Capacitated Network Design on Outerplanar Graphs

by

Ishan Bansal

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2020

Author's Declaration

Abstract

Network design problems model the efficient allocation of resources like routers, optical fibres, roads, canals etc. to effectively construct and operate critical infrastructures. In this thesis, we consider the capacitated network design problem (CapNDP), which finds applications in supply-chain logistics problems and network security. Here, we are given a network and for each edge in the network, several security reinforcement options. In addition, for each pair of nodes in the network, there is a specified level of protection demanded. The objective is to select a minimum-cost set of reinforcements for all the edges so that an adversary with strength less than the protection level of a particular pair of nodes cannot disconnect these nodes. Several special cases of CapNDP are by themselves *NP*-hard and *APX*-hard to approximate. Hence, researchers have attempted to find approximation algorithms for the problem by adding constraints on the structure of the network. In this thesis, we investigate CapNDP when the network structure is constrained to belong to a class of graphs called outerplanar graphs. This particular special case was first considered by Carr, Fleischer, Leung and Philips; while they claimed to obtain an FPTAS here, their algorithm has certain fatal flaws. We build upon some of the ideas they use to approximate CapNDP on general networks to develop a new algorithm for CapNDP on outerplanar graphs. Prior to our work, the best known approximation ratio of an approximation algorithm here was $O(n)$ where n is the number of nodes in the outerplanar graph. Our main result provides an approximation ratio that is improved by a doubly exponential factor giving an $O((\log\log n)^2)$-approximation algorithm for CapNDP on outerplanar graphs. We also notice that our methods can be applied to a more general class of problems called column-restricted covering integers programs, and be adapted to improve the approximation ratio on more instances of CapNDP if the structure of the network is suitable. Furthermore, our techniques also yield interesting results for a completely unrelated problem in the area of data structures.

Acknowledgements

I am grateful to my parents, my brothers and my family for their unwavering love and support

I am thankful to my supervisors Professor Chaitanya Swamy and Professor Jochen Koenemann for their continuous and thorough guidance throughout my graduate studies. They encouraged me to pursue my research topic and to develop my research and reading skills. During our weekly meetings, they patiently listened to my ideas and prodded me along the right direction whenever there came an obstacle. Their detailed feedback on my thesis has also improved my mathematical writing skills. Leading by example, they inspired me to pursue a doctorate degree and I will always be grateful.

I would like to thank Professor Joseph Cheriyan and Professor Ricardo Fukasawa for reading my thesis and providing valuable feedback.

I am grateful to Sharat for constantly guiding and mentoring me like an elder brother. Also to my friends from CLV for always providing such warm company and joyful times. I would also like to thank uncle Anil and aunty Anju for providing the most homely environment I could have asked for.

Thanks to Melissa to whom I could reach out unhesitatingly with any administrative formalities. Also to the entire C&O staff and faculty for making my graduate experience a memorable one.

Lastly, I would like to thank my spiritual guide, Mr. Kamlesh Patel for his constant presence.

Dedication

To the mind's thirst for challenges

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CapNDP Capacitated Network Design Problem xi, , 1–5, 15–18, 20–22, 26, 27, 29, 30, 34, 39, 41–44, 48, 51, 54, 55, 60, 61

FPTAS Fully Polynomial Time Approximation Scheme , 3, 8, 21, 61

IP Integer Program

KC Knapsack Cover

LP Linear Program

SNDP Survivable Network Design Problem , 2

List of Algorithms

Chapter 1

Introduction

Governments today have identified several assets that are essential for the functioning of a society and an economy. The disruption of these assets would lead to immediate and catastrophic results and they have been termed as critical infrastructures. Many of these infrastructures are physical networks like transportation, water, natural gas, telecommunication, post etc. Network design problems model the efficient allocation of resources like routers, optical fibres, roads, canals etc. to effectively construct and operate these physical infrastructures. In this thesis, we consider the *CapNDP*, which finds applications in supply-chain logistics problems and network security. Here, we are given a network or a multigraph $G = (V, E)$ along with capacities $\{u(e)\}_{e \in E}$ and costs $\{c(e)\}_{e \in E}$ for each edge and non-negative demands D_{ij} for every pair of nodes (i, j), we wish to find a minimum cost subgraph H such that for each pair of nodes (i, j), H admits a flow of value D_{ij} between i and j.

As Carr *et al.* [6] observed, the capacitated network design problem arises as a network reinforcement problem. Here we are given an existing network, and for each edge in the network several security reinforcement options. In addition, for each pair of nodes in the network there is a specified level of protection demanded. The objective is to select a minimum-cost set of reinforcements for all the edges so that an adversary with strength less than the protection level of a particular pair of nodes cannot disconnect these nodes.

We would like to point out that a different and somewhat related problem is also referred to by the same name. In this version one wishes to design a network with enough capacity to route all the demands between the nodes simultaneously. This is more closely related to buy-at-bulk network design [11, 1, 25] and the fixed charge network flow [19] problems.

1.1 Related Work

The capacitated network design problem generalizes the classical *SNDP* where the capacity of each edge is 1. SNDP already captures several connectivity problems in combinatorial optimization like the min-cost Steiner tree and the min-cost λ-edge-connected subgraph problems. CapNDP also captures source and facility location problems [9, 18]. Several of these special cases are by themselves *NP*-hard and *APX*-hard to approximate. Jain [22] obtained a 2-approximation algorithm for SNDP via the standard cut-based LP relaxation using the iterated rounding technique. Chakrabarty *et al.* [7] considered the version of capacitated network design where multiple copies of the same edge can be picked and obtained an $O(\log p)$-approximation algorithm where p is the number of pairs of nodes with positive demand. They also showed that it is $\Omega(\log \log n)$-hard to approximate CapNDP when multiple copies of an edge can be picked where n is the number of nodes in the graph G.

The version of CapNDP that we are interested in was introduced by Goemans *et al.* [16] and they made several observations about the problem including: *(i)* CapNDP reduces to SNDP if all the capacities are the same, *(ii)* There is an $O(\min(M, D_{max}))$-approximation algorithm where M is the number of edges in the graph G and $D_{max} = \max_{i,j} D_{ij}$ is the maximum demand between pairs of nodes. Carr *et al.* [6] introduced the Knapsack Cover inequalities to strengthen the LP relaxation of CapNDP and used the Bucketing Algorithm described in Chapter 2 to obtain a $(\beta(G) + 1)$-approximation algorithm for general graphs where $\beta(G)$ is the maximum size of a bond. A *bond* is a minimal set of edges that separates a pair of nodes with positive demand in the underlying simple graph. For most graphs, $\beta(G)$ is in $\Theta(m)$ where m is the number of edges in the graph G and this is currently the best algorithm to tackle the problem on general graphs.

We now move on to some hardness results for the problem. Even *et al.* [15] showed that in directed graphs, the CapNDP cannot be approximated to a factor better than $2^{\log^{1-\delta} n}$ for any $\delta < 1$ unless $NP \subseteq DTIME(n^{\text{polylog}n})$ and subsequently Chakrabarty *et al.* [9] proved the same result for undirected graphs too. Both these results are via the label cover problem and are true even when we restrict to instances of CapNDP with just one demand pair.

Due to these hardness results, it makes sense to look at special instances of CapNDP. Chakrabarty *et al.* [7] observed that one could use the KC inequalities, randomized rounding and Chernoff bounds in the case where we have uniform demands *i.e.* $D_{ij} = D$ for every pair of nodes (i, j) to obtain an $O(\log n)$-approximation algorithm for CapNDP. Krumke *et al.* [23] considered the case where the underlying graph G is series-parallel and the pair of terminals of the series-parallel graph is the only pair of nodes with positive demand. Here, they described a pseudo-polynomial algorithm using dynamic programming that runs in time $O(m^3C^2)$ where m is the number of edges in G and C is the maximum cost of an edge in G. Notice that the running time of the dynamic program depends only on the cost of the edges and not on the demand between the terminal nodes. Hence, they were able to scale down the cost of each edge by a suitable factor (and apply the ceiling function to maintain integrality) so that the maximum cost of an edge after scaling is polynomial in the size of the problem input. Doing so, allows one to obtain an optimal solution to the cost-scaled problem in polynomial time and this solution is feasible for the original problem as well since the demand between the terminal nodes and the capacities of the edges was not changed. Depending on the scaling factor chosen, the cost of this solution is close to the optimum cost of the original problem and Krumke *et al.* [23] were able to obtain an FPTAS for CapNDP on series-parallel graphs when the terminals of the series-parallel graph are the only pair of nodes with positive demand. Carr *et al.* [6] obtained approximation factors of 2 and 3 for line graphs and circle graphs respectively. They also described a pseudo-polynomial time algorithm via dynamic programming for instances of CapNDP where the underlying graph G is outerplanar and only one pair of vertices has a positive demand D between them. An outerplanar graph is a graph that admits a planar embedding such that every node lies on the outer face. The running time of their dynamic program is $O(mD^3)$ where m is the number of edges in G. Carr *et al.* [6] claim that this DP can be utilized to obtain an FPTAS via scaling techniques used by Krumke *et al.* [23], but this argument is erroneous and fatally flawed since the running time of this dynamic program depends on the demand D. The issue is that one would have to scale down the demand D and capacities of the edges (and apply the ceiling function to maintain integrality) in order to obtain a solution to the scaled-down problem in polynomial time. However since the demand D and the capacities of the edges were changed, this solution need not be feasible for the original problem. Hence until the results of this thesis, the best approximation ratio of an approximation algorithm for CapNDP on outerplanar graphs was $O(\beta(G))$. For outerplanar graphs, $\beta(G)$ is in $\Theta(n)$ and so the best approximation ratio of an approximation algorithm for CapNDP on outerplanar graphs was $O(n)$ [1].

1.2 Our Contributions and Outline of Thesis

In this thesis, we design an approximation algorithm with a better approximation ratio for CapNDP on outerplanar graphs. The $(\beta(G)+1)$-approximation algorithm provided by Carr *et al.* [6] begins with a crucial Bucketing Algorithm which generates for disjoint edge sets of G, a collection of candidate subsets of these edge sets to include in a final solution. These candidate subsets are then arbitrarily "merged" from the different disjoint edge sets to obtain a final solution as seen in Algorithm 2. Our idea is that "merging" the candidate subsets in a particular way depending on the structure of the bonds in the underlying graph can lead to better results. We use this idea to provide a 2-approximation algorithm for the single demand case in another class of graphs which we will call non-crossing interval graphs. These graphs generalize line graphs and are building blocks in the structure of outerplanar graphs. We then improve the approximation factor for instances of CapNDP on outerplanar graphs by a doubly exponential factor. This is done by imposing certain conditions on the demand pairs that are not too restrictive. These conditions capture the single demand pair case and we obtain an $O((\log\log n)^2)-$approximation algorithm. We also show how our results can be extended to work even if the graph is directed.

Chapter 2 begins with a discussion on the KC inequalities as described by Carr *et al.*

[1] However, we observe that the randomized rounding algorithm of [12] yields an $O(\log n)$-approximation algorithm for CapNDP on outerplanar graphs (see Section 5.1)

[6]. We describe their bucketing algorithm and see how it was used by them to obtain a $(\beta(G)+1)$-approximation algorithm for CapNDP on general graphs. This will provide a starting point for our work.

Chapter 3 begins with a discussion on the structure of outerplanar graphs. One of the two main ingredients of our $O((\log \log n)^2)-$approximation algorithm, the Merging Algorithm is described next and we see how it can be used to obtain a 2-approximation algorithm for non-crossing interval graphs.

Chapter 4 deals with the other main ingredient of our algorithm which is a combinatorial problem we call the Exact Range Cover Problem. We also find that our results here have applications in the well-studied Array Range Query Problem. In this problem, we are given n entries $a_1, a_2, \ldots, a_n$ each coming from a semi-group. The user is allowed to update the entries and also query the product of any range $[i, j]$ which is just $a_i \cdot a_{i+1} \cdot \ldots \cdot a_j$. One wishes to design a data structure that efficiently allows these two operations. A survey of results for this problem of querying the product of a range can be found in [26]. We describe a family of data structures for this problem and are able to obtain for any fixed constant c, a data structure that has $O(n \log \log n)$ space complexity, $O(n^{1/c})$ update time and $O(\log \log n)$ query time.

Chapter 5 is devoted to extensions of our results. We first show how we can relax certain conditions that we had imposed on the demand pairs by re-using our ideas. We also show how our ideas can be used in the setting of directed outerplanar graphs. Finally we consider general column-restricted covering integer programs (CCIP). These are integer programs of the type $\min\{c^T x : Ax \geq b, x \in \{0,1\}\}$ where each column of A is restricted by the property that every non-zero entry of that column is the same. CCIPs generalize $\{0,1\}$-covering integer programs and hence capture a wide variety of hard problems. Chakrabarty, Grant and Könemann [8] showed that if the underlying $\{0,1\}$-CIP has an integrality gap $O(\gamma)$ and its priority version has an integrality gap $O(\omega)$, there exists an $O(\gamma+\omega)$ approximation algorithm for the CCIP. Subsequently Chan, Grant, Könemann and Sharpe [10] built on these results and discovered an $O(1)$-approximation algorithm in the case where the constraint matrix A is a network matrix. This covers the case for example when the support of each column of A is a consecutive set of rows. Carr *et al.* [6] used their bucketing algorithm 1 to provide a p-approximation algorithm for general capacitated covering integer programs where p is the maximum number of non-zero entries in a row of the constraint matrix A. Using our techniques, we show that if the support of each row of A is a set of at most k sets of consecutive columns, then we can obtain an $O(k(\log\log n)^2)$-approximation algorithm for the CCIP.

Chapter 2

Preliminaries

In this chapter, we give a brief introduction to the Capacitated Network Design Problem and lay the groundwork for further discussions. We describe in detail the Bucketing Algorithm of Carr *et al.* [6] as the algorithm will be used as a building block for our results. We begin by describing the minimum knapsack problem and the knapsack cover inequalities.

2.1 Minimum Knapsack Problem and Knapsack Cover Inequalities

The *minimum knapsack problem* is the minimization version of the well-known *NP*-complete knapsack problem. The problem statement is quite simple to describe,

Problem 1 (Minimum Knapsack Problem)**.** *Given a set of items E along with non-negative integral capacities $\{u(e)\}_{e\in E}$ and costs $\{c(e)\}_{e\in E}$ for each item and a demand D. We wish to find a minimum cost subset of the items whose total capacity is at least the demand D.*

It is not difficult to see that the *NP*-complete knapsack problem is poly-time reducible to the minimum knapsack problem and so the decision version of the minimum knapsack problem is also *NP*-complete. Also, the FPTAS for the knapsack problem [21] can be easily modified to provide an FPTAS for the minimum knapsack problem. We are, however, more interested in approximation algorithms for the minimum knapsack problem via its integer programming formulation as those methods easily generalize to the capacitated network design problem. The rest of this chapter is based on work by Carr, Fleischer, Leung, and Phillips [6]. We shall first formulate the minimum knapsack problem as an integer program,

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e)z(e) && \text{(MinKP-IP)} \\ \text{s.t.} \quad & \sum_{e \in E} u(e)z(e) \geq D && \text{(covering constraints)} \\ & z(e) \in \{0,1\} \qquad \forall \quad e \in E \end{aligned}$$

Here $z(e)$ is 1 or 0 according to whether the item e is part of our solution or not. We can now construct linear relaxations of MinKP-IP and round an optimal fractional solution to obtain a "good" integral solution. Thus consider the following simple LP relaxation of MinKP-IP,

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e)x(e) && \text{(MinKP-LP)} \\ \text{s.t.} \quad & \sum_{e \in E} u(e)x(e) \geq D \\ & x(e) \in [0,1] \qquad \forall \quad e \in E \end{aligned}$$

A very useful parameter that helps decide the strength of a particular LP relaxation when trying to design approximation algorithms is the integrality gap which we now define,

Definition 2.1.1 (Integrality Gap)**.** Given an integer program IP and a linear relaxation of the integer program LP, the integrality gap of LP is defined as the maximum over all instances of the ratio Opt_{IP}/Opt_{LP} where Opt_{IP} and Opt_{LP} are the optimal values of IP and LP on a particular instance respectively.

Remark 1. It should be clear that no algorithm that compares the cost of its solution to the optimum value of LP can achieve an approximation factor better than the integrality gap of LP.

The integrality gap of MinKP-LP is large as can be seen from the following instance of the minimum knapsack problem.

Example. *Consider a set with two items $E = \{e_1, e_2\}$. Let $u(e_1) = D - 1$, $c(e_1) = 0$, $u(e_2) = D$ and $c(e_2) = 1$ and let the demand be D. Any feasible integer solution must include the item e_2 and hence $Opt_{MinKP-IP}$ is 1. However the optimal solution for MinKP-LP is $x(e_1) = 1$ and $x(e_2) = 1/D$ and so $Opt_{MinKP-LP}$ is $1/D$ giving an integrality gap of at least D.*

In situations like this when a particular linear relaxation turns out to be unfruitful, it is natural to consider tightening the linear program by adding more constraints that are valid for the integer program. To this end, consider any subset of the items $A \subseteq E$ and define the capacity of A as $u(A) := \sum_{e \in A} u(e)$. Suppose we select every item of A, then the remaining items must meet a residual demand given by $D(A) := \max\{D - u(A), 0\}$. Additionally, since we are interested in integer solutions, we can assume that the capacity of each remaining item is at most this residual demand. Thus define for all $e \notin A$, the reduced capacity $u_A(e) = \min\{u(e), D(A)\}$ and we get the following constraints that are valid for MinKP-IP known as the knapsack cover (KC) inequalities,

$$\sum_{e \notin A} u_A(e)x(e) \geq D(A) \qquad \forall \quad A \subseteq E \qquad \text{(KC inequalities)}$$

This capacitated version of the knapsack cover inequalities was introduced by Carr *et al.* [6] to strengthen the linear programming relaxations of the minimum knapsack problem, the capacitated network design problem, the generalized vertex cover problem and the capacitated covering problem. Prior to this, researchers [2, 27] considered an uncapacitated form of the KC inequalities showing that they are facet defining under certain conditions. This idea of picking a partial solution and reducing the values of the remaining variables accordingly has now been used in a variety of complicated problems. For example, Carnes and Shmoys [5] used them in the single-demand facility location problem and also used them to obtain a primal-dual 2-approximation algorithm for the minimum knapsack problem [5]. Chakrabarty, Grant and Könemann [8] used them to obtain approximation algorithms for column-restricted covering integer programs. Bansal, Gupta and Krishnaswamy [3] used them in the generalized min-sum set cover problem and Cheung, Mestre, Shmoys and Verschae [13] used them in the generalized min-sum scheduling problem. Other instances of the knapsack cover inequalities can be found in [14, 4, 24, 17].

