k-traveling repairman problem with repair-times*, Journal of Discrete Algorithms, 5(2):293–303, 2004.

[58] H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko, *Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs* J. ACM, 52(4):602-626, 2005.

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

[60] R. Karp, *Reducibility among combinatorial problems*, In Raymond E. Miller and James W. Thatcher, editors *Complexity of Computer Computations*, 85–103, Plenum Press, 1972.

[61] J. M. Kleinberg, Approximation Algorithms for Disjoint Paths Problems, PhD thesis, MIT, 1996.

[62] F. Lam and A. Newman, *Traveling salesman path problems*, Mathematical Programming, 113(1):39–59, 2008.

[63] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The traveling salesman problem*, John Wiley, 1985.

[64] L. Lewin-Eytan, J. Naor, and A. Orda, *Routing and admission control in networks with advance reservations*, In Proceedings of APPROX, 2002.

[65] W. Malik, S. Rathinam, S. Darbha, *An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem*, Operations Research Letters 35(6):747-753, 2007.

[66] C. Mathieu and W. Schudy, *How to rank with few errors: a PTAS for weighted feedback arc set on tournaments*, In Proceedings of STOC, 2007.

[67] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. on Comput., 28:1298–1309, 1999.

[68] M. Mitzenmacher and E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, Cambridge, UK, 2005.

[69] T. Mömke and O. Svensson, *Approximate graphic TSP by matchings*, arXiv:1104.3090, available online at `http://arxiv.org/pdf/1104.3090`.

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

[71] S. Oveis Gharan, A. Saberi, *Asymmetric traveling salesman problem on graphs with bounded genus*, In Proceedings of SODA, 2011

[72] S. Oveis Gharan, A. Saberi, and M. Singh, *A randomized rounding approach to the traveling salesman problem*, Manuscript, Available online at `http://www.cs.mcgill.ca/~mohit/publications.html`.

[73] C. H. Papadimitriou and S. Vempala, *On the approximability of the traveling salesman problem*, Combinatorica, 26(1):101–120, 2006.

[74] C. H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Math. Oper. Res., 18:1–11, 1993.

[75] A. Phillips, R. N. Uma and J. Wein, *Online admission control for general scheduling problems*, Journal of Scheduling, 3(6):365–381, 2000.

[76] S. Rathinam and R. Sengupta, *Matroid intersection and its application to a multiple depot, multiple TSP*, Technical report, University of California, Berkely, 2006.

[77] S. Rathinam and R. Sengupta, *3/2-approximation algorithm for a generalized, multiple depot, Hamiltonina path problem* Technical report, University of California, Berkeley, 2007.

[78] S. Rathinam, R. Sengupta, *5/3-approximation algorithm for a multiple depot, terminal Hamiltonian path problem* Technical report, University of California, Berkeley, 2007.

[79] S. Rathinam, R. Sengupta, *3/2-approximation algorithm for two variants of a 2-depot Hamiltonian path problem*, Operations Research Letters 38(1):63-68, 2010.

[80] S. Rathinam, R. Sengupta, and S. Darbha, *A resource allocation algorithm for multi-vehicle systems with non holonomic constraints*, IEEE Transactions on Automation Sciences and Engineering, 4(1):98-104, 2006.

[81] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas, *Efficiently four-coloring planar graphs*, In Proceedings of STOC, 1996.

[82] N. Robertson N, P. D. Seymour, *Graph minors XIII: the disjoint paths problem*, Journal of Combinatorial Theory, Series B, 63(1):65–110, 1995.

[83] B. Rodrigues and Z. Xu, *A 3/2-approximation for multiple depot multiple traveling salesman problem*, In Proceedings of SWAT, 2010.

[84] A. Schrijver, Combinatorial Optimization - Polyhedra and Efficiency, Springer-Verlag, New York, 2005.

[85] D. B. Shmoys and D. P. Williamson, *Analyzing the Held-Karp TSP bound: a monotonicity property with application*, Information Processing Letters, 35(6):281-285, 1990.

[86] J. M. S. Simes-Pereira, *On subgraphs as matroid cells*, Mathematische Zeitschrift, 127:315–322, 1972.

[87] A. Srinivasan, *Improved approximations for edge-disjoint paths, unsplittable ow, and related routing problems*, In Proceedings of FOCS, 1997.

[88] V. Vazirani, Approximation Algorithms, Springer, 2003.

[89] D. P. Williamson, *Analysis of the Held-Karp heuristic for the traveling salesman problem*, M.Sc. Thesis, Masachusetts Institute of Technology, 1990.

[90] L. A. Wolsey, *Heuristic analysis, linear programming and branch and bound*, In Combinatorial Optimization II, volume 13 of Mathematical Programming Studies, pages 121134. Springer Berlin Heidelberg, 1980.

[91] D. Zuckerman, *Linear degree extractors and the inapproximability of max clique and chromatic number*, Theory of Computing, Volume 3 (2007), pp. 103128