Cross-monotonic Cost Sharing Methods for Network Design Games

by

David Wheatley

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

Waterloo, Ontario, Canada, 2007

©David Wheatley, 2007

Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis we consider some network design games that arise from common network design problems. A network design game involves multiple players who control nodes in a network, each of which has a personal interest in seeing their nodes connected in some manner. To this end, the players will submit bids to a mechanism whose task will be to select which of the players to connect, how to connect their nodes, and how much to charge each player for the connection. We rely on many fundamental results from mechanism design (from [8], [9] & [5]) in this thesis and focus our efforts on designing and analyzing cost-sharing methods. That is, for a given set of players and their connection requirements, our algorithms compute a solution that satisfies all the players' requirements and calculates 'fair' prices to charge each of them for the connection.

Our cost-sharing methods use a primal-dual framework developed by Agrawal, Klein and Ravi in [1] and generalized by Goemans & Williamson in [3]. We modify the algorithms by using the concept of death-time introduced by Könemann, Leonardi & Schäfer in [6].

Our main result is a 2-budget balanced and cross-monotonic cost sharing method for the downwards monotone set cover game, which arises naturally from any downwards monotone $0,1$-function. We have also designed a 2-budget balanced and cross-monotonic cost sharing method for two versions of the edge cover game arising from the edge cover problem. These games are special cases of the downwards monotone set cover game. By a result by Immorlica, Mahdian & Mirrokni in [4] our result is best possible for the edge cover game.

We also designed a cross-monotonic cost sharing method for a network design game we call the Even Parity Connection game arising from the T-Join problem that generalizes proper cut requirement functions. We can show our algorithm returns cost shares that recover at least half the cost of the solution. We conjecture that our cost sharing method for the even parity connection game is competitive and thus 2-budget balance.

Acknowledgments

Thank you to my advisor Jochen Könemann for his guidance and invaluable advice.

Thank you to my readers for assisting me in the preparation of this thesis.

Thank you to the Faculty of Math and Department of Combinatorics and Optimization for their wonderful programs and financial assistance.

Thank you to the support staff, especially Marg Feeney, who helped me numerous times throughout my tenure as a graduate student.

Thank you to my supportive family, without whom my post-secondary studies would not have been possible.

Finally, thank you to my wonderful fiancée whose patience and kind words helped me stay focused on my tasks.

Contents

1 Introduction 1
- 1.1 Network Design Problems 1
- 1.2 Network Design Games 3
- 1.3 Cost-Sharing Mechanisms 4
- 1.4 Primal Dual Algorithms 8
  - 1.4.1 AKR Algorithm 9
  - 1.4.2 GW Algorithm 11

2 Algorithms for Network Design Games 17
- 2.1 Death-time concept 18
- 2.2 Single-node Multi-player Edge-Cover Game 21
  - 2.2.1 Primal-dual method 22
  - 2.2.2 Analysis: Cost Recovery 24
  - 2.2.3 Analysis: Cross-monotonic Cost-Sharing Method 26
- 2.3 Multi-player Multi-node Edge Cover Game 27
  - 2.3.1 Primal-Dual Method 28
  - 2.3.2 Analysis: Cost Recovery 28
  - 2.3.3 Analysis: Cross-monotonic Cost-Sharing Method 29
- 2.4 Downwards Monotone Set Cover Game 30
  - 2.4.1 Downwards Monotone Cut Requirements 31
  - 2.4.2 A Lazy Primal-Dual Method 32
  - 2.4.3 Analysis: Cost Recovery 36
  - 2.4.4 Analysis: Cross-monotonic Cost-Sharing Method 40
- 2.5 Even Parity Connection Game 45
  - 2.5.1 Primal-Dual Method 46
  - 2.5.2 Analysis: Cost Recovery 48
  - 2.5.3 Analysis: Cross-monotonic Cost-Sharing Method 50

3 Discussion 53
- 3.1 Future Work 53
  - 3.1.1 Lifted Cut Relaxation 53
  - 3.1.2 Competitive Conjectures 56
- 3.2 Conclusions 56

Bibliography **59**

Chapter 1

Introduction

In the past decade many advancements have been made in the field of classical game theory, a broad research area that studies the behaviour of rational and selfish agents interacting in a well defined environment. This renewed interest has been largely driven by the explosive growth of the Internet, a complex network that brings together millions of users and computers each desiring to send, share and receive huge amounts of data. The problems of how to connect these users and how to efficiently transmit their data could be loosely defined as network design problems. This is a well studied area within the field of combinatorial optimization, which seeks to minimize the cost of accomplishing a specific goal given finite resource constraints.

For a motivating example, consider an Internet Service Provider (ISP) that wishes to provide cable Internet service to a remote town. Given a list of businesses and households in the town that wish to receive this service, the ISP would like to determine how to lay new cables to service the new customers while minimizing the cost of laying the cable. In the broad terms of combinatorial optimization, the service provider wishes to accomplish the goal of connecting all the new customers while minimizing the cost of laying the cable, constrained by building restrictions from local and regional governments.

1.1 Network Design Problems

The above problem is an example of a *network design problem.* We are given an undirected graph G with node set V and edge set E, where each edge $e \in E$ has a non-negative cost c_e. In this example, the edges represent feasible sites for new cable, the nodes represent homes, businesses and cable junctions, and the cost of each edge segment captures the cost of clearing land and the cable itself. We assume our graph includes one root node which represents a connection to the Internet through the ISP; it wouldn't do to just connect the townspeople to one another!

We now develop a mathematical model that captures all network design problems presented in this thesis. Consider our ISP example, and notice that any set S that includes a customer node but not the root node must have an edge crossing the cut $\delta(S)$ for the customer to receive service. To generalize this concept, we define a *cut requirement function* $f(S)$ which maps each subset of nodes $S \subseteq V$ to $\mathbb{N}$. In our example, any subset of nodes S that includes a customer that wishes to receive service and not the root node would have $f(S) = 1$. That is, at least one line of cable must cross the boundary of the set S connecting the customer to the Internet.

We can model and solve network design problems using linear programming. Consider the following integer programming formulation for the general network design problem:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e & & \text{(IP)} \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S) & \forall S \subseteq V, & \\ & x_e \in \{0,1\} & \forall e \in E. & \end{aligned}$$

We will see in Section 1.4 how to use this integer program and its dual to solve our network design problems.

The well known *Steiner Tree problem* falls into this framework. Here we are given an input graph $G = (V, E)$, non-negative edge costs c_e for each edge $e \in E$, a set of *terminals* $R \subseteq V$ and a root vertex $r \in V$. The goal is to connect all the terminals and the root node with a single tree of minimum cost. The appropriate cut requirement function has $f(S) = 1$ if and only if $S \subset (R \cup \{r\})$. Comparing the Steiner Tree problem to the example above, we can identify each interested customer and the root node as terminals, and observe that the ISP's problem of servicing