We now add the KC inequalities to MinKP-LP to obtain the following tighter linear programming relaxation of MinKP-IP,

$$\begin{aligned} \min \quad & \sum_{e\in E} c(e)x(e) & & \text{(MinKP-KCLP)} \\ \text{s.t.} \quad & \sum_{e\notin A} u_A(e)x(e) \geq D(A) \qquad \forall \quad A \subseteq E & & \text{(KC inequalities)} \\ & x(e) \geq 0 \qquad \forall \quad e \in E & & \end{aligned}$$

Note that we have dropped the constraints $x(e) \leq 1$, because it turns out that they are redundant in the following sense: For any item e', if x is feasible to MinKP-KCLP, then so is the vector x' defined as $x'(e') = \min\{1, x(e')\}$ and $x'(e) = x(e)$ for every other item. This is because if $e' \in A$, then x' clearly satisfies the KC inequality for A. Else $e' \notin A$ and let $A' = A \cup \{e'\}$. Then $\sum_{e\notin A} u_A(e)x'(e) = u_A(e') + \sum_{e\notin A'} u_A(e)x(e) \geq u_A(e') + \sum_{e\notin A'} u_{A'}(e)x(e) \geq u_A(e') + D(A') \geq D(A)$. The last inequality follows since $u_A(e')$ is either $u(e)$ or $D(A)$. Hence setting $x(e') = \min\{1, x(e')\}$ will still be feasible and the cost of the vector after this change cannot increase.

Let us look at the structural strength of these inequalities by considering the example 2.1 above. The rational point that was optimal for MinKP-LP is now cut-off.

Example. *Consider a set with two items $E = \{e_1, e_2\}$. Let $u(e_1) = D - 1$, $c(e_1) = 0$, $u(e_2) = D$ and $c(e_2) = 1$ and let the demand be D. Any feasible integer solution must include the item e_2 and hence $Opt_{MinKP-IP}$ is 1. Now, set $A = \{e_1\}$. Then $D(A) = 1$ and so $u_A(e_2) = 1$. The KC inequalities now force $x(e_2) = 1$ so that $Opt_{MinKP-KCLP}$ is also 1.*

Carr *et al.* [6] provided an algorithm that rounds a feasible fractional solution of MinKP-KCLP to a feasible integer solution of MinKP-IP while at most doubling the cost of the solution. We shall now describe the general framework of this algorithm as it will be required while designing our algorithm for the capacitated network design problem.

2.1.1 Bucketing Algorithm

The algorithm takes as input a fractional (rational) vector $x \in [0,1]^E$ and an integral parameter $\alpha > 1$. Note that an integer vector $z \in \{0,1\}^E$ can be viewed as a subset of E and vice versa and we shall use these notions interchangeably. Carr *et al.* [6] referred to these subsets as buckets and it may be helpful to think of these subsets that way. Let r be the least common multiple of the denominators of x. The algorithm returns r integer vectors $z^1, z^2, \ldots, z^r$, each in $\{0,1\}^E$ as follows.

```
Algorithm 1: Bucketing Algorithm
    Input: An instance $E, \{u(e)\}_{e\in E}, \{c(e)\}_{e\in E}, D$ of minimum knapsack, a rational
      vector $x \in [0,1]^E$ and an integral parameter $\alpha > 1$. The demand $D$ and $\{c(e)\}_{e\in E}$
      may be omitted in the input.
    Setup: Let $r$ be the least common multiple of the denominators of $x$ and let
      $A^{x,\alpha} = A := \{e \in E : x(e) \geq 1/\alpha\}$. Let $e_1, e_2, \ldots, e_k$ be an enumeration of the
      items in $E \backslash A$ in order of non-increasing $u(e)$ values.
    Step 1: For each item $e \in A$, set $z^i(e) = 1$ for all $i$.
    Step 2: Take $r\alpha x(e_1)$ copies of the item $e_1$ and put them into the first $r\alpha x(e_1)$
      subsets of $E$(vectors $z^1, z^2, \ldots, z^{r\alpha x(e_1)}$). Then take $r\alpha x(e_2)$ copies of the item $e_2$
      and put them into the next $r\alpha x(e_2)$ subsets of $E$ cyclically i.e. after placing in
      the last subset of $E$(vector $z^r$, continue from the first subset(vector $z^1$). Do this
      till the last item $e_k$ of $E \backslash A$.
```

Remark 2. The least common multiple r is not necessarily polynomial in the size of the input. However, the number of distinct subsets created by Algorithm 1 above is at most $|E| + 1$ and so, the algorithm can be implemented in polynomial time as follows: for each item e, let z^{s_e} (z^{t_e}) be the first (last) subset of E where item e is placed. These can be calculated using the recursion $t_{e_i} = s_{e_i} + r\alpha x(e_i) - 1 \pmod r$, $s_{e_{i+1}} = t_{e_i} + 1 \pmod r$ and

$s_{e_1} = 1$. Here, 0 (mod r) is identified with r (mod r) so that each s_e (t_e) is an integer between 1 and r. Let $t_1, t_2, \ldots, t_k$ be an enumeration of the $t_e's$ in non-decreasing order and let $t_0 = 0, t_{k+1} = r$. Then for every $i = 1, \ldots, k+1$, vectors $z^{t_{i-1}+1}$ to z^{t_i} are the same and contain items e such that either (i) $s_e \leq t_{i-1}+1$ and ($t_e \geq t_i$ or $t_e \leq s_e$) or (ii) $s_e \geq t_i$ and ($t_e \leq s_e$ and $t_e \geq t_i$). Thus there are at most $|E|+1$ distinct subsets created and they can be described in polynomial time. Also note that any multiple of r will also work, all we want is for $r\alpha x(e)$ to be integral. We can find at least one such number in polynomial time namely the product of all the denominators of x. All of our algorithms in this thesis will follow a similar description of creating r vectors with the understanding that as in this case, these algorithms can be implemented in polynomial time.

Example (Bucketing Algorithm)**.** *Let us consider a simple example with six elements and* $\alpha = 2$. *Thus* $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$. *Let vector* $x = (1/7, 2/7, 3/7, 4/7, 5/7, 6/7)$ *and let the capacities be* $u = (3, 5, 2, 4, 6, 8)$. *Then,* $r = 7$ *and we will create seven integer vectors* $z^1, z^2, z^3, z^4, z^5, z^6, z^7$. *The set* $A^{x,\alpha} = \{e_4, e_5, e_6\}$ *and so* $z^i(e_j) = 1$ *for all* $i = 1, \ldots, 7$ *and* $j = 4, 5, 6$. *The remaining items of* E *namely* e_1, e_2, e_3 *are sorted in order of non-increasing* $u(e)$ *values giving the order* e_2, e_1, e_3. *Step 2 of the bucketing algorithm starts by taking* $r\alpha x(e) = 7 * 2 * 2/7 = 4$ *copies of item* e_2 *and creating the following vectors.*

	z^1	z^2	z^3	z^4	z^5	z^6	z^7
e_2	1	1	1	1	0	0	0

We then take $r\alpha x(e) = 7 * 2 * 1/7 = 2$ *copies of item* e_1 *and create the following vectors.*

	z^1	z^2	z^3	z^4	z^5	z^6	z^7
e_2	1	1	1	1	0	0	0
e_1	0	0	0	0	1	1	0

Finally, we take $r\alpha x(e) = 7*2*3/7 = 6$ *copies of item* e_3 *and create the following vectors.*

	z^1	z^2	z^3	z^4	z^5	z^6	z^7
e_2	1	1	1	1	0	0	0
e_1	0	0	0	0	1	1	0
e_3	1	1	1	1	1	0	1

These are the vectors that are outputted by the bucketing algorithm

Let us now analyze some properties of the integer vectors $z^1, z^2, \ldots, z^r$ that we obtain using this algorithm.

Firstly note that since $x(e)$ is less than $1/\alpha$ for each item $e \notin A$, $r\alpha x(e) < r$ for each item $e \notin A$. Thus, at most one copy of each item is added to a subset and the integer vectors $z^1, z^2, \ldots, z^r$ are in fact all in $\{0,1\}^E$. Secondly,

- $z^i(e) = 1$ for all i and every $e \in A$.

This follows straight from Step 1 of Algorithm 1.
For any vector x and subset $S \subseteq E$, we shall denote the cost of x on S using the shorthand notation $c(x(S))$ which means $\sum_{e \in S} c(e)x(e)$ and similarly $u_A(x(S)) = \sum_{e \in S} u_A(e)x(e)$. We shall also omit the parameter S if dealing with the entire set E *i.e.* $c(x) = c(x(E))$.

- $\sum_{i=1}^r c(z^i) \leq r\alpha c(x)$

This is because $\sum_{i=1}^r c(z^i) = \sum_{i=1}^r \{c(z^i(A)) + c(z^i(E \backslash A))\}$. Since $\alpha x(e) \geq 1$ *for all* $e \in A$, we have $\sum_{i=1}^r c(z^i(A)) = r\sum_{e \in A} c(e) \leq r\sum_{e \in A} c(e)\alpha x(e) = r\alpha c(x(A))$. Also, due to the number of copies of each item in $E \backslash A$ created by Algorithm 1, $\sum_{i=1}^r c(z^i(E \backslash A)) = r\alpha c(x(E \backslash A))$. Combining these, we get our inequality. This tells us that the integer vector with the smallest cost among the ones obtained through Algorithm 1, $z^1, z^2, \ldots, z^r$ has cost at most $\alpha c(x)$.

- $|u_A(z^i(E \backslash A)) - u_A(z^j(E \backslash A))| \leq D(A)$ for every pair i, j

This is proven as follows. Observe that $e_1, e_2, \ldots, e_k$ is also an enumeration of the items in $E \backslash A$ in order of non-increasing $u_A(e)$ values. Now due to the way Algorithm 1 assigns these items, it is clear that z^1 has the largest u_A-value and z^r the smallest. Let us pair the j^{th} item added to z^r with the $j+1^{th}$ item added to z^1. Again, due to the way Algorithm 1 assigns these items, it is clear that in each pair, the item from z^r has a greater u_A-value than the item from z^1. The only items from z^1 and z^r that may not be paired are the first item added to z^1 and the last item added to z^r but the difference between the

u_A-values of these two items is at most $D(A)$ since $u_A(e) \leq D(A)$ for any item e. Thus $u_A(z^1(E\backslash A)) - u_A(z^r(E\backslash A)) \leq D(A)$ and since we considered the largest and smallest integer vectors, we have proven the result for any pair of vectors. Lastly,

- $\sum_{i=1}^{r} u_A(z^i(E\backslash A)) = r\alpha u_A(x(E\backslash A))$

This follows simply because we have taken $r\alpha x(e)$ copies of each item in $E\backslash A$.

The last two properties immediately tell us that the the integer vector with the smallest u_A-value (*i.e.* z^r) satisfies $u_A(z^r(E\backslash A)) \geq \alpha u_A(x(E\backslash A)) - D(A)$ and so every integer vector satisfies this inequality. In particular the least cost integer vector, say z^*, satisfies this inequality and we have shown that $c(z^*) \leq \alpha c(x)$. Now if x satisfies the KC inequalities for the set $A^{x,\alpha}$ i.e. if $u_A(x(E\backslash A)) \geq D(A)$ and we set $\alpha = 2$, then $c(z^*) \leq 2c(x)$ and $u_A(z^*(E\backslash A)) \geq D(A)$ which implies $u(z^*(E\backslash A)) \geq D(A)$. Additionally since $z^*(e) = 1$ for every $e \in A$, $u(z^*(A)) \geq D - D(A)$ so that $u(z^*) \geq D$. Thus z^* is feasible to the original problem and even though we do not know how to separate over the KC inequalities we can work in conjunction with the ellipsoid method to obtain an approximation algorithm for the Minimum Knapsack Problem with approximation factor 2. A sketch of how this works is as follows: fix a cost K and add the inequality $c(x) \leq K$ to the $MinKP - KCLP$ polytope. At a given point x, one can run the above algorithm to obtain an integer point z with cost at most $2c(x)$. Either z is feasible to the original problem or $u_A(x(E\backslash A)) < D(A)$ and we can add this separating hyperplane. Thus, using the ellipsoid method, we either obtain a feasible integer point with cost at most $2K$ or we obtain that there is no feasible point x to $MinKP - KCLP$ such that $c(x) \leq K$. We can then run a binary search on K to find the smallest cost K such that we obtain a feasible integer vector z. This also shows that the integrality gap of $MinKP - KCLP$ is at most 2 and there are tight examples [6] showing that the integrality gap is in fact equal to 2. We shall now describe the *capacitated network design problem* and leverage the Bucketing Algorithm to obtain an approximation algorithm for the problem [6].

2.2 Capacitated Network Design Problem

Network design problems model the efficient allocation of resources like routers, optical fibres, roads, canals etc. to effectively construct and operate physical infrastructures. Here, we consider the *CapNDP* arising in network security. We define the problem below.

Problem 2 (Capacitated Network Design Problem)**.** *Given a network or a multigraph $G = (V, E)$ along with capacities $\{u(e)\}_{e \in E}$ and costs $\{c(e)\}_{e \in E}$ for each edge and non-negative demands D_{ij} for every pair of nodes (i, j), we wish to find a minimum cost subgraph H such that for each pair of nodes (i, j), H admits a flow of value D_{ij} between i and j.*

Note that we allow the network to be a multigraph and so we could have multiple edges between the same pair of nodes. The max-flow min-cut theorem allows one to equivalently view and define the problem as follows,

Problem. *Given a network or a multigraph $G = (V, E)$ along with non-negative integral capacities $\{u(e)\}_{e \in E}$ and costs $\{c(e)\}_{e \in E}$ for each edge and non-negative demands D_{ij} for every pair of nodes (i, j), we wish to find a minimum cost subgraph H such that for each pair of nodes (i, j), the capacity of the minimum-cut between nodes i and j in H is at least D_{ij}.*

As Carr *et al.* [6] observed, the capacitated network design problem arises as a network reinforcement problem. Here we are given an existing network, and for each edge in the network several security reinforcement options. In addition for each pair of nodes in the network, there is a specified level of protection demanded. The objective is to select a minimum-cost set of reinforcements for all the edges so that an adversary with strength less than the protection level of a particular pair of nodes cannot disconnect these nodes.

We will now describe the $(\beta(G) + 1)$-approximation algorithm for CapNDP on general graphs [6] as it will be useful in our discussion on outerplanar graphs. Here $\beta(G)$ is the maximum size of a bond of G and a bond is a minimal set of edges that separates a pair of nodes with positive demand in the underlying simple graph.

2.2.1 Approximation Algorithm for CapNDP on General Graphs

The idea of the algorithm is that the integer programming formulation of the Capacitated Network Design Problem is very much like that of the Minimum Knapsack Problem with just a collection of covering constraints and so we can add the KC inequalities for each of the covering constraints separately. Then, given a fractional vector $x \in [0,1]^E$, we can consider each edge $e \in E$ separately and run the bucketing algorithm with input $e, u(e), c(e), x(e)$ to obtain a set of integer vectors for each edge e separately. We can then "merge" these sets of integer vectors to obtain a complete solution for CapNDP. We formalize these ideas by first defining some new notation related to multigraphs so as to avoid confusion. Let $G = (V, E)$ be a multigraph,

- The underlying simple graph will be denoted by G^{spl}.
- The number of nodes in G or G^{spl} will be denoted by n. The number of edges in G^{spl} will be denoted by m and the number of edges in G by M.
- The complete set of edges that connects two particular nodes of G will be called a *multiedge* and denoted by $\bar{e}$. The term *edge* will always mean a single edge of the graph G and will be denoted by e.

We wish to formulate CapNDP as an integer program and to do so we will need the following definition of a bond of a graph. This definition is motivated by observing that we can restrict our attention to satisfying the demands across inclusion-wise minimal cuts of the graph as then we would have satisfied all the cuts of the graph.

Definition 2.2.1 (Bond of a Graph)**.** Given a connected multigraph G, an inclusion-wise minimal set of edges whose removal disconnects the graph is called a bond of the graph.

Note that due to the property of a bond being inclusion-wise minimal, the removal of a bond creates exactly two connected components. Hence every bond is a cut of the graph. We will also need more notation which we will use consistently throughout this thesis. Let $G = (V, E), \{u(e)\}_{e \in E}, \{c(e)\}_{e \in E}, D_{ij}$ be an instance of CapNDP,

- The set of bonds in the graph G that disconnect a pair of nodes with positive demand will be denoted by $\mathcal{C}(G)$ or simply $\mathcal{C}$.
- Bonds in the graph G will be denoted by C and the corresponding bond in G^{spl} will be denoted by C^{spl}
- Given a bond C of G, $D(C)$ will denote the maximum demand between nodes disconnected by the removal of C *i.e.* $D(C) = \max\{D_{ij} : C \text{ is an } ij\text{-cut}\}$.
- The maximum size of a bond in the underlying simple graph that disconnects a pair of nodes with positive demand is denoted $\beta(G)$. Thus $\beta(G) = \max\{|C| : C \in \mathcal{C}(G^{spl})\}$

We can now formulate CapNDP as an integer program,

$$\begin{aligned} \min \quad & \sum_{e\in E} c(e)z(e) & & \qquad \text{(CapNDP-IP)} \\ \text{s.t.} \quad & \sum_{e\in C} u(e)z(e) \geq D(C) & \forall \quad & C \in \mathcal{C} \\ & z(e) \in \{0,1\} & \forall \quad & e \in E \end{aligned}$$

We notice that each of the constraints here is a simple covering constraint and so we can add the KC inequalities described earlier to each constraint separately. This introduces some more notation. Let C be a bond in $\mathcal{C}(G)$, A and S be subsets of the edge set E and x a vector in $[0,1]^E$,

- The capacity of a subset A on a bond C is defined as $u(A\cap C) := \sum_{e\in A\cap C} u(e)$
- The residual demand of A on C is then $D(A,C) := \max\{D(C) - u(A\cap C), 0\}$
- The reduced capacity for every edge $e \in C\backslash A$ is $u_{A,C}(e) = \min\{u(e), D(A,C)\}$
- We will use the shorthand $c(x(S))$ to mean $\sum_{e\in S} c(e)x(e)$
- We will use the shorthand $u_{A,C}(x(S))$ to mean $\sum_{e\in S} u_{A,C}(e)x(e)$

– We will omit S if dealing with the entire set E *i.e.* $c(x) = c(x(E))$

With this notation in place, we can introduce the KC inequalities to CapNDP-IP and obtain the following linear programming relaxation of CapNDP,

$$\begin{aligned}
\min \quad & \sum_{e\in E} c(e)x(e) && \text{(CapNDP-KCLP)}\\
\text{s.t.} \quad & \sum_{e\in C\setminus A} u_{A,C}(e)x(e) \geq D(A,C) && \forall \quad C \in \mathcal{C}, A \subseteq E\\
& x(e) \geq 0 && \forall \quad e \in E
\end{aligned}$$

Observe that for any bond C and any multiedge $\bar{e}$, the entire multiedge either lies inside the bond or outside the bond. Thus we can use the Bucketing Algorithm on each multiedge separately and in so doing we will lose a capacity of at most $D(A,C)$ from each multiedge. We can then merge the integer vectors that we obtain through the bucketing on each multiedge to obtain a complete integer solution to CapNDP. We formalize this idea through the description and analysis of the algorithm below,

Algorithm 2: General Algorithm for CapNDP

```
Input: An instance $G = (V, E), \{u(e)\}_{e\in E}, \{c(e)\}_{e\in E}, D_{ij}$ of capacitated network
    design, a fractional vector $x \in [0,1]^E$ and an integral parameter $\alpha > 1$.
Setup: Let $r$ be the least common multiple of the denominators of $x$ and let
    $A := \{e \in E : x(e) \geq 1/\alpha\}$.
Step 1: Perform the Bucketing Algorithm 1 for each multiedge separately. That
    is, for each multiedge $\bar{e}$, use $\bar{e}$, $\{u(e)\}_{e\in\bar{e}}$, $\{c(e)\}_{e\in\bar{e}}$, $x|_{\bar{e}}$, $\alpha$ as the input in
    Algorithm 1 to obtain a set of $r$ integer vectors $z^1_{\bar{e}}, z^2_{\bar{e}}, \ldots, z^r_{\bar{e}}$ with coordinates for
    each edge $e \in \bar{e}$.
Step 2: Merge these sets of integer vectors for every multiedge in any arbitrary
    order to obtain and output $r$ integer vectors $z^1_E, z^2_E, \ldots, z^r_E$ with coordinates for
    each edge $e \in E$. For example, if $E = \bar{e} \cup \bar{f}$ and step 1 returned $z^1_{\bar{e}}, z^2_{\bar{e}}, \ldots, z^r_{\bar{e}}$ and
    $z^1_{\bar{f}}, z^2_{\bar{f}}, \ldots, z^r_{\bar{f}}$, then pick any random permutations of $1, 2, \ldots, r$, say $\tau$ and $\sigma$ and
    set $z^i_E$ to be $z^{\tau(i)}_{\bar{e}}$ appended with $z^{\sigma(i)}_{\bar{f}}$.
```

Remark 3. Due to Remark 2, one can implement Algorithm 2 in polynomial time.

Let us analyze the properties of the integer vectors that we obtain using this algorithm. Firstly,

- $z^i_{\bar{e}}(e) = 1$ for all $i, \bar{e}$ and every $e \in A \cap \bar{e}$. This implies, $z^i_E(e) = 1$ for all i and every $e \in A$.

This is because we used the bucketing algorithm 1 in Step 1 for each multiedge separately and then just merged these sets of integer vectors together in Step 2. Secondly,

- $\sum_{i=1}^r c(z^i_{\bar{e}}(\bar{e})) \leq r\alpha c(x(\bar{e}))$ for all multiegdes $\bar{e}$. This implies, $\sum_{i=1}^r c(z^i_E) \leq r\alpha c(x)$.

Again, this follows because of the bucketing algorithm 1 in Step 1 and the implication is due to the merging in Step 2. We can thus infer that the integer vector with the smallest cost among the ones obtained through Algorithm 2, $z^1_E, z^2_E, \ldots, z^r_E$ has cost at most $\alpha c(x)$. Thirdly,

- $|u_{A,C}(z^i_{\bar{e}}(\bar{e} \backslash A)) - u_{A,C}(z^j_{\bar{e}}(\bar{e} \backslash A))| \leq D(A, C)$ for every pair i, j and every bond C such that $\bar{e} \subseteq C$. This implies that $|u_{A,C}(z^i_E(C \backslash A)) - u_{A,C}(z^j_E(C \backslash A))| \leq |C^{spl}| \cdot D(A, C)$ for every pair i, j and every bond C.

This follows because of the bucketing algorithm 1 used for every multiedge in Step 1. The implication follows due to the merging in Step 2 and the observation that for any bond C and any multiedge $\bar{e}$, either $\bar{e} \subseteq C$ or $\bar{e} \cap C = \emptyset$. Finally,

- $\sum_{i=1}^r u_{A,C}(z^i_{\bar{e}}(\bar{e} \backslash A)) = r\alpha u_{A,C}(x(\bar{e} \backslash A))$ for all bonds C such that $\bar{e} \subseteq C$. This implies that $\sum_{i=1}^r u_{A,C}(z^i_E(C \backslash A)) = r\alpha u_{A,C}(x(C \backslash A))$.

Again, this follows straight from the bucketing algorithm 1 used for every multiedge in Step 1. The implication follows due to the merging in Step 2 and the observation that for any bond C and any multiedge $\bar{e}$, either $\bar{e} \subseteq C$ or $\bar{e} \cap C = \emptyset$.

The last two properties immediately tell us that for every bond C, the integer vector with the smallest $u_{A,C}$-value among $z_E^1, z_E^2, \dots, z_E^r$ (*say* z_E^r) satisfies $u_{A,C}(z_E^r(C \backslash A)) \geq \alpha u_{A,C}(x(C \backslash A)) - |C^{spl}| \cdot D(A,C)$ and so every integer vector satisfies this inequality. In particular the least cost integer vector, say z^*, satisfies this inequality and we have shown that $c(z^*) \leq \alpha c(x)$. Now if x satisfies the KC inequalities for the set $A^{x,\alpha}$ i.e. if $u_{A,C}(x(C \backslash A)) \geq D(A,C)$ for every bond C and we set $\alpha = \beta(G) + 1$, then $c(z^*) \leq (\beta(G)+1)c(x)$ and $u_{A,C}(z^*(C \backslash A)) \geq D(A,C)$ which implies $u(z^*(C \backslash A)) \geq D(A,C)$. Additionally since $z^*(e) = 1$ for every $e \in C \cap A$, $u(z^*(C \cap A)) \geq D(C) - D(A,C)$ so that $u(z^*(C)) \geq D(C)$ for every bond C. Thus z^* is feasible to the original problem and even though we do not know how to separate over the KC inequalities we can work in conjunction with the ellipsoid method to obtain an approximation algorithm for the Capacitated Network Design Problem with approximation factor $\beta(G) + 1$. Additionally it is *NP*-hard to compute $\beta(G)$, but this too can be overcome since $\beta(G) \leq m$ where m is the number of edges in the underlying simple graph. A sketch of how this works is as follows: for each guess of $\beta(G) = 1, \dots, m$, fix a cost K and add the inequality $c(x) \leq K$ to the $CapNDP - KCLP$ polytope. At a given point x, one can run the above algorithm to obtain an integer point z with cost at most $(\beta(G)+1)c(x)$ for our guess of $\beta(G)$. Either z is feasible to the original problem or we can locate a bond C such that $u(z(C)) < D(C)$. But then $u_{A,C}(x(C \backslash A)) < D(A,C)$ and we can add this separating hyperplane. Thus, using the ellipsoid method, we either obtain a feasible integer point with cost at most $(\beta(G)+1) \cdot K$ for our guess of $\beta(G)$ or we obtain that there is no feasible point x to $CapNDP - KCLP$ with our guess of $\beta(G)$ such that $c(x) \leq K$. We can then, for each guess of $\beta(G)$ run a binary search on K to find the smallest cost K such that we obtain a feasible integer vector z. Thus we can find the smallest value of $(\beta(G)+1)K$ such that z is feasible to the original problem. This also shows that the integrality gap of $CapNDP - KCLP$ is at most $(\beta(G)+1)$. In the next chapter we shall modify Algorithm 2 using a new merging technique that will allow us to obtain a better approximation algorithm for CapNDP on outerplanar graphs.

Chapter 3

CapNDP on Outerplanar Graphs

In their very influential paper, Carr *et al.* [6] had provided a dynamic programming solution to the Capacitated Network Design Problem on outerplanar graphs for instances with exactly one pair of nodes with positive demand. The running time of their dynamic program is $O(mD^3)$ where m is the number of edges in G and D is the demand between the only pair of nodes with positive demand. The running time of this dynamic program depends on D and so common scaling arguments cannot be used to obtain an FPTAS in this case as is claimed by the authors of that paper. The issue is that one would have to scale down the demand D and capacities of the edges (and apply the ceiling function to maintain integrality) in order to obtain a solution to the scaled-down problem in polynomial time. However since the demand D and the capacities of the edges were changed, this solution need not be feasible for the original problem. Thus the best known approximation algorithm for CapNDP on outerplanar graphs prior to our work was the $O(\beta(G))$-approximation algorithm by Carr *et al.* [6] which we have described in chapter 2. In this thesis, we improve the approximation factor for instances of CapNDP on outerplanar graphs by a doubly exponential factor by imposing certain conditions on the demand pairs. These conditions capture the single demand pair case and we obtain an $O((\log\log n)^2)$-approximation algorithm here. The conditions on the demand pairs are technical and depend on the structure of the outerplanar graph. We will also exploit the structure of bonds of outerplanar graphs to obtain our approximation algorithm and so let us begin by describing what outerplanar graphs look like.

Definition 3.0.1 (Outerplanar Graphs)**.** A graph that has a planar embedding in which every node appears on the outer face is called an outerplanar graph

Before describing the structure of outerplanar graphs, we note a useful observation about the CapNDP problem. A biconnected component of a graph G is a maximal 2 node-connected subgraph of G. Given a graph G, we can compute the biconnected components of G using the algorithm by Hopcroft and Tarjan [20]. Since bonds of G lie completely within a biconnected component of G or are bridges of G, we can solve the CapNDP on each biconnected component separately and then piece things together to obtain a solution to the problem on the original graph. Hence we shall now focus only on biconnected outerplanar graphs and they have a very nice structure.

3.1 Structure of Biconnected Outerplanar Graphs

Any biconnected graph must contain a simple cycle. Additionally since we are interested in outerplanar graphs, there can be no other nodes in the graph and in any outerplanar embedding of the graph, every edge must be drawn within this cycle. We can fix two nodes in this cycle and label them s and t respectively. One of the paths on the cycle joining s and t will be called the *Upper Path* and the other the *Lower Path*. Edges that connect a node on the upper path to a node on the lower path will be called *Chords*. Nodes s, t and nodes that are incident to a chord will be called *Chordal Nodes*. The sub-path between any two chordal nodes could have additional edges that don't intersect (see Figure 3.1), we call such a graph a *Non-crossing Interval Graph* as defined below,

Definition 3.1.1 (Non-crossing Interval Graph)**.** A non-crossing interval graph with n nodes say $1, 2, \ldots, n$ has an edge between each pair of consecutive nodes *i.e* 1—2—3—$\ldots$—$(n-1)$—n along with potentially other edges. Each additional edge i—j can be represented by the open interval (i, j). These open intervals are required to form a laminar family *i.e* the intersection of any two intervals I_1 and I_2 is either I_1 or I_2 or empty.

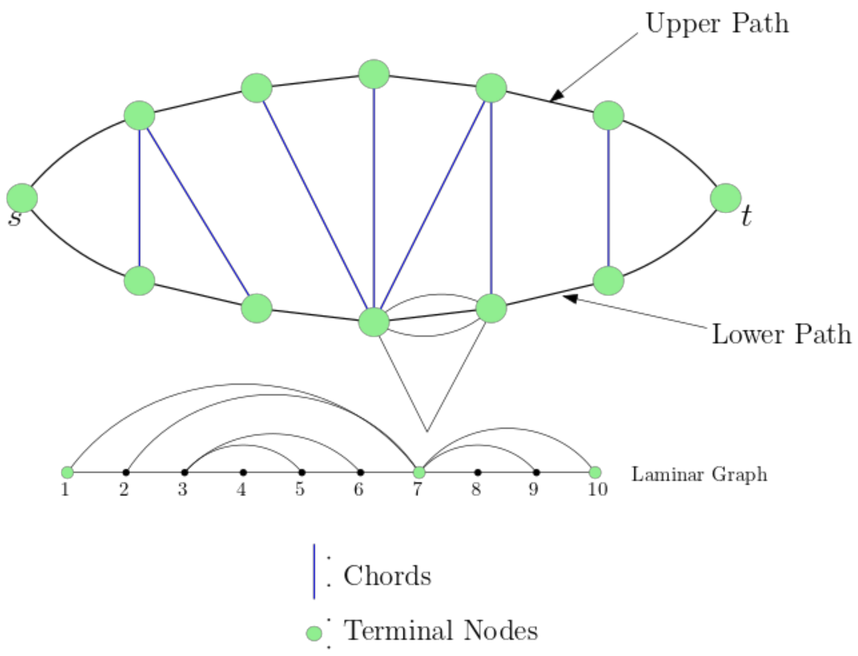

Figure 3.1: Biconnected Outerplanar Graphs

We shall call nodes 1 and n of a non-crossing interval graph as its terminals. We would like to note here that a non-crossing interval graph is biconnected if and only if its terminals are connected and biconnected outerplanar graphs are the same as biconnected non-crossing interval graphs. This can be seen by identifying any two consecutive nodes on the outer cycle of the biconnected outerplanar graph with the terminals of a biconnected non-crossing interval graph. Coming back to the structure of outerplanar graphs, we notice that the path between two consecutive chordal nodes must be a set of biconnected non-crossing interval graphs connected in series. We shall call nodes of the outerplanar graphs that occur as terminals of these biconnected non-crossing interval graphs as *Terminal Nodes* (see Figure 3.1).

Remark 4. Every chordal node is a terminal node. Also the definition of chords, chordal nodes and terminal nodes depend on the choice of s and t.

3.1.1 Bonds of Biconnected Outerplanar Graphs

We shall now explore the structure of bonds in biconnected outerplanar graphs. Firstly recall that a bond C of a graph $G = (V, E)$ is an inclusion-wise minimal set of edges that disconnects the graph. Thus $G \backslash C$ must have exactly two connected components. Thus a bond can also be described using the node sets of these two connected components.

Lemma 3.1.2. *The node set of each of the two connected components separated by a bond in a biconnected outerplanar graph must be a set of consecutive nodes on the cycle.*

Proof. We prove this by contradiction. Suppose that the lemma is not true and consider any particular outerplanar embedding of the graph. Then there exists a bond C and nodes a_1, a_2, a_3, a_4 in order around the cycle such that a_1 and a_3 belong to one of the connected components of $G \backslash C$ and a_2 and a_4 belong to the other. Since all edges must be drawn within the cycle in the outerplanar embedding, it is impossible to connect a_1, a_3 and a_2, a_4 with non-intersecting paths without contradicting the hypothesis that the graph is outerplanar. □

This very crucial lemma allows us to understand the structure of edges in bonds of an outerplanar graph. We shall restrict ourselves to describing bonds that disconnect a pair of terminal nodes of the outerplanar graph called *Terminal Bonds.* First let us understand such bonds in biconnected non-crossing interval graphs.

Terminal Bonds of Biconnected Non-crossing Interval Graphs Recall that the intervals representing the edges of a biconnected non-crossing interval graph $G = (V, E)$ form a laminar family and so can be represented by a tree $T(E)$ rooted at the interval $(1, n)$ as follows: we have a vertex in $T(E)$ for each interval and an edge in $T(E)$ between intervals I_1 and I_2 if I_2 is the smallest interval in E containing I_1. The leaves of this tree $T(E)$ will have to correspond to the edges of G that connect consecutive nodes of G. Since a non-crossing interval graph is also outerplanar, Lemma 3.1.2 applies here and the node sets of the connected components separated by a terminal bond have to be $1, 2, \ldots, i$ and $(i+1), \ldots, n$ for some $i = 1, 2, \ldots, (n-1)$. Let the leaf in $T(E)$ corresponding to the edge $(i, i+1)$ of G be v. In such a case, the edges of G that form the terminal bond are simply the edges of G corresponding to nodes on the path from leaf v to root in $T(E)$(see figure 3.2).

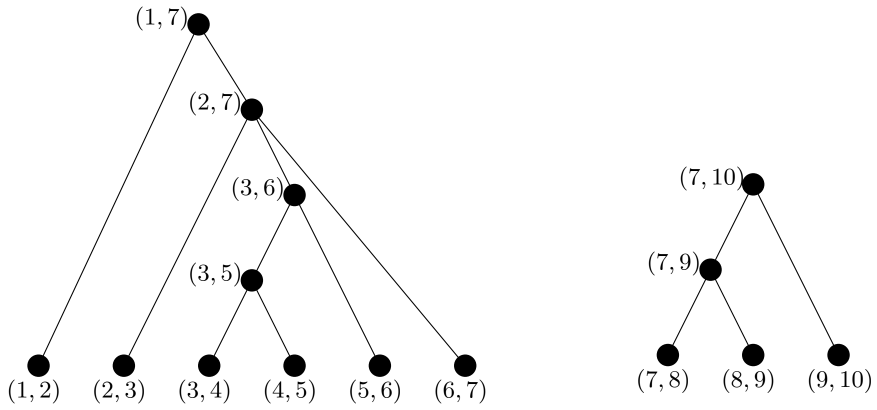

Figure 3.2: T(E) for the biconnected non-crossing interval graphs in Fig 3.1

Terminal Bonds of Biconnected Outerplanar Graphs Firstly the chords of a biconnected outerplanar graph can be ordered from s to t. Due to Lemma 3.1.2, we are interested in the bond generated by a set of consecutive nodes on the cycle that contains a terminal node. This bond contains a terminal bond from each of the biconnected non-crossing interval graphs that the end points of this consecutive set belong to and also contains the set of consecutive chords that lie between these end points. Thus, referring to figure 3.3, we have a possibly empty set of consecutive chords (blue) along with a terminal bond from one of the biconnected non-crossing interval graphs connected in series to the immediate left of the leftmost chord ($i.e$ from the non-crossing interval graphs that occur on either the upper or lower path to the left of the leftmost chord till the next chordal nodes on the left, shown in green) and a terminal bond from one of the biconnected non-crossing interval graphs connected in series to the immediate right of the rightmost chord (shown in red).

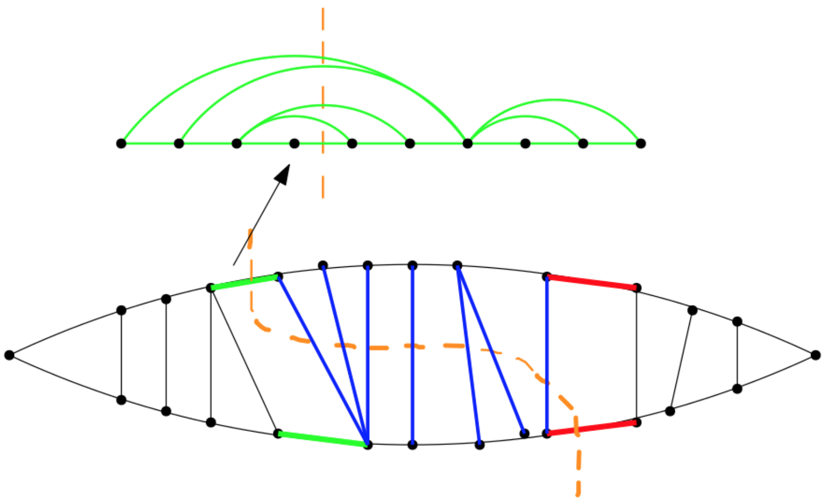

Figure 3.3: Terminal bonds of biconnected outerplanar graphs

With these descriptions of terminal bonds, we can begin to describe our new algorithm for CapNDP on outerplanar graphs.

3.2 Merging Algorithm

The main idea is that since the terminal bonds of biconnected outerplanar graphs have a very nice structure, it seems wasteful to arbitrarily merge the integer vectors as done in Step 2 of Algorithm 2. During this merging step, we are given two sets of integer vectors $z^1_{E_1}, z^2_{E_1}, \dots, z^r_{E_1}$ and $z^1_{E_2}, z^2_{E_2}, \dots, z^r_{E_2}$ defined on disjoint edge sets E_1 and E_2 such that for any bond C, the difference between the $u_{A,C}$-values of any two vectors from the same set E_1 or E_2 is at most $D(A, C)$. After merging them arbitrarily, the $u_{A,C}$-values of the new integer vectors $z^1_{E_1 \cup E_2}, z^2_{E_1 \cup E_2}, \dots, z^r_{E_1 \cup E_2}$ defined on $E_1 \cup E_2$ differ by at most $2D(A, C)$. If instead we were to initially sort the two sets of vectors according to non-increasing $u_{A,C}$-values and merge them in opposite orders, then the $u_{A,C}$-values of the new integer vectors defined on $E_1 \cup E_2$ would differ by at most $D(A, C)$. The issue here however is that the $u_{A,C}$-orderings of the integer vectors could change when considering different bonds. We overcome this by sorting the vectors using their u-values and observing that if this ordering does not match with a $u_{A,C}$-ordering, then the capacity of some edge must be capped by $D(A,C)$ and so the bond C is trivially satisfied. We formalize and explain this idea in the algorithm below.

```
Algorithm 3: Merging Algorithm
  Input: Two disjoint edge sets $E_1$ and $E_2$ along with capacities on edges
    $\{u(e)\}_{e\in E_1\cup E_2}$; two sets of integer vectors $z^1_{E_1}, z^2_{E_1}, \ldots, z^r_{E_1}$ and $z^1_{E_2}, z^2_{E_2}, \ldots, z^r_{E_2}$
    defined on the disjoint edge sets $E_1$ and $E_2$; two subsets of edges $F_1 \subseteq E_1$ and
    $F_2 \subseteq E_2$. The sets $F_1$ and $F_2$ can be thought of as the intersection of a bond $C$
    with $E_1$ and $E_2$ respectively.
  Step 1: Sort the first set of integer vectors according to non-increasing
    $u(z^i_{E_1}(F_1))$-values and sort the second set of integer vectors according to
    non-increasing $u(z^i_{E_2}(F_2))$-values. WLOG, let these sorted ordered be
    $z^1_{E_1}, z^2_{E_1}, \ldots, z^r_{E_1}$ and $z^1_{E_2}, z^2_{E_2}, \ldots, z^r_{E_2}$
  Step 2: Merge these two sorted sets in opposite orders to obtain and output a set
    of integer vectors $z^1_{E_1\cup E_2}, z^2_{E_1\cup E_2}, \ldots, z^r_{E_1\cup E_2}$ defined on $E_1 \cup E_2$ i.e. $z^i_{E_1\cup E_2}$ is
    simply $z^i_{E_1}$ appended with $z^{r-i+1}_{E_2}$.
```

Let us analyze how the merging algorithm performs in the setting of the Capacitated Network Design Problem. So let $G = (V,E), \{u(e)\}_{e\in E}, \{c(e)\}_{e\in E}, D_{ij}$ be an instance of CapNDP. Let x be a vector in $[0,1]^E$ and α and integral parameter greater than 1. Let r be the least common multiple of the denominators of x and let $A := \{e \in E : x(e) \geq 1/\alpha\}$. Let C be a bond in the graph such that $C \cap E_1 = F_1$ and $C \cap E_2 = F_2$. Additionally let $z^1_{E_1}, z^2_{E_1}, \ldots, z^r_{E_1}$ and $z^1_{E_2}, z^2_{E_2}, \ldots, z^r_{E_2}$ be integer vectors defined on disjoint subsets of E namely E_1 and E_2 sorted in non-increasing $u(z^i_{E_1}(F_1))$-values and non-increasing $u(z^i_{E_2}(F_2))$-values respectively. Let $z^1_{E_1\cup E_2}, z^2_{E_1\cup E_2}, \ldots, z^r_{E_1\cup E_2}$ be the output of the merging algorithm 3 using input $E_1, E_2, u(e), F_1 \subseteq E_1, F_2 \subseteq E_2$ (*i.e* $z^i_{E_1\cup E_2}$ is just $z^i_{E_1}$ appended with $z^{r-i+1}_{E_2}$). Then we have the following theorem.

Theorem 3.2.1. *Suppose the original sets of integer vectors satisfy the following properties for $k = 1, 2$ and any fixed index $i^* = 1, 2, \ldots, r$:*

(a) $z^i_{E_k}(e) = 1$ *for all* $e \in A \cap E_k$ *and for all* i.

(b) $\sum_{i=1}^{r} c(z^i_{E_k}(E_k)) \leq r\alpha c(x(E_k))$.

(c) $|u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) - u_{A,C}(z^i_{E_k}(F_k \backslash A))| \leq D(A,C)$ *for all* i.

(d) $\sum_{i=1}^{r} u_{A,C}(z^i_{E_k}(F_k \backslash A)) = r\alpha u_{A,C}(x(F_k \backslash A))$.

Then the set of merged integer vectors satisfy:

1. $z^i_{E_1 \cup E_2}(e) = 1$ *for all* $e \in A \cap (E_1 \cup E_2)$ *and for all* i.
2. $\sum_{i=1}^{r} c(z^i_{E_1 \cup E_2}(E_1 \cup E_2)) \leq r\alpha c(x(E_1 \cup E_2))$.
3. ***Either*** $|u_{A,C}(z^{i^*}_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A)) - u_{A,C}(z^i_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A))| \leq D(A,C)$ *for all* i ***or*** $u_{A,C}(z^{i^*}_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A)) \geq D(A,C)$.
4. $\sum_{i=1}^{r} u_{A,C}(z^i_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A)) = r\alpha u_{A,C}(x(F_1 \cup F_2 \backslash A))$.

Proof. Properties 1,2 and 4 will trivially hold for any arbitrary merging as seen earlier in the analysis of Algorithm 2. We need to prove that property 3 holds after merging and we do so by considering the following exhaustive list of cases.

Case 1: For all i and every $k = 1, 2$, $u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) \geq u_{A,C}(z^i_{E_k}(F_k \backslash A))$ if and only if $u(z^{i^*}_{E_k}(F_k \backslash A)) \geq u((z^i_{E_k}(F_k \backslash A))$
Then the $u(z^i_{E_k}(F_k))$-ordering used in the merging algorithm matches with the $u_{A,C}(z^i_{E_k}(F_k))$-ordering with respect to the index i^*. Since we are then merging in oppositely sorted orders and the difference in the $u_{A,C}(z^i_{E_k}(F_k))$-values of z^{i^*} and any z^i is at most $D(A,C)$ to begin with, the difference between $u_{A,C}(z^i_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A))$-values of z^{i^*} and any z^i is also at most $D(A,C)$.

Case 2: For some i and some $k = 1, 2$, $u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) \geq u_{A,C}(z^i_{E_k}(F_k \backslash A))$ but $u(z^{i^*}_{E_k}(F_k \backslash A)) < u((z^i_{E_k}(F_k \backslash A))$
Then we claim that there must be an edge e in $F_k \backslash A$ such that $z^i_{E_k}(e) = 1$ and $u(e) \geq D(A,C)$. If this is not the case, then

$$\begin{aligned} u_{A,C}(z^i_{E_k}(F_k \backslash A)) &= u((z^i_{E_k}(F_k \backslash A)) && \text{since no edge capacity is capped} \\ &> u(z^{i^*}_{E_k}(F_k \backslash A)) && \text{due to the hypothesis of Case 2} \\ &\geq u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) && \text{since } u(e) \geq u_{A,C}(e) \end{aligned}$$

But this contradicts the hypothesis of Case 2. Thus we can conclude that $u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) \geq u_{A,C}(z^{i}_{E_k}(F_k \backslash A)) \geq D(A,C)$ and so $u_{A,C}(z^{i^*}_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A)) \geq D(A,C)$.

Case 3: For some i and some $k = 1, 2$, $u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) \leq u_{A,C}(z^{i}_{E_k}(F_k \backslash A))$ but $u(z^{i^*}_{E_k}(F_k \backslash A)) > u((z^{i}_{E_k}(F_k \backslash A))$
This is very similar to Case 2 and we will obtain that $u_{A,C}(z^{i^*}_{E_k}(F_k \backslash A)) \geq D(A,C)$ and so $u_{A,C}(z^{i^*}_{E_1 \cup E_2}(F_1 \cup F_2 \backslash A)) \geq D(A,C)$. □

The merging algorithm can thus be used to improve step 2 of algorithm 2. If the bonds in the underlying graph have a nice structure, the merging algorithm may be implemented in a suitable way to improve the approximation ratio of CapNDP. We exhibit this first in the case of terminal bonds of non-crossing interval graphs where we are able to prove the following result.

Theorem 3.2.2. *There exists a 2-approximation algorithm for CapNDP on non-crossing interval graphs when demands occur only between terminal nodes*

3.3 2-approximation for Terminal Bonds of Non-crossing Interval Graphs

We wish to exploit the structure of terminal bonds in non-crossing interval graphs using the merging algorithm to obtain a better approximation algorithm for such instances. Terminal bonds of non-crossing interval graphs have the very nice structure described earlier in that they are paths from leaves to roots in $T(E)$ (*see 3.1.1*). The nodes of $T(E)$ correspond to edges of G and can be divided into levels as with any rooted tree (level 0 comprises of the root and level $i+1$ comprises of the immediate children of nodes from level i). Given any edge e_i from level i, we know exactly which edges from lower levels are used in bonds containing e_i, they will be the edges on the unique path in $T(E)$ from e_i to the root. We can exploit this observation using the following algorithm.

Algorithm 4: 2-approximation for CapNDP on non-crossing interval graphs

Input: An instance $G = (V, E), \{u(e)\}_{e \in E}, \{c(e)\}_{e \in E}, D_{ij}$ of CapNDP where G is a non-crossing interval graph and the only pair of nodes with positive demand is the pair of terminals, a vector $x \in [0, 1]^E$ and an integral parameter $\alpha > 1$.

Setup: Let r be the least common multiple of denominators of x and let $A := \{e \in E : x(e) \geq 1/\alpha\}$

Step 1: Perform the Bucketing Algorithm 1 for each multiedge separately. That is, for each multiedge $\bar{e}$, use $\bar{e}$, $\{u(e)\}_{e \in \bar{e}}$, $\{c(e)\}_{e \in \bar{e}}$, $x|_{\bar{e}}$, α as the input in Algorithm 1 to obtain a set of r integer vectors $z^1_{\bar{e}}, z^2_{\bar{e}}, \ldots, z^r_{\bar{e}}$ with coordinates for each edge $e \in \bar{e}$.

Step 2: Instead of arbitrarily merging these sets of integer vectors, merge them level by level as follows: When merging a multiedge $\bar{e}_{i+1}$ from level $i + 1$ of $T(E)$, we would have already merged all multiedges from levels $0, 1, \ldots, i$ and so we already have a set of integer vectors $z^1_{E_1}, z^2_{E_1}, \ldots, z^r_{E_1}$ for a subset $E_1 \subseteq E$ that contains all multiedges from levels $0, 1, \ldots, i$. Let $\bar{e}_{i+1}$,$\bar{e}_i, \ldots, \bar{e}_0$ be the unique path in $T(E)$ from $\bar{e}_{i+1}$ to the root and let $F_1 = \{\bar{e}_i, \ldots, \bar{e}_0\}$. We then merge $z^1_{E_1}, z^2_{E_1}, \ldots, z^r_{E_1}$ with $z^1_{\bar{e}_{i+1}}, z^2_{\bar{e}_{i+1}}, \ldots, z^r_{\bar{e}_{i+1}}$ using our merging algorithm 3 with $E_1, \bar{e}_{i+1}, u(e), F_1 \subseteq E_1, \bar{e}_{i+1} \subseteq \bar{e}_{i+1}$ as input. We output $z^1_E, z^2_E, \ldots, z^r_E$ after merging every multiedge (See Example 3.3 for an illustration of this step).

Remark 5. Due to Remark 2, one can implement Algorithm 4 in polynomial time.

Example (Step 2 of Algorithm 4)**.** *Let us consider the simple non-crossing interval graph shown in figure 3.4. Here, the graph G along with its corresponding $T(E)$ is shown. Let us assume that each multiedge $\bar{e}_i$ comprises of two edges e_{i0} and e_{i1}. Furthermore, let us* *assume that the bucketing algorithm in Step 1 of Algorithm 4 outputs the following vectors.*

	$u(e)$	$z^1_{\bar{e}_0}$	$z^2_{\bar{e}_0}$	$z^3_{\bar{e}_0}$	$z^4_{\bar{e}_0}$	$z^5_{\bar{e}_0}$
e_{00}	5	1	1	1	1	0
e_{01}	4	1	0	0	0	1

	$u(e)$	$z^1_{\bar{e}_1}$	$z^2_{\bar{e}_1}$	$z^3_{\bar{e}_1}$	$z^4_{\bar{e}_1}$	$z^5_{\bar{e}_1}$
e_{10}	7	1	1	1	0	0
e_{11}	2	1	1	0	1	1

	$u(e)$	$z^1_{\bar{e}_2}$	$z^2_{\bar{e}_2}$	$z^3_{\bar{e}_2}$	$z^4_{\bar{e}_2}$	$z^5_{\bar{e}_2}$
e_{20}	5	1	0	0	0	0
e_{21}	3	0	1	1	1	0

	$u(e)$	$z^1_{\bar{e}_3}$	$z^2_{\bar{e}_3}$	$z^3_{\bar{e}_3}$	$z^4_{\bar{e}_3}$	$z^5_{\bar{e}_3}$
e_{30}	2	1	1	1	0	0
e_{31}	1	1	0	0	1	1

	$u(e)$	$z^1_{\bar{e}_4}$	$z^2_{\bar{e}_4}$	$z^3_{\bar{e}_4}$	$z^4_{\bar{e}_4}$	$z^5_{\bar{e}_4}$
e_{40}	4	1	1	1	1	0
e_{41}	1	1	0	0	0	1

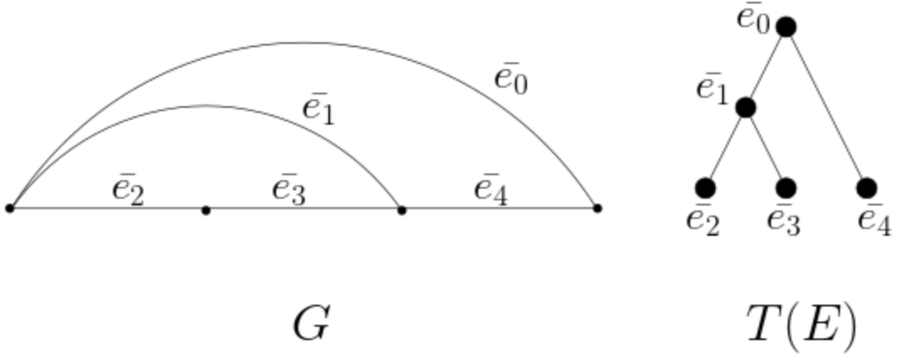

Figure 3.4: Non-crossing interval graph G and its corresponding $T(E)$

Step 2 begins from level 0 of $T(E)$. Here there is only one node $\bar{e}_0$ and so no merging is required. We currently have merged all nodes of level 0 and obtained $z^1_{\bar{e}_0}, \ldots, z^5_{\bar{e}_0}$. We move on to level 1 of $T(E)$ and consider the node $\bar{e}_1$. The unique path from this node to the root in $T(E)$ is $\bar{e}_1, \bar{e}_0$. Thus E_1 which is the set of multiedges already merged is the singleton $\{\bar{e}_0\}$ and F_1 is also $\{\bar{e}_0\}$. We then apply the merging algorithm by sorting $z^1_{\bar{e}_0}, \ldots, z^5_{\bar{e}_0}$ according to non-increasing $u(z^i_{\bar{e}_0}(F_1))$-values to obtain the order $z^1_{\bar{e}_0}, z^2_{\bar{e}_0}, z^3_{\bar{e}_0}, z^4_{\bar{e}_0}, z^5_{\bar{e}_0}$ and we sort $z^1_{\bar{e}_1}, \ldots, z^5_{\bar{e}_1}$ according to non-increasing $u(z^i_{\bar{e}_1}(\bar{e}_1))$-values to obtain the order $z^1_{\bar{e}_1}, z^2_{\bar{e}_1}, z^3_{\bar{e}_1}, z^4_{\bar{e}_1}, z^5_{\bar{e}_1}$. We then merge these vectors in opposite orders to obtain the following vectors (here the *set $\bar{E}_1$ is $\{\bar{e}_0, \bar{e}_1\}$).*

	$u(e)$	$z^1_{\bar{E}_1}$	$z^2_{\bar{E}_1}$	$z^3_{\bar{E}_1}$	$z^4_{\bar{E}_1}$	$z^5_{\bar{E}_1}$
e_{00}	5	1	1	1	1	0
e_{01}	4	1	0	0	0	1
e_{10}	7	0	0	1	1	1
e_{11}	2	1	1	0	1	1

We currently have merged all nodes of level 0 and 1 and obtained $z^1_{\bar{E}_1}, \ldots, z^5_{\bar{E}_1}$. We move on to level 2 of $T(E)$ and consider the node $\bar{e}_2$. The unique path from this node to the root in $T(E)$ is $\bar{e}_2, \bar{e}_1, \bar{e}_0$. Thus E_1 which is the set of multiedges already merged is the set $\{\bar{e}_0, \bar{e}_1\}$ and F_1 is also $\{\bar{e}_0, \bar{e}_1\}$. We then apply the merging algorithm by sorting $z^1_{\bar{E}_1}, \ldots, z^5_{\bar{E}_1}$ according to non-increasing $u(z^i_{\bar{E}_1}(F_1))$-values to obtain the order $z^4_{\bar{E}_1}, z^5_{\bar{E}_1}, z^3_{\bar{E}_1}, z^1_{\bar{E}_1}, z^2_{\bar{E}_1}$ and we sort $z^1_{\bar{e}_2}, \ldots, z^5_{\bar{e}_2}$ according to non-increasing $u(z^i_{\bar{e}_2}(\bar{e}_2))$-values to obtain the order $z^1_{\bar{e}_2}, z^2_{\bar{e}_2}, z^3_{\bar{e}_2}, z^4_{\bar{e}_2}, z^5_{\bar{e}_2}$. We then merge these vectors in opposite orders to obtain the following vectors (here the set $\bar{E}_2$ is $\{\bar{e}_0, \bar{e}_1, \bar{e}_2\}$).

	$u(e)$	$z^1_{\bar{E}_2}$	$z^2_{\bar{E}_2}$	$z^3_{\bar{E}_2}$	$z^4_{\bar{E}_2}$	$z^5_{\bar{E}_2}$
e_{00}	5	1	0	1	1	1
e_{01}	4	0	1	0	1	0
e_{10}	7	1	1	1	0	0
e_{11}	2	1	1	0	1	1
e_{20}	5	0	0	0	0	1
e_{21}	3	0	1	1	1	0

We currently have merged all nodes of level 0 and 1 and node $\bar{e}_2$ to obtain $z^1_{\bar{E}_2}, \ldots, z^5_{\bar{E}_2}$. We move on to node $\bar{e}_3$. The unique path from this node to the root in $T(E)$ is $\bar{e}_3, \bar{e}_1, \bar{e}_0$. Thus E_1 which is the set of multiedges already merged is the set $\{\bar{e}_0, \bar{e}_1, \bar{e}_2\}$ and F_1 is the set $\{\bar{e}_0, \bar{e}_1\}$. We then apply the merging algorithm by sorting $z^1_{\bar{E}_2}, \ldots, z^5_{\bar{E}_2}$ according to non-increasing $u(z^i_{\bar{E}_2}(F_1))$-values to obtain the order $z^1_{\bar{E}_2}, z^2_{\bar{E}_2}, z^3_{\bar{E}_2}, z^4_{\bar{E}_2}, z^5_{\bar{E}_2}$ and we sort $z^1_{\bar{e}_3}, \ldots, z^5_{\bar{e}_3}$ according to non-increasing $u(z^i_{\bar{e}_3}(\bar{e}_3))$-values to obtain the order $z^1_{\bar{e}_3}, z^2_{\bar{e}_3}, z^3_{\bar{e}_3}, z^4_{\bar{e}_3}, z^5_{\bar{e}_3}$. We then merge these vectors in opposite orders to obtain the following vectors (here the *set $\bar{E}_3$ is $\{\bar{e}_0, \bar{e}_1, \bar{e}_2, \bar{e}_3\}$).*

	$u(e)$	$z^1_{\bar{E}_3}$	$z^2_{\bar{E}_3}$	$z^3_{\bar{E}_3}$	$z^4_{\bar{E}_3}$	$z^5_{\bar{E}_3}$
e_{00}	5	1	0	1	1	1
e_{01}	4	0	1	0	1	0
e_{10}	7	1	1	1	0	0
e_{11}	2	1	1	0	1	1
e_{20}	5	0	0	0	0	1
e_{21}	3	0	1	1	1	0
e_{30}	2	0	0	1	1	1
e_{31}	1	1	1	0	0	1

We currently have merged all nodes of level 0 and 1 and nodes $\bar{e}_2, \bar{e}_3$ to obtain $z^1_{\bar{E}_3}, \ldots, z^5_{\bar{E}_3}$. We move on to node $\bar{e}_4$. The unique path from this node to the root in $T(E)$ is $\bar{e}_4, \bar{e}_0$. Thus E_1 which is the set of multiedges already merged is the set $\{\bar{e}_0, \bar{e}_1, \bar{e}_2, \bar{e}_3\}$ and F_1 is the singleton $\{\bar{e}_0\}$. We then apply the merging algorithm by sorting $z^1_{\bar{E}_3}, \ldots, z^5_{\bar{E}_3}$ according to non-increasing $u(z^i_{\bar{E}_3}(F_1))$-values to obtain the order $z^4_{\bar{E}_3}, z^1_{\bar{E}_3}, z^3_{\bar{E}_3}, z^5_{\bar{E}_3}, z^2_{\bar{E}_3}$ and we sort $z^1_{\bar{e}_4}, \ldots, z^5_{4_3}$ according to non-increasing $u(z^i_{\bar{e}_4}(\bar{e}_4))$-values to obtain the order $z^1_{\bar{e}_4}, z^2_{\bar{e}_4}, z^3_{\bar{e}_4}, z^4_{\bar{e}_4}, z^5_{\bar{e}_4}$. We then merge these vectors in opposite orders to obtain the following vectors (here the set $\bar{E}_4$ is $\{\bar{e}_0, \bar{e}_1, \bar{e}_2, \bar{e}_3, \bar{e}_4\} = E$).

	$u(e)$	$z^1_{\bar{E}_4}$	$z^2_{\bar{E}_4}$	$z^3_{\bar{E}_4}$	$z^4_{\bar{E}_4}$	$z^5_{\bar{E}_4}$
e_{00}	5	1	1	1	1	0
e_{01}	4	1	0	0	0	1
e_{10}	7	0	1	1	0	1
e_{11}	2	1	1	0	1	1
e_{20}	5	0	0	0	1	0
e_{21}	3	1	0	1	0	1
e_{30}	2	1	0	1	1	0
e_{31}	1	0	1	0	1	1
e_{40}	4	0	1	1	1	1
e_{41}	1	1	0	0	0	1

We have now merged every multiedge and we output $z^1_{\bar{E}_4}, \ldots, z^5_{\bar{E}_4}$

Let us analyze the properties of the integer vectors that we obtain using this algorithm. Firstly, since we are essentially changing Step 2 of algorithm 2, all properties that hold after arbitrarily merging the sets of integer vectors obtained after Step 1 will still hold here as in Algorithm 2. Thus,

- $z_E^i(e) = 1$ for all i and every $e \in A$.
- $\sum_{i=1}^r c(z_E^i) \leq r\alpha c(x)$.
- $\sum_{i=1}^r u_{A,C}(z_E^i(C \backslash A)) = r\alpha u_{A,C}(x(C \backslash A))$ for any bond $C \in \mathcal{C}$.

Let $z_E^{i^*}$ be the integer vector with least cost among the ones obtained through algorithm 4. We know from the second property above that $c(z_E^{i^*}) \leq \alpha c(x)$. We also claim that due to Step 2 of the algorithm, we have the following property,

- For any bond $C \in \mathcal{C}$, **either** $|u_{A,C}(z_E^{i^*}(C \backslash A)) - u_{A,C}(z_E^i(C \backslash A))| \leq D(A,C)$ for all i **or** $u_{A,C}(z_E^{i^*}(C \backslash A)) \geq D(A,C)$.

This easily follows due to Property 3 of the merging algorithm 3 and the fact that terminal bonds of non-crossing interval graphs are the paths from a leaf to the root in $T(E)$. Essentially we are merging every multiedge using the unique path to the root of $T(E)$ in step 2 of algorithm 4 and so if we follow the components of $z_E^{i^*}$, we will obtain the desired result for any terminal bond.

The last two properties immediately tell us that for every bond C, $u_{A,C}(z_E^{i^*}(C \backslash A)) \geq D(A,C)$ or $u_{A,C}(z_E^{i^*}(C \backslash A)) \geq \alpha u_{A,C}(x(C \backslash A)) - D(A,C)$. Now if x satisfies the KC inequalities for the set $A^{x,\alpha}$ i.e. if $u_{A,C}(x(C \backslash A)) \geq D(A,C)$ for every bond C and we set $\alpha = 2$, then $c(z_E^{i^*}) \leq 2c(x)$ and $u_{A,C}(z_E^{i^*}(C \backslash A)) \geq D(A,C)$ which implies $u(z_E^{i^*}(C \backslash A)) \geq D(A,C)$. Additionally since $z_E^*(e) = 1$ for every $e \in C \cap A$, $u(z_E^*(C \cap A)) \geq D(C) - D(A,C)$ so that $u(z_E^*(C)) \geq D(C)$ for every bond C. Thus, z_E^* is feasible to the original problem. We can run the ellipsoid method to obtain a 2-approximation algorithm for the CapNDP on non-crossing interval graphs having terminal demands proving theorem 3.2.2.

3.4 Approximation Algorithm for Terminal Bonds of Outerplanar Graphs

We wish to exploit the structure of terminal bonds in outerplanar graphs using the merging algorithm to obtain a better approximation algorithm for such instances. Terminal bonds of outerplanar graphs have the very nice structure described earlier in that they are a possibly empty set of consecutive chords along with a terminal bond from one of the biconnected non-crossing interval graphs connected in series to the immediate left of the leftmost chord (*i.e* from the non-crossing interval graphs that occur on either the upper or lower path to the left of the leftmost chord till the next chordal nodes on the left) and a terminal bond from one of the biconnected non-crossing interval graphs connected in series to the immediate right of the rightmost chord (*see 3.1.1*). As before, we shall first use the bucketing algorithm 1 on each multiedge separately. What we mean by this is, given a fractional vector $x \in [0,1]^E$, we consider each multiedge $\bar{e} \subseteq E$ separately and run the bucketing algorithm with input $\bar{e}, \{u(e)_{e\in\bar{e}}\}, \{c(e)_{e\in\bar{e}}\}, x(e)|_{\bar{e}}$ to obtain a set of integer vectors for each multiedge $\bar{e}$ separately. We already know how to merge the sets of integer vectors obtained from the multiedges on the non-crossing interval graphs from the previous section. Thus, we only have to worry about merging the sets of integer vectors obtained from the multichords. Let the multichords in order from s to t be $\bar{e}_1, \bar{e}_2, \dots, \bar{e}_k$. We merge the sets of integer vectors obtained from these multiedges as follows:

Merging along a Tree: First, construct a rooted binary tree T whose set of leaves are labelled $\bar{e}_1, \bar{e}_2, \dots, \bar{e}_k$. Second, label every other node in T using the set of its leaf-descendants (so for example the root will be labelled $\{\bar{e}_1, \bar{e}_2, \dots, \bar{e}_k\}$). Third, merge the integer sets upwards from the leaves of T using the merging algorithm. Essentially, we take two nodes that share a parent, say they correspond to sets E_1 and E_2, and merge the sets of integer vectors that are already obtained while merging upwards, $z^1_{E_1}, z^2_{E_1}, \dots, z^r_{E_1}$ and $z^1_{E_2}, z^2_{E_2}, \dots, z^r_{E_2}$ using the merging algorithm 3 with input $E_1, E_2, u(e), E_1 \subseteq E_1, E_2 \subseteq E_2$ (See Example 3.4 for an illustration).

Example (Merging along a tree)**.** *Let us consider a simple example with five multichords and the tree shown in figure 3.5. In the figure,* $\bar{E}_2 = \{\bar{e}_1, \bar{e}_2\}$*;* $\bar{E}_3 = \{\bar{e}_1, \bar{e}_2, \bar{e}_3\}$*;* $\bar{E}_4 =$

$\{\bar{e}_4, \bar{e}_5\}$ and $E = \{\bar{e}_1, \bar{e}_2, \bar{e}_3, \bar{e}_4, \bar{e}_5\}$. Let us assume that each multiedge $\bar{e}_i$ comprises of two edges e_{i0} and e_{i1}. Furthermore, let us assume that after applying the bucketing algorithm on each multiedge separately we obtain the following vectors.

	$u(e)$	$z^1_{\bar{e}_1}$	$z^2_{\bar{e}_1}$	$z^3_{\bar{e}_1}$	$z^4_{\bar{e}_1}$	$z^5_{\bar{e}_1}$
e_{00}	5	1	1	1	1	0
e_{01}	4	1	0	0	0	1

	$u(e)$	$z^1_{\bar{e}_2}$	$z^2_{\bar{e}_2}$	$z^3_{\bar{e}_2}$	$z^4_{\bar{e}_2}$	$z^5_{\bar{e}_2}$
e_{10}	7	1	1	1	0	0
e_{11}	2	1	1	0	1	1

	$u(e)$	$z^1_{\bar{e}_3}$	$z^2_{\bar{e}_3}$	$z^3_{\bar{e}_3}$	$z^4_{\bar{e}_3}$	$z^5_{\bar{e}_3}$
e_{20}	5	1	0	0	0	0
e_{21}	3	0	1	1	1	0

	$u(e)$	$z^1_{\bar{e}_4}$	$z^2_{\bar{e}_4}$	$z^3_{\bar{e}_4}$	$z^4_{\bar{e}_4}$	$z^5_{\bar{e}_4}$
e_{30}	2	1	1	1	0	0
e_{31}	1	1	0	0	1	1

	$u(e)$	$z^1_{\bar{e}_5}$	$z^2_{\bar{e}_5}$	$z^3_{\bar{e}_5}$	$z^4_{\bar{e}_5}$	$z^5_{\bar{e}_5}$
e_{40}	4	1	1	1	1	0
e_{41}	1	1	0	0	0	1

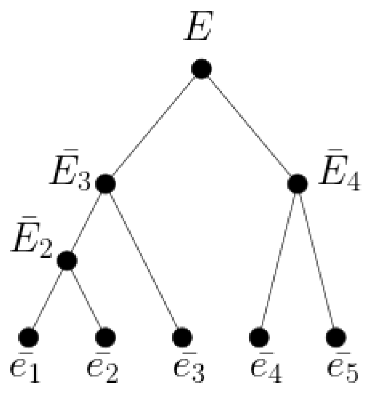

Figure 3.5: Tree used to merge the integer vectors obtained from chords in Example 3.4

The merging along a tree method considers two nodes for which we already have integer vectors such that they share a common parent. Let us start with nodes $\bar{e}_1$ and $\bar{e}_2$. We apply the merging algorithm by sorting $z^1_{\bar{e}_1}, \ldots, z^5_{\bar{e}_1}$ according to non-increasing $u(z^i_{\bar{e}_1}(\bar{e}_1))$-values to obtain the order $z^1_{\bar{e}_1}, z^2_{\bar{e}_1}, z^3_{\bar{e}_1}, z^4_{\bar{e}_1}, z^5_{\bar{e}_1}$ and we sort $z^1_{\bar{e}_2}, \ldots, z^5_{\bar{e}_2}$ according to non-increasing $u(z^i_{\bar{e}_2}(\bar{e}_2))$-values to obtain the order $z^1_{\bar{e}_2}, z^2_{\bar{e}_2}, z^3_{\bar{e}_2}, z^4_{\bar{e}_2}, z^5_{\bar{e}_2}$. We then merge these vectors in *opposite orders to obtain the following vectors.*

	$u(e)$	$z^1_{\bar{E}_2}$	$z^2_{\bar{E}_2}$	$z^3_{\bar{E}_2}$	$z^4_{\bar{E}_2}$	$z^5_{\bar{E}_2}$
e_{10}	5	1	1	1	1	0
e_{11}	4	1	0	0	0	1
e_{20}	7	0	0	1	1	1
e_{21}	2	1	1	0	1	1

We can now consider the two nodes $\bar{E}_2$ and $\bar{e}_3$. We apply the merging algorithm by sorting $z^1_{\bar{E}_2}, \ldots, z^5_{\bar{E}_2}$ according to non-increasing $u(z^i_{\bar{E}_2}(\bar{E}_2))$-values to obtain the order $z^4_{\bar{E}_2}, z^5_{\bar{E}_2}, z^3_{\bar{E}_2}, z^1_{\bar{E}_2}, z^2_{\bar{E}_2}$ and we sort $z^1_{\bar{e}_3}, \ldots, z^5_{\bar{e}_3}$ according to non-increasing $u(z^i_{\bar{e}_3}(\bar{e}_3))$-values to obtain the order $z^1_{\bar{e}_3}, z^2_{\bar{e}_3}, z^3_{\bar{e}_3}, z^4_{\bar{e}_3}, z^5_{\bar{e}_3}$. We then merge these vectors in opposite orders to obtain the following vectors.

	$u(e)$	$z^1_{\bar{E}_3}$	$z^2_{\bar{E}_3}$	$z^3_{\bar{E}_3}$	$z^4_{\bar{E}_3}$	$z^5_{\bar{E}_3}$
e_{10}	5	1	0	1	1	1
e_{11}	4	0	1	0	1	0
e_{20}	7	1	1	1	0	0
e_{21}	2	1	1	0	1	1
e_{30}	5	0	0	0	0	1
e_{31}	3	0	1	1	1	0

We can now consider the two nodes $\bar{e}_4$ and $\bar{e}_5$. We apply the merging algorithm by sorting $z^1_{\bar{e}_4}, \ldots, z^5_{\bar{e}_4}$ according to non-increasing $u(z^i_{\bar{e}_4}(\bar{e}_4))$-values to obtain the order $z^1_{\bar{e}_4}, z^2_{\bar{e}_4}, z^3_{\bar{e}_4}, z^4_{\bar{e}_4}, z^5_{\bar{e}_4}$ and we sort $z^1_{\bar{e}_5}, \ldots, z^5_{\bar{e}_5}$ according to non-increasing $u(z^i_{\bar{e}_5}(\bar{e}_5))$-values to obtain the order $z^1_{\bar{e}_5}, z^2_{\bar{e}_5}, z^3_{\bar{e}_5}, z^4_{\bar{e}_5}, z^5_{\bar{e}_5}$. We then merge these vectors in opposite orders to obtain the following vectors.

	$u(e)$	$z^1_{\bar{E}_4}$	$z^2_{\bar{E}_4}$	$z^3_{\bar{E}_4}$	$z^4_{\bar{E}_4}$	$z^5_{\bar{E}_4}$
e_{40}	2	1	1	1	0	0
e_{41}	1	1	0	0	1	1
e_{50}	4	0	1	1	1	1
e_{51}	1	1	0	0	0	1

We can now consider the two nodes $\bar{E}_3$ and $\bar{E}_4$. We apply the merging algorithm by sorting $z^1_{\bar{E}_3}, \ldots, z^5_{\bar{E}_3}$ according to non-increasing $u(z^i_{\bar{E}_3}(\bar{E}_3))$-values to obtain the order $z^2_{\bar{E}_3}, z^3_{\bar{E}_3}, z^1_{\bar{E}_3}, z^4_{\bar{E}_3}, z^5_{\bar{E}_3}$

and we sort $z^1_{\bar{E}_4}, \ldots, z^5_{\bar{E}_4}$ *according to non-increasing* $u(z^i_{\bar{E}_4}(\bar{E}_4))$*-values to obtain the order* $z^2_{\bar{E}_4}, z^3_{\bar{E}_4}, z^5_{\bar{E}_4}, z^4_{\bar{E}_4}, z^1_{\bar{E}_4}$. *We then merge these vectors in opposite orders to obtain the following vectors.*

	$u(e)$	$z^1_{\bar{E}}$	$z^2_{\bar{E}}$	$z^3_{\bar{E}}$	$z^4_{\bar{E}}$	$z^5_{\bar{E}}$
e_{10}	5	0	1	1	1	1
e_{11}	4	1	0	0	1	0
e_{20}	7	1	1	1	0	0
e_{21}	2	1	0	1	1	1
e_{30}	5	0	0	0	0	1
e_{31}	3	1	1	0	1	0
e_{40}	2	1	0	0	1	1
e_{41}	1	1	1	1	0	0
e_{50}	4	0	1	1	1	1
e_{51}	1	1	0	1	0	0

Having merged all the multiedges, we output $z^1_{\bar{E}}, \ldots, z^5_{\bar{E}}$.

The description of the merging along a tree method above provides a concrete way to merge the sets of integer vectors obtained from multichords using any rooted binary tree with k leaves. We would then want to consider the least cost integer vector obtained after merging every set of integer vectors. We could also consider constructing multiple rooted binary trees and taking the union (integer vectors can also be viewed as sets) of the least cost integer vectors obtained from each of these trees. We formalize this idea in the algorithm below and further analyze the properties of this union of least cost integer vectors.

Algorithm 5: Approximation algorithm for CapNDP on Outerplanar Graphs

Input: An instance $G = (V, E)$, $\{u(e)\}_{e\in E}$, $\{c(e)\}_{e\in E}$, D_{ij} of CapNDP where G is an outerplanar graph and nodes with positive demand occur as terminal nodes; a fractional vector $x \in [0, 1]^E$ and an integral parameter $\alpha > 1$. Let the set of multichords of G be $\bar{F} = \{\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_k\}$ and F be the set of chords. We also need a set (say $T_1, T_2, \ldots, T_t$) of rooted binary trees whose set of leaves are labelled $\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_k$

Setup: Let r be the least common multiple of denominators of x and let $A := \{e \in E : x(e) \geq 1/\alpha\}$

Step 1: Perform the Bucketing Algorithm 1 for each multiedge separately. That is, for each multiedge $\bar{e}$, use $\bar{e}$, $\{u(e)\}_{e\in\bar{e}}$, $\{c(e)\}_{e\in\bar{e}}$, $x|_{\bar{e}}$, α as the input in Algorithm 1 to obtain a set of r integer vectors $z^1_{\bar{e}}, z^2_{\bar{e}}, \ldots, z^r_{\bar{e}}$ with coordinates for each edge $e \in \bar{e}$.

Step 2: Merge the sets of integer vectors obtained from multiedges on each biconnected non-crossing interval graph $G' = (V', E')$ on the cycle of G using Step 2 of Algorithm 4 separately and choose the least cost integer vector $z^*_{E'}$ out of the ones obtained for each non-crossing interval graph separately.

Step 3: For each rooted binary tree, use the merging along a tree method above to merge the sets of integer vectors obtained from the multichords. Let $z^{i^*}_F$ be the least cost integer vector obtained using the i^{th} tree. Let z^*_F be the union of these least cost integer vectors ($i.e$ $z^*_F(e) = 1$ iff $z^{i^*}_F = 1$ for some $i = 1, 2, \ldots, t$).

Step 4: Append z^*_F with all the least cost integer vectors $z^*_{E'}$ obtained for each biconnected non-crossing interval graph G' from step 2 to obtain and output an integer vector z^*_E defined on the entire edge set E.

Remark 6. Due to Remark 2, one can implement Algorithm 4 in polynomial time.

Let us analyze the properties of the integer vector that we obtain using this algorithm. Firstly, the properties that hold for arbitrary merging still continue to hold. Thus,

- $z^*_E(e) = 1$ for all i and every $e \in A$.

- $c(z^*_{E'}(E')) \leq \alpha c(x(E'))$ for every biconnected non-crossing interval graph G'. $c(z^{i*}_F(F) \leq \alpha c(x(F))$ for every $i = 1, 2, \ldots, t$ so that $c(z^*_F(F)) \leq t \cdot \alpha c(x(F))$. Combining these we get, $c(z^*_E) \leq t \cdot \alpha c(x)$.

Now consider any terminal bond $C \in \mathcal{C}$. We know from 3.1.1 that C consists of a terminal bond each (say C_1 and C_2) from two biconnected non-crossing intervals graphs on the cycle of G (say $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$) as well as a consecutive set of multichords (say $\bar{C}_3$). Then we have the following,

- For $j = 1, 2$, either $u_{A,C}(z^*_{E_j}(C_j \backslash A)) \geq D(A, C)$ or $u_{A,C}(z^*_{E_j}(C_j \backslash A)) \geq \alpha u_{A,C}(x(C_j \backslash A)) - D(A, C)$. This implies that either $u_{A,C}(z^*_E(C_1 \cup C_2 \backslash A)) \geq D(A, C)$ or $u_{A,C}(z^*_E(C_1 \cup C_2 \backslash A)) \geq \alpha u_{A,C}(x(C_1 \cup C_2 \backslash A)) - 2D(A, C)$

This follows immediately from the properties that we derived for biconnected non-crossing interval graphs using Algorithm 4. The implication follows because C_1 and C_2 are disjoint. We also have the following property for the multichords,

- For any node from a rooted binary tree T_i with label say $\bar{F}'$ (*i.e* the set of leaf-descendants of the node is $\bar{F}'$) such that $\bar{F}' \subseteq \bar{C}_3$, either $u_{A,C}(z^{i*}_F(\bar{F}' \backslash A)) \geq D(A, C)$ or $u_{A,C}(z^{i*}_F(\bar{F}' \backslash A)) \geq \alpha u_{A,C}(x(\bar{F}' \backslash A)) - D(A, C)$. This implies that for any arbitrary q nodes from the set of rooted binary trees $T_1, T_2, \ldots, T_t$ with labels say $\bar{F}'_1, \bar{F}'_2, \ldots, \bar{F}'_q$ such that $\bar{F}'_j \subseteq \bar{C}_3$ for all j and $\bar{F}'_j$ are pairwise disjoint, either $u_{A,C}(z^*_E(\cup_{j=1}^q \bar{F}'_j \backslash A)) \geq D(A, C)$ or $u_{A,C}(z^*_E(\cup_{j=1}^q \bar{F}'_j \backslash A)) \geq \alpha u_{A,C}(x(\cup_{j=1}^q \bar{F}'_j \backslash A)) - qD(A, C)$

This also follows immediately from the merging algorithm 3. The implication follows because we assumed that the labels of the nodes, $\bar{F}'_j$, are pairwise disjoint.

These properties along with the fact that terminal bonds in outerplanar graphs contain a consecutive set of multichords allow us to conclude the following: Let us say we can construct t rooted binary trees $T_1, T_2, \ldots, T_t$ whose set of leaves are labelled $\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_k$ and whose other nodes are labelled using the set of their leaf-descendants such that, for any set of consecutive multichords $\bar{F}_i^j = \{\bar{e}_i, \bar{e}_{i+1}, \ldots, \bar{e}_j\}$, we can find a total of at most q nodes from the trees with labels (say $\bar{F}'_1, \bar{F}'_2, \ldots, \bar{F}'_q$) such that $\cup_{l=1}^q \bar{F}'_l = \bar{F}_i^j$ and $\bar{F}'_l$ are pairwise disjoint. Then the output z_E^* obtained from algorithm 5 using trees $T_1, T_2, \ldots, T_t$ satisfies: $c(z_E^*) \leq t \cdot \alpha c(x)$ and for any terminal bond C, either $u_{A,C}(z_E^*(C \backslash A)) \geq D(A, C)$ or $u_{A,C}(z_E^*(C \backslash A)) \geq \alpha u_{A,C}(x(C \backslash A)) - (q+2)D(A, C)$. Now if x satisfies the KC inequalities for the set $A^{x,\alpha}$ i.e. if $u_{A,C}(x(C \backslash A)) \geq D(A, C)$ for every bond C and we set $\alpha = q+3$, then $c(z_E^*) \leq t(q+3)c(x)$ and $u_{A,C}(z_E^*(C \backslash A)) \geq D(A, C)$ which implies $u(z_E^*(C \backslash A)) \geq D(A, C)$. Additionally since $z_E^*(e) = 1$ for every $e \in C \cap A$, $u(z_E^*(C \cap A)) \geq D(C) - D(A, C)$ so that $u(z_E^*(C)) \geq D(C)$ for every bond C. Thus, z_E^* is feasible to the original problem. We can run the ellipsoid method to obtain an $O(tq)$-approximation algorithm for CapNDP on outerplanar graphs with demands occurring only on terminal nodes. The objective now should be to minimize the product $t \cdot q$ and we tackle this design problem in the next chapter.

Chapter 4

Exact Range Cover Problem

4.1 Problem Description

Let us start by formally defining the design problem introduced towards the end of Chapter 3 with regards to optimizing the approximation factor for CapNDP on outerplanar graphs. We call this problem the *Exact Range Cover Problem.* Here we are given an ordered set of n leaves say $L = \{1, 2, \ldots, n\}$ and are interested in constructing rooted binary trees on this same leaf set. Given a set of rooted binary trees on L, $\mathcal{T} = \{T_1, T_2, \ldots, T_t\}$ and a node v from some tree T_i, define its *leaf-subset* as the set of leaves in the binary sub-tree of T_i rooted at v and denote it using $L_{\mathcal{T}}(v)$. Now given any subset of the leaf set $L' \subseteq L$, we say that a set of nodes $V = \{v_1, v_2, \ldots, v_q\}$ chosen from the binary trees in $\mathcal{T}$ *exactly covers* L' if *(i)* $\cup_{i=1}^{q} L_{\mathcal{T}}(v_i) = L'$ and *(ii)* $L_{\mathcal{T}}(v_i)$ are pairwise disjoint. As seen in chapter 3, we are interested in exactly covering subsets that are a consecutive set of leaves for example $\{i, i+1, \ldots, j\}$. We shall call such subsets of L as *ranges* denoted I.

Now, given any set of rooted binary trees $\mathcal{T} = \{T_1, T_2, \ldots, T_t\}$ on the leaf set L, define the following terms,

- The *size* of $\mathcal{T}$ denoted $t(\mathcal{T})$ is the number of trees in $\mathcal{T}$

- The *query time* of $\mathcal{T}$ denoted $q(\mathcal{T})$ is defined as $\max_{\{I \subseteq L : I \text{ is a range}\}} \min\{|V| : V \text{ is a set of nodes from } \mathcal{T} \text{ that exactly covers } I\}$. Thus $q(\mathcal{T})$ is the maximum number of nodes needed to exactly cover any range.

- The *height* of $\mathcal{T}$ denoted $h(\mathcal{T})$ is defined as $\sum_{T \in \mathcal{T}} h(T)$ where $h(T)$ is the height of tree T or the largest size of a leaf to root path in T.

Different choices of $\mathcal{T}$ result in various trade-offs between these three factors. For the purposes of CapNDP on outerplanar graphs as exhibited in chapter 3, we are interested in finding $\mathcal{T}$ that minimizes the product $t(\mathcal{T}) \cdot q(\mathcal{T})$. We will now describe the construction of $\mathcal{T}$ that gives an $O((\log \log n)^2)$ bound for this product.

4.2 Construction of Rooted Binary Trees

Consider first, the case when $\mathcal{T}$ contains just one tree $T_{complete}$, which is the complete binary tree (see Figure 4.1). We define the levels of $T_{complete}$ recursively. Level 0 consists of the leaves of the tree and level i consists of those nodes whose children are from level $i-1$. It is clear that $t(\mathcal{T}) = 1$ and $h(\mathcal{T}) = \log n$. We prove below that $q(\mathcal{T}) \leq 2 \log n$

Lemma 4.2.1. *If $\mathcal{T} = \{T_{complete}\}$, then $q(\mathcal{T}) \leq 2 \log n$*

Proof. Let $I \subseteq L$ be an arbitrary range. Color a node v of $T_{complete}$ green if $L_{\mathcal{T}}(v) \subseteq I$. Let V_I be a least-size set of nodes of $T_{complete}$ that exactly covers I. Clearly every node $v \in V_I$ is colored green. We claim that if $v \in V_I$, then the immediate parent of v in $T_{complete}$ is not colored green. This claim can be proven by contradiction: let w be the parent of v with children v and u. Suppose w is coloured green, then to exactly cover I, V_I contains v along with certain nodes from the subtree rooted at u. However all these nodes could have been replaced by just choosing w itself thus giving a smaller cover. We now show that V_I contains at most two nodes from each level of $T_{complete}$. For the sake of contradiction, suppose V_I contained at least three nodes from a certain level say i. Let three of those nodes in order from left to right be u, v, w. Since I is a set of consecutive leaves, the node to the immediate left of v from level i is also coloured green and similarly the node to

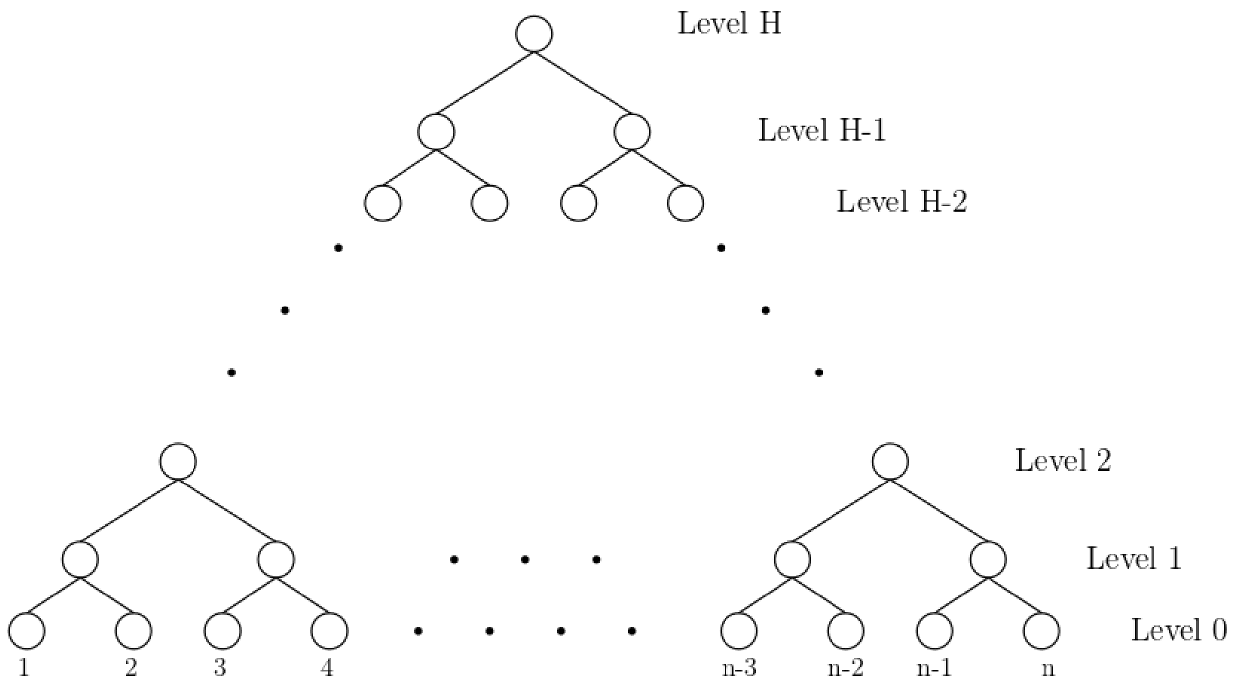

Figure 4.1: Complete binary tree $T_{complete}$

its immediate right. But then consider the immediate parent of v. Both its children are coloured green and so it too is coloured green. This contradicts our earlier claim. Hence V_I contains at most two nodes from each level and $|V_I| \leq 2 \log n$. Since I was arbitrary, the proof is complete. □

Thus by setting $\mathcal{T} = \{T_{complete}\}$, we obtain $t(\mathcal{T}) \cdot q(\mathcal{T}) \leq 2 \log n$ and so we obtain an $O(\log n)$-approximation algorithm for CapNDP on outerplanar graphs. This by itself is an improvement over $O(\beta(G))$ and we can do even better by constructing multiple binary trees.

4.2.1 Using Multiple Trees

Consider first, the following two trees called the Left-to-Right Tree (T_{LtoR}) and the Right-to-Left Tree (T_{RtoL}). In the descriptions below, γ is a user-defined integer parameter and the factors $t(\mathcal{T}), q(\mathcal{T}), h(\mathcal{T})$, all depend on γ [1].

Construction of T_{LtoR}: The tree is constructed using "Layers". Layer 0 consists of all the leaves in order from left to right. We construct Layer $i+1$ from Layer i by taking the first two nodes and connecting them to a common parent. Then, we connect the third node of Layer i and this new parent to a second parent. Then, we connect the fourth node of Layer i and the second parent to a third parent and so on until the γ^{th} node is connected. The parent of the γ^{th} node corresponds to the first node of Layer $i+1$. We then repeat this process for the next γ nodes in Layer i and continue this way to obtain Layer $i+1$ (see Figure 4.2).

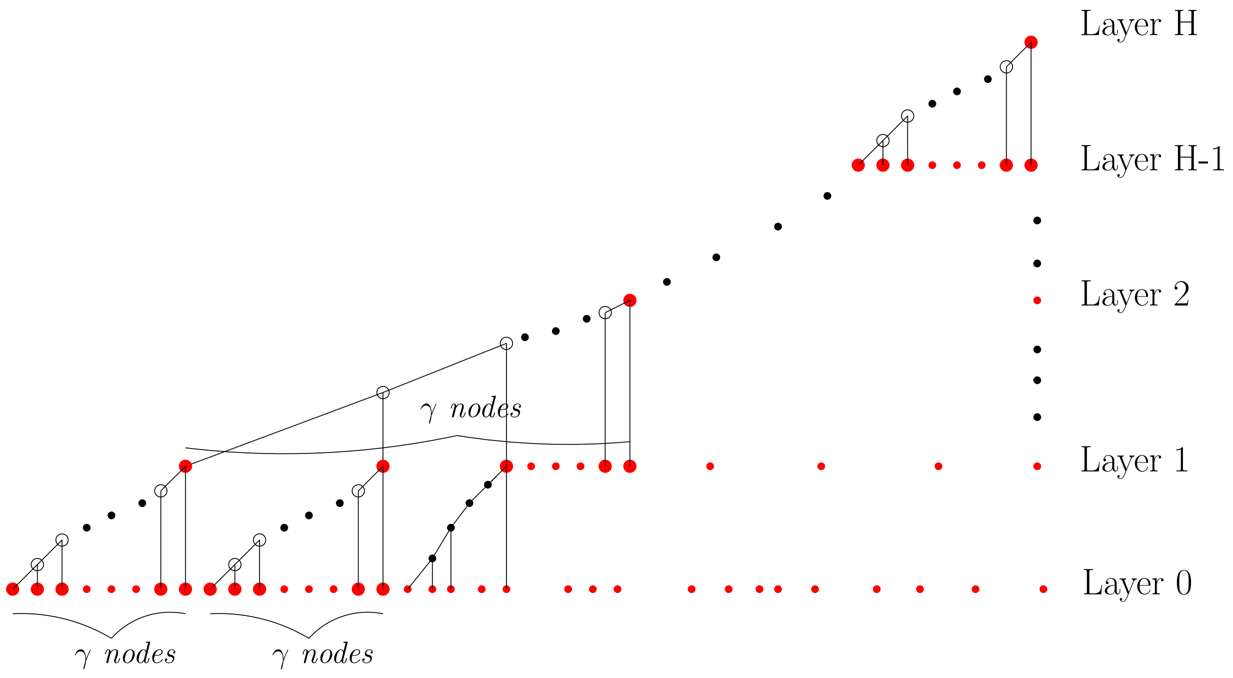

Figure 4.2: Construction of T_{LtoR}

[1] To avoid cumbersome notation, at various places, we set some integer quantity to an expression that need not be an integer without worrying about placing the floor or ceiling functions. This does not cause any issues, and all the calculations and analysis go through once we place the floors and ceilings as needed.

Construction of T_{RtoL}: The construction of T_{RtoL} is very similar. Again, we construct the tree using Layers. The only difference is that instead of connecting each set of γ nodes of layer i from left to right to obtain a node of layer $i+1$, we connect them from right to left. That is, we take the γ^{th} and the $(\gamma-1)^{th}$ nodes of layer i and connect them to a parent, then we connect the $(\gamma-2)^{th}$ node of layer i and this new parent to a second parent and so on. The parent of the first node corresponds to the first node of layer $i+1$ and then we repeat this process for the next γ nodes of Layer i and so on. Figure 4.3 below shows how each node of Layer $i+1$ is built using the nodes of Layer i.

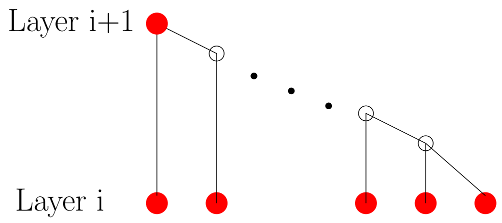

Figure 4.3: Construction of T_{RtoL}

Both trees are the same when viewed in layers: Consider the graph T'_{LtoR} obtained from T_{LtoR} as follows: The nodes of T'_{LtoR} are exactly the nodes that are on the Layers of T_{LtoR} and there is an edge between a node of Layer $i+1$ (say u) and Layer i (say v) if v is a descendant of u in T_{LtoR}. T'_{RtoL} is constructed similarly. We then have the following lemma,

Lemma 4.2.2. *The canonical identity map between T'_{LtoR} and T'_{RtoL} which maps the corresponding j^{th} nodes from Layer i to the other is a graph isomorphism*

Proof. It should be clear from the constructions of T_{LtoR} and T_{RtoL} that this is the case. Essentially we are taking the same γ nodes from Layer $i-1$ and just joining them in a different way to obtain the j^{th} node of Layer i in both the trees. $\square$

Exactly covering any range Pick an arbitrary range $I \subseteq L$. Let $\mathcal{T} = \{T_{LtoR}, T_{RtoL}\}$ and color the nodes of these trees as follows:

A node v is colored *green* if $L_{\mathcal{T}}(v) \subseteq I$. A node v is colored *blue* if $L_{\mathcal{T}}(v)$ intersects I as well as $L \backslash I$. Leave every other node uncoloured. An exact cover for I can only contain nodes that are colored green. Due to Lemma 4.2.2 above, the colours of nodes that lie on any layer are the same in both the trees, T_{LtoR} and T_{RtoL}.
Now since Layer 0 has $|I|$ colored nodes, all colored green, and the highest Layer consists of only one node which is in fact the root of each tree, we know that there exists a largest layer l which contains at least two colored nodes (blue or green). This implies that Layer $l+1$ has exactly one colored node and that every colored node in each tree is a descendant of this node. Let the colored nodes of Layer l in order from left to right be $v_1, v_2, \ldots, v_p$ (We can conveniently use the same name for the nodes of both the trees due to Lemma 4.2.2 above). Since I is a set of consecutive leaves, we have that $v_1, v_2, \ldots, v_p$ are also consecutive nodes from Layer l. Additionally, the nodes $v_2, \ldots, v_{p-1}$ have to be colored green and the nodes v_1 and v_p are either colored green or blue. Since $v_1, v_2, \ldots, v_p$ are descendants of the same node from Layer $l+1$, we know that $p \leq \gamma$. We are allowed to select the green nodes $v_2, \ldots, v_{p-1}$ to exactly cover a part of I and the only leaves in I left now will be $I_{left} := L_{\mathcal{T}}(v_1) \cap I$ and $I_{right} := L_{\mathcal{T}}(v_p) \cap I$. To exactly cover I_{right} we do the following:
Since I is a range, the descendants of v_p from Layer $l-1$ are colored in the following way: we have a set (possibly empty) of consecutive green nodes starting from the left most node followed by possibly one blue node and the rest after that are uncolored. However, the way we have constructed T_{LtoR} allows us to select just one node to exactly cover the consecutive green nodes of Layer $l-1$ and we are now left with at most one blue node of Layer $l-1$ which is dealt with in exactly the same manner (see Figure 4.4: the ticked nodes are what we would select in our cover).

Since we select at most one node from each layer, I_{right} can be covered by at most H nodes where H is the number of layers in T_{LtoR}. A simple calculation gives $H = \log n / \log \gamma$. I_{left} is exactly covered using the very same argument just using T_{RtoL} instead. Thus we need at most $2H$ nodes to exactly cover $I_{left} \cup I_{right}$ and the rest of I called I_{center} is covered using at most $p \leq \gamma$ nodes.

Before going further, let us pause and take stock of what we obtain using $\mathcal{T} = \{T_{LtoR}, T_{RtoL}\}$. Clearly $t(\mathcal{T}) = 2$ and the height of each tree is equal to the product

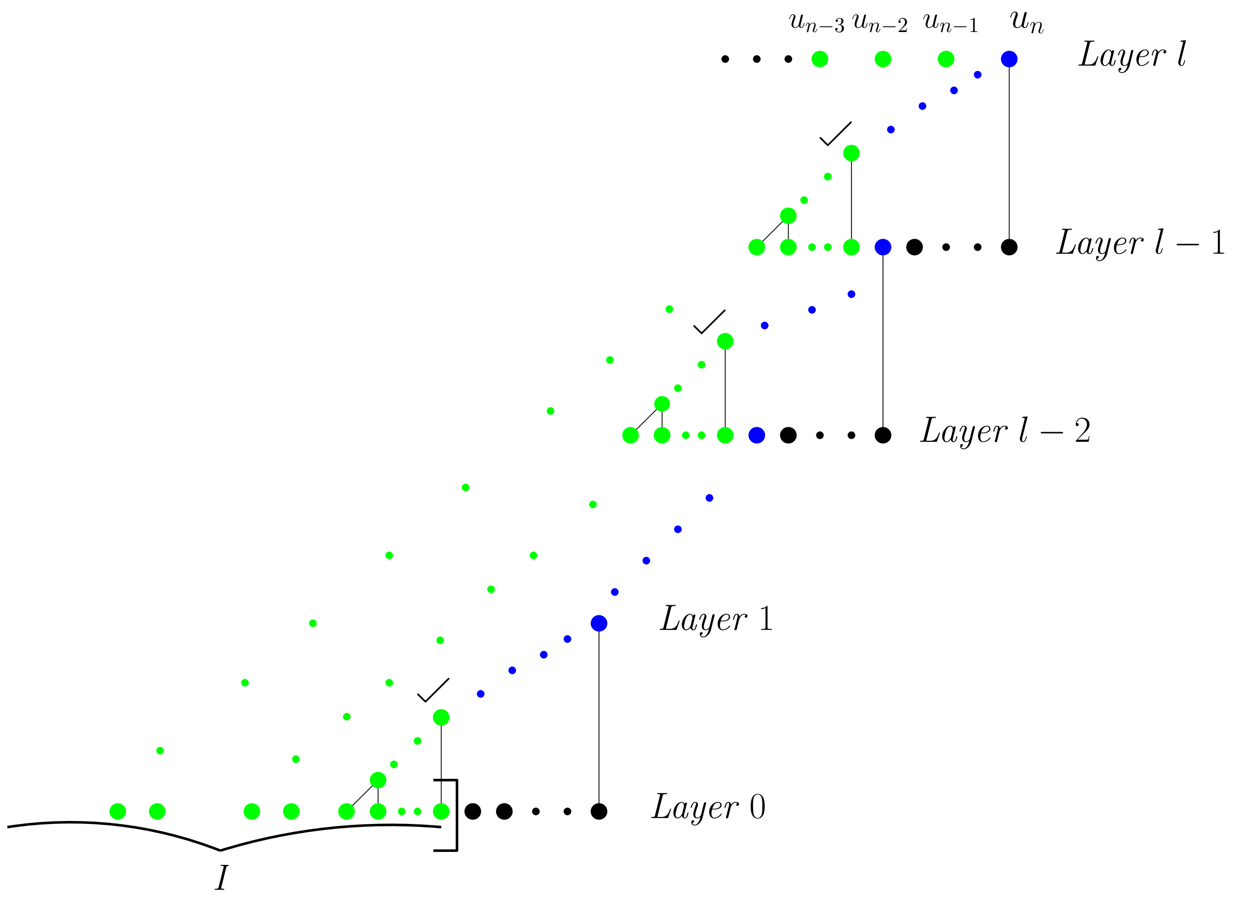

Figure 4.4: Exact cover for I_{right}

of the parameter γ with the number of layers H so that $h(\mathcal{T}) = 2\gamma \cdot \log n/\log\gamma$. Furthermore, the discussion above shows that $q(\mathcal{T}) \leq \gamma + 2H = \gamma + 2\log n/\log\gamma$. If we set $\gamma = \log n/\log\log n$, then $t(\mathcal{T}) \cdot q(\mathcal{T}) = O(\log n/\log\log n)$ and we obtain an $O(\log n/\log\log n)$-approximation algorithm for CapNDP on outerplanar graphs. This is an improvement from the previous case where $\mathcal{T} = \{T_{complete}\}$ and we can do even better by exactly covering I_{center} recursively.

Exactly covering I_{center} recursively Earlier, we exactly covered I_{center} using at most γ nodes by observing that $v_2, \ldots, v_{p-1}$ are all descendants of the same node from Layer

$l+1$ and are all colored green. Thus to exactly cover I_{center}, we need to exactly cover a consecutive set of at most γ nodes that are all descendants of the same node. We can use the previous idea recursively here by constructing another pair of trees which splits up each layer into sub-layers as follows: while building a node from a particular Layer i using γ nodes from Layer $i+1$, we can instead of connecting them using the left to right or right to left design, further divide them into sets of γ' nodes and then use the left to right or right to left design here. It is essentially the same as T_{LtoR} or T_{RtoL} just using γ' as the parameter while building the same nodes on the Layers as before (See figure 4.5).

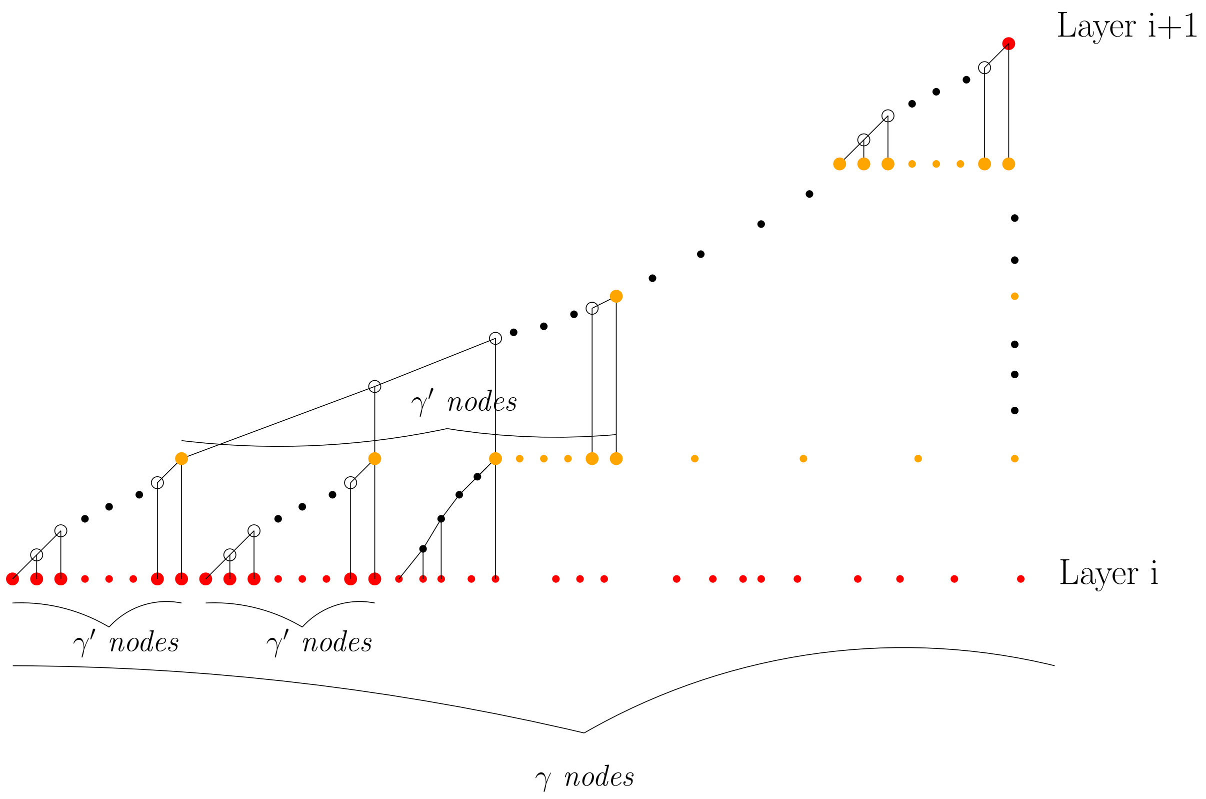

Figure 4.5: Recursively constructing trees

In so doing, we can now exactly cover I_{center} using at most $2H' + \gamma'$ nodes where H' is the number of sub-layers created between each pair of consecutive layers. The reason for this is exactly the argument provided earlier for I_{left} and I_{right}. A simple calculation gives $H' = \log \gamma / \log \gamma'$. We can further do this recursively by constructing another pair of trees to take care of the (at most) γ' green nodes that are consecutive descendants of the same node from a sub-layer. Thus we can consider a sequence $n = \gamma_0 \geq \gamma_1 \geq \gamma_2 \geq \gamma_3 \ldots \geq \gamma_{t/2}$ and construct rooted binary trees using these parameters as described above. Suppose $\mathcal{T}$ is the collection of all these rooted binary trees, then $t(\mathcal{T}) = t$ and as we have shown earlier, $q(\mathcal{T})$ is at most $2(\frac{\log n}{\log \gamma_1} + \frac{\log \gamma_1}{\log \gamma_2} + \cdots + \frac{\log \gamma_{t/2-1}}{\log \gamma_{t/2}}) + \gamma_{t/2}$.

Let us now concretely put some values to these $\gamma_i's$ and see how we perform. For any parameter $r \leq \log n$, choose $\gamma_i's$ such that $\gamma_{i+1}^r = \gamma_i$ or $\log \gamma_i / \log \gamma_{i+1} = r$ and stop at $\gamma_{t/2} = 2$. Let the collection of the rooted binary trees we obtain this way be $\mathcal{T}_r$. Then we have $2^{r^{t/2}} = n$ so that $t(\mathcal{T}_r) = 2 \log \log n / \log r$ and $q(\mathcal{T}_r) = 2(t/2 \cdot r + 1) = 2(\frac{r \log \log n}{\log r} + 1)$. The height h_1, of the first pair of trees created (using parameter γ_1) is equal to γ_1 times the number of layers created and is thus equal to $\gamma_1 \cdot \log n / \log \gamma_1 = n^{1/r} r$. We can calculate the height h_2, of the next pair of trees created (using parameter γ_2) by observing that the difference in the two trees (with parameters γ_1 and γ_2) is that instead of connecting the γ_1 nodes from left to right to or right to left directly, we are further dividing it into γ_2 batches. Thus $h_2 = h_1/\gamma_1 \cdot \gamma_2 \log \gamma_1 / \log \gamma_2 = n^{1/r^2} r^2$. Continuing this way, the height h_i, of the i^{th} pair of trees created (using parameter γ_i is $n^{1/r^i} r^i$. Since $n = 2^{r^{t/2}}$, $h_i = 2^{r^{t/2-i}} r^i$ and so $h(\mathcal{T}_r) = 2 \sum_{i=1}^{t/2} 2^{r^{t/2-i}} r^i$. Let us try to analyze the order of $h(\mathcal{T}_r)$ using the lemma below.

Lemma 4.2.3. *Let n be a large enough integer and $r \leq \log n$. Let $t/2$ be such that $n = 2^{r^{t/2}}$. Then, $2^{r^{t/2-i}} r^i \leq 2^{r^{t/2-1} - (i-1)} r$ for any $i = 1, 2, \ldots, t/2$.*

Proof. We start with the inequality $\log \frac{\log n}{2r} + \frac{\log \log n}{\log r} \leq \frac{\log n}{r+2}$ which is true for large enough n since the terms on the left hand side of the inequality have an additional logarithmic factor. We get the following set of implications then.

$$\begin{aligned}
&\log \frac{\log n}{2r} + \frac{\log\log n}{\log r} \le \frac{\log n}{r+2} \\
&\implies \log \frac{\log n}{2r} + t/2 \le \frac{\log n}{r+2} && (t/2 = \frac{\log\log n}{\log r}) \\
&\implies (r+2)\log \frac{2^{t/2}\cdot r^{t/2}}{2r} \le \log n && (r^{t/2} = \log n) \\
&\implies ((2r)^{(t/2-1)})^{(r+2)} \le n \\
&\implies ((2r)^{(i-1)})^{(r+2)} \le n && (i \le t/2) \\
&\implies ((2r)^{(i-1)})^{(r^i/(r^{i-1}-1))} \le n && (r^i/(r^{i-1}-1) \le r+2) \\
&\implies ((2r)^{(i-1)})^{r^i} \le (2^{r^{t/2}})^{r^{i-1}-1} && (2^{r^{t/2}} = n) \\
&\implies (2r)^{(i-1)} \le 2^{r^{t/2-i}(r^{i-1}-1)} = 2^{r^{t/2-1}}/2^{r^{t/2-i}} \\
&\implies 2^{r^{t/2-i}} r^i \le 2^{r^{t/2-1}-(i-1)} r
\end{aligned}$$

□

Now due to Lemma 4.2.3, we obtain that $h(\mathcal{T}_r) = 2\sum_{i=1}^{t/2} 2^{r^{t/2-i}} r^i \le 2\sum_{i=1}^{t/2} 2^{r^{t/2-1}-(i-1)} r$ and this is a geometric sum. Hence $O(h(\mathcal{T}_r)) = O(2^{r^{t/2-1}} \cdot r) = O(n^{1/r} \cdot r)$. Table 4.1 below, summarizes these results.

t	q	h
$2\log\log n/\log r$	$(rt+2)$	$2\sum_{i=1}^{t/2} 2^{r^{t/2-i}} r^i = O(n^{1/r}\cdot r)$

Table 4.1: Parameters of $\mathcal{T}_r$

Approximation factor for CapNDP We wish to minimize the product $t(\mathcal{T}) \cdot q(\mathcal{T})$ and this is obtained when $r = 4$. Hence, we obtain the following result.

Theorem 4.2.4. *There exists an $O((\log\log k)^2)$-approximation algorithm for CapNDP on outerplanar graphs with demands occurring only on terminal nodes where k is the number of multichords in the underlying outerplanar graph.*

4.3 Application to Array Range Query Problem

We will now explore an application of the *Exact Range Cover Problem.* Here we are given an array with n entries $a_1, a_2, \ldots, a_n$, each coming from a semi-group. A semi-group is a set G along with an associative binary product $(\cdot) : G \times G \to G$. The user is allowed to update the entries of the array and also query the product of any range $[i, j]$ which is just $a_i \cdot a_{i+1} \cdot \ldots \cdot a_j$. We wish to design a data structure that efficiently allows these two operations. This problem has been studied with various modifications and is called the *array range query problem* or the *range-sum query problem.* A survey of related results can be found in [26].

We can use the exact range cover problem and the results we developed therein to provide reasonable data structures for the array range query problem. Essentially the leaves of each rooted binary tree that we construct stores the values from the array and every other node stores the value obtained by taking the product of its children. Lets say we fix a particular r to use and have constructed $\mathcal{T}_r$. Each tree in $\mathcal{T}_r$ has exactly $2n-1$ nodes since they are rooted binary trees with n leaves. Hence the space required by this data structure is $O(nt(\mathcal{T}_r)) = O(n \log\log n / \log r)$. The time required to implement an update using this data structure is exactly equal to the height of $\mathcal{T}_r$, $h(\mathcal{T}_r) = 2\sum_{i=1}^{t/2} 2^{r^{t/2-i}} r^i = O(n^{1/r} \cdot r)$. Now suppose we are given a query in the form of a range $[i, j]$. We have already seen that we need at most $q(\mathcal{T}_r)$ nodes from $\mathcal{T}_r$ to exactly cover this range. Furthermore, we can find these nodes in $O(q(\mathcal{T}_r))$ time as follows: As shown in Figure 4.4, we need to pick at most $\log\gamma_i / \log\gamma_{i+1} = r$ nodes from each tree and these nodes can be found in $O(r)$ by referring to Figure 4.4. We start with the rightmost green node from Layer 0, say u and go up the tree layer by layer coloring the nodes that are ancestors of u. This takes time $O(r)$ as there are r layers. Next, we traverse back down the tree layer by layer and while doing so, if we encounter a blue colored node v from Layer i, we will have to select the parent of the node to the immediate left of v in Layer i. On the other hand, if we encounter a green colored node, we select it immediately and it will be the last node we select from that particular tree. This also takes time $O(r)$ as there are r layers. Hence any range query can be answered in time $O(rt(\mathcal{T}_r)) = O(q(\mathcal{T}_r))$. Thus we get an entire family of data structures with various trade-offs between the update time and the query time using different values of r. For example, if $r = \log n$, then we obtain $O(n)$ space complexity, $O(\log n)$ update time and $O(\log n)$ query time. On the other hand if $r = c$ for some constant c, then we obtain $O(n \log \log n)$ space complexity, $O(n^{1/c})$ update time and $O(\log \log n)$ query time. Table 4.2 below summarizes these results and here $t = 2 \log \log n / \log r$

Space	Query Time	Update Time
$O(n \log \log n / \log r)$	$O(r \log \log n / \log r)$	$O(n^{1/r} \cdot r)$

Table 4.2: Array Range Query using $\mathcal{T}_r$

Chapter 5

Extensions

In this chapter, we will see algorithms that extend our previous results to more general cases. First, we shall allow for a wider range of demand pairs and not just terminal demands. Secondly, we shall see an extension to CapNDP on directed outerplanar graphs and show that all our results still hold there. Finally, we consider a generalization of CapNDP called column-restricted covering integer programs

5.1 Including More Bonds

We have already seen how to deal with all the terminal bonds and have provided an $O((\log\log k)^2)$-approximation algorithm for such bonds where k is the number of multi-chords in the underlying outerplanar graph. Referring back to Lemma 3.1.2, we notice that the only bonds left out then are bonds such that the nodes of either of the connected components separated by it is a set of consecutive nodes that lie entirely within one of the biconnected non-crossing interval graphs that forms the outer cycle of the biconnected outerplanar graph. We have already noted earlier that biconnected non-crossing interval graphs are also biconnected outerplanar graphs. Hence, we can run the entire algorithm 5 separately on each of these biconnected non-crossing interval graphs to include more bonds and hence allow for more nodes to have demands between them. Suppose z^*_E is the integer vector that we obtain by running algorithm 5 on the entire outerplanar graph and z'_E is the integer vector that we obtain by running algorithm 5 on each biconnected non-crossing interval graph separately and appending all of these together (along with 0's for all chords to obtain a vector defined on the entire edge set E). Further let z_E be the integer vector obtained by taking the union of z^*_E and z'_E. Then we know that $c(z^*_E) \leq O((\log\log k)^2)c(x)$ and $c(z'_E) \leq O((\log\log k')^2)c(x)$. Here, as before, x is the fractional vector used as an input into algorithm 5, k is the number of multichords in the underlying outerplanar graph and k' is now the maximum number of multichords in any of the biconnected non-crossing interval graphs that form the outer cycle. Furthermore z_E now satisfies all terminal bonds of the outerplanar graph as well as all outerplanar terminal bonds of the biconnected non-crossing interval graphs that form the outer cycle. The only bonds left now will be bonds such that the nodes of either of the connected components separated by it is a set of consecutive nodes that lie entirely within one of the biconnected non-crossing interval graphs that forms the outer cycle of one of the biconnected non-crossing interval graphs of the underlying outerplanar graph. We can thus further use algorithm 5 and include more bonds if we desire. Suppose we perform this iteration d times, then the approximation ratio we obtain is at most $O(d(\log\log n)^2)$ where n is the number of nodes in the underlying outerplanar graph since the number of multichords k is of order $O(n)$ at any step.

We would also like to point out that one can include all bonds and thus have arbitrary node demands by utilizing the randomized algorithm provided by Chekuri and Quanrud [12]. There they provide a $(\log\Delta_0 + \log\log\Delta_0 + O(1))$-approximation ratio for arbitrary integer covering problems where Δ_0 is the maximum number of non-zero entries in any column of the constraint matrix. Due to Lemma 3.1.2, we know that there are at most $O(n^2)$ bonds in biconnected outerplanar graphs so that $\Delta_0 = O(n^2)$. Also, due to the description of terminal bonds of outerplanar graphs (see 3.1.1), we know that $\Delta_0 = \Omega(n^2)$ as a particular edge can be part of $\Omega(n^2)$ bonds. Hence the randomized algorithm provided in [12] gives an $O(\log(n))$-approximation algorithm for CapNDP on outerplanar graphs with arbitrary demand pairs.

5.2 Directed Graphs

We shall now explore how our results perform in the setting of Capacitated Network Design Problem on directed graphs. The problem statement is still exactly the same as Problem 2. The definition of bonds will change slightly as follows,

Definition 5.2.1 (Bond of a Directed Graph)**.** Given a directed multigraph G, for every ordered pair of nodes (u, v) a minimal set of edges whose removal disconnects v from u is called a bond of the graph.

With this definition of bonds of a graph, the IP formulation, CapNDP-IP and the corresponding KC-LP relaxation, CapNDP-KCLP still remain the same. Let us now see Lemma 3.1.2 in the context of directed graphs.

Lemma 5.2.2. *Bonds of a directed biconnected outerplanar graph $G = (V, E)$ can be written as $\delta^{out}(S)$ where S is a set of consecutive nodes on the outer cycle of the outerplanar graph and $\delta^{out}(S)$ is the set of edges directed from S to $V \backslash S$*

Proof. Since a bond C that disconnected two nodes, say v from u is also a cut, we can express C as $\delta^{out}(S)$ for some set $S \subseteq V$. Since all nodes of a biconnected outerplanar graph occur on the outer cycle, we can express S as the disjoint union of maximal sets of consecutive nodes around the outer cycle. Thus $S = S_1 \cup S_2 \cup \ldots \cup S_r$ where each S_i is a set of consecutive nodes around the outer cycle and there is a non-empty set of consecutive nodes T_i between S_i and S_{i+1} (T_r is between S_r and S_1). Now choose an S such that $C = \delta^{out}(S)$ with minimum r. Suppose $r = 1$, then there is nothing to prove. If not, WLOG let $u \in S_1$. Also, either $v \notin T_1$ or $v \notin T_r$. WLOG let $v \notin T_r$. Now consider the following list of exhaustive cases.

<u>*Case 1:*</u> There exists an undirected path from S_1 to S_r. Then since the graph is outerplanar, there cannot exist an edge between T_r and any other T_i. Consider then $S' = S \cup T_r$. Then $\delta^{out}(S')$ is still a $u - v$ cut and further $\delta^{out}(S') \subseteq \delta^{out}(S)$ since there is no edge from T_r to any other T_i. Since $\delta^{out}(S) = C$ is a bond (a minimal set of edges), we must have that $\delta^{out}(S') = \delta^{out}(S) = C$. But then S' has a smaller r than S and this is a contradiction.

Case 2: There is no undirected path from S_1 to S_r. Let $S' = \cup\{S_i : S_i$ is reachable from S_1 in the underlying undirected outerplanar graph$\}$. Then clearly S_r is disjoint from S' and $\delta^{out}(S')$ is a $u - v$ cut. Furthermore, $\delta^{out}(S') \subseteq \delta^{out}(S)$ since there is no edge from any of the $S_i's$ included in S' to any of the $S_j's$ not included in S'. Since $\delta^{out}(S) = C$ is a bond (a minimal set of edges), we must have that $\delta^{out}(S') = \delta^{out}(S)$. But then S' has a smaller r than S and this is a contradiction. □

Now that we have Lemma 5.2.2, everything that we have proven earlier will still go through with a few changes that we present here. First let's re-describe the terminal bonds of outerplanar graphs.

Terminal Bonds of Directed Biconnected Non-crossing Interval Graphs Given a directed non-crossing interval graph $G = (V, E)$ with n nodes say $1, 2, \ldots, n$, let E_{1n} be the set of edges directed from node 1 to n and let E_{n1} be the set of edges directed from node n to 1. Let $G_{1n} = (V, E_{1n})$ and $G_{n1} = (V, E_{n1})$ where both are undirected graphs. Then a bond separating n from 1 is just a bond in the undirected graph G_{1n} and a bond separating 1 from n is just a bond in the undirected graph G_{n1}. This follows immediately from Lemma 5.2.2.

The above description allows us to still achieve the 2-approximation for terminal bonds of directed non-crossing interval graphs by running algorithm 4 separately on G_{1n} and G_{n1}.

Terminal Bonds of Directed Biconnected Outerplanar Graphs Firstly the chords of a biconnected outerplanar graph can be ordered from s to t. Call the chords directed from the upper path to the lower path, *downward chords* and similarly call the chords directed from the lower path to the upper path, *upward chords*. Due to Lemma 5.2.2, we are interested in the bond generated by a set of consecutive nodes on the cycle that contains a terminal node. The edges in this bond contains a terminal bond from each of the biconnected non-crossing interval graphs that the end points of this consecutive set belong to and also contains either the set of consecutive upward chords or the set of consecutive downward chords that lie between these end points. Thus, we have a possibly empty set of consecutive upward or downward chords along with a terminal bond from one of the biconnected non-crossing interval graphs connected in series to the immediate left of the leftmost chord ($i.e$ from the non-crossing interval graphs that occur on either the upper or lower path to the left of the leftmost chord till the next chordal nodes on the left) and a terminal bond from one of the biconnected non-crossing interval graphs connected in series to the immediate right of the rightmost chord.

The above description allows us to still achieve the $O((\log\log k)^2)$-approximation for terminal bonds of directed outerplanar graphs (where k is the number of multichords) by running algorithm 5 on the downward chords and the upward chords separately and also running the part of algorithm 4 used in algorithm 5 separately on G_{1n} and G_{n1} for each biconnected non-crossing interval graph.

Furthermore, the remarks with regards to including more bonds in Section 5.1 still hold since we have proven Lemma 5.2.2.

5.3 Column-Restricted Covering Integer Programs

Column-Restricted Covering Integer Programs (CCIPs) generalize 0, 1- *covering integer programs* (0,1-CIPs). In a 0,1-CIP, the goal is to solve an integer program of the form

$$min\{c^T x : Ax \geq b, \quad x \in \{0,1\}\}$$

Here $A \in \{0,1\}^{m\times n}$ is the constraint matrix. $b \in \mathbb{Z}_+^m$ is the demand vector and $c \in \mathbb{Z}_+^n$ is the cost vector. $\{0,1\}$-CIPs are essentially equivalent to set-cover problems where sets correspond to columns and elements correspond to rows. It is known that $\{0,1\}$-CIPs cannot be approximated to a factor better than $O(\log n)$ unless $P = NP$. CCIPs are a capacitated version of $\{0,1\}$-CIPs in the sense that the constraint matrix A is now in $\mathbb{Z}_+^{m\times n}$ with the restriction that for each column, every non-zero entry is the same. The Capacitated Network Design Problem is an example of a CCIP. Given the hardness result for $\{0,1\}$-CIPs, it is reasonable to look for better approximation algorithms in cases where the constraint matrix A is structured. Chakrabarty, Grant and Könemann [8] initiated a systematic study of CCIPs by considering the underlying $\{0,1\}$-CIP and its *priority version*. If the underlying $\{0,1\}$-CIP has an integrality gap $O(\gamma)$ and its priority version has an integrality gap $O(\omega)$, then [8] show an $O(\gamma + \omega)$ approximation algorithm for the CCIP. Subsequently Chan, Grant, Könemann and Sharpe [10] built on these results and discovered a $O(1)$-approximation algorithm in the case where the constraint matrix A is a network matrix. This covers the case for example when the support of each column of A is a consecutive set of rows. Carr *et al.* [6] used their bucketing algorithm 1 to provide a p-approximation algorithm for general capacitated covering integer programs where p is the maximum number of non-zero entries in a row of the constraint matrix A. The bucketing algorithm 1 along with the merging algorithm 3 that we developed can also be used in situations where the constraint matrix A is structured. For example if the rows of A are given by paths from leaves to the root of a particular rooted tree where each node of this tree corresponds to a column of A (similar to terminal bonds of non-crossing interval graphs in 3.1.1), then we can provide a 2-approximation algorithm for the CCIP using algorithm 4. Similarly, if the supports of each row of A are a set of at most k sets of consecutive columns, then we can use the results developed for the chords of outerplanar graphs (algorithm 5) to obtain a $O(k(\log\log n)^2)$-approximation algorithm for the CCIP. We thus have the following theorem.

Theorem 5.3.1. *Let* $min\{c^T x : Ax \geq b, \quad x \in \{0,1\}\}$ *be a column-restricted covering integer program where* $A \in \mathbb{Z}_+^{m\times n}$. *Suppose the support of each row of the constraint matrix* A *is a set of at most* k *sets of consecutive columns, then there exists a* $O(k(\log\log n)^2)$*-approximation algorithm for the CCIP.*

We can also consider a combination of the two structures above as seen in the case of outerplanar graphs. Thus the methods developed in this thesis can be used for CCIPs where the constraint matrix A is structured.

Chapter 6

Conclusions and Future Work

In this thesis, we considered the *Capacitated Network Design Problem* arising in network security and presented an approximation algorithm for CapNDP on outerplanar graphs. Prior to our work, the best known approximation ratio of an approximation algorithm for this problem was $O(n)$ where n is the number of nodes in the outerplanar graph. We were able to improve this ratio by a doubly exponential factor by restricting the set of nodes that can hold positive demands, while capturing the single demand pair case. Here, we were able to achieve an approximation ratio of $O((\log\log n)^2)$. We designed a new algorithm called the merging algorithm which builds on the bucketing algorithm by Carr *et al.* [6] to achieve this. We showed that the merging algorithm is a versatile tool and used it to also achieve a 2-approximation algorithm for CapNDP on another class of graphs called non-crossing interval graphs generalizing line graphs. We were also able to generalize our results to CapNDP on directed graphs. Furthermore, we observed that our merging algorithm can be used in a much larger class of problems called *column-restricted covering integer programs* to achieve better approximation ratios there if the constraint matrix is suitably structured. Along the way, we encountered a combinatorial design problem which finds applications in the *array range query problem* and provided interesting results there. Our work and ideas lead to various other interesting questions and we mention some of them below.

FPTAS for CapNDP on outerplanar graphs with a single demand pair Carr *et al.* [6] had provided a pseudo-polynomial time algorithm via dynamic programming for CapNDP on outerplanar graphs with a single demand pair. However, we observed that their dynamic program cannot be used to design an FPTAS for the problem. Attempts were made during our work to find a dynamic program that can be used to design an FPTAS for the problem but no results were obtained. It will be interesting to see such a result or a result which proves that there is no FPTAS for this problem unless $P = NP$.

Including more demand pairs As we observed in section 5.1, our approximation algorithm takes care of most demand pairs from the outerplanar graph. The only pairs of nodes left out correspond to bonds of non-crossing interval graphs that are not terminal bonds. Due to the nice structure of non-crossing interval graphs, it is possible that the merging algorithm (or a different idea) can be adapted to work for all bonds of non-crossing interval graphs. If such is the case, one would be able to remove the restriction on the demand pairs from Theorem 4.2.4.

Improving the approximation ratio In Chapter 4, we describe the *exact range cover problem* and provide a solution to the problem. We have however not proven that the solution we provide is the optimal solution. It is possible that there are better solutions to this combinatorial design problem. If such is the case, one would be able to improve the approximation factor in Theorem 4.2.4. One would also improve our results on the *array range query problem* and on the *column-restricted covering integer programs*.

Applications of Exact Range Cover Problem In Chapter 4, we show how our solution to the *exact range cover problem* finds applications in the well-studied *array range query problem*. It is possible that there are other applications of our combinatorial design and it would be interesting to see such results. For example, our results should be extendable to the *multi-dimensional array range query problem* arising in image processing.

Applications of Merging Algorithm We have shown the versatility of our merging algorithm by applying it in the problem of $CapNDP$ on outerplanar graphs, $CapNDP$ on non-crossing interval graphs and *column-restricted covering integer programs*. The merging algorithm exploits the structure of the underlying graph or constraint matrix in these cases. It is possible that the merging algorithm can be adapted to work for other classes of graphs or constraint matrices and it would be interesting to see such results.

References

[1] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[2] Egon Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164, 1975.

[3] Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 1539–1545, 2010.

[4] Suman K. Bera, Shalmoli Gupta, Amit Kumar, and Sambuddha Roy. Approximation algorithms for the partition vertex cover problem. *Theoretical Computer Science*, 555:2 – 8, 2014.

[5] Tim Carnes and David Shmoys. Primal-dual schema for capacitated covering problems. In *Integer Programming and Combinatorial Optimization*, pages 288–302, 2008.

[6] Robert D. Carr, Lisa K. Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 106–115, 2000.

[7] Deeparnab Chakrabarty, Chandra Chekuri, Sanjeev Khanna, and Nitish Korula. Approximability of capacitated network design. In *Integer Programming and Combinatoral Optimization*, pages 78–91, 2011.

[8] Deeparnab Chakrabarty, Elyot Grant, and Jochen Könemann. On column-restricted and priority covering integer programs. In *Integer Programming and Combinatorial Optimization*, pages 355–368, 2010.

[9] Deeparnab Chakrabarty, Ravishankar Krishnaswamy, Shi Li, and Srivatsan Narayanan. Capacitated network design on undirected graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 71–80, 2013.

[10] Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, page 1576–1585, 2012.

[11] Chandra Chekuri, Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R Salavatipour. Approximation algorithms for nonuniform buy-at-bulk network design. *SIAM Journal on Computing*, 39(5):1772–1798, 2010.

[12] Chandra Chekuri and Kent Quanrud. On approximating (sparse) covering integer programs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1596–1615. SIAM, 2019.

[13] Maurice Cheung, Julián Mestre, David B. Shmoys, and José Verschae. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. *SIAM Journal on Discrete Mathematics*, 31(2):825–838, 2017.

[14] Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: Better and simpler. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, page 169–178, 2011.

[15] Gu Even, Guy Kortsarz, and Wolfgang Slany. On network design problems: Fixed cost flows and the covering steiner problem. In *Algorithm Theory — SWAT 2002*, pages 318–327, 2002.

[16] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, page 223–232, 1994.

[17] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 149–162, 2009.

[18] MohammadTaghi Hajiaghayi, Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. Combinatorial algorithms for capacitated network design, 2011.

[19] Mike Hewitt, George L Nemhauser, and Martin WP Savelsbergh. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22(2):314–325, 2010.

[20] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.

[21] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

[22] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[23] Sven O Krumke, Hartmut Noltemeier, S Schwarz, H-C Wirth, and R Ravi. Flow improvement and network flows with fixed costs. In *Operations Research Proceedings 1998*, pages 158–167, 1999.

[24] Adam Kurpisz, Samuli Leppänen, and Monaldo Mastrolilli. A lasserre lower bound for the min-sum single machine scheduling problem. In *Algorithms - ESA 2015*, pages 853–864, 2015.

[25] F Sibel Salman, Joseph Cheriyan, Ramamoorthi Ravi, and Sairam Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM Journal on Optimization*, 11(3):595–610, 2001.

[26] Matthew Skala. *Array Range Queries*. Springer, 2013.

[27] Laurence A. Wolsey. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1):165–178, 1975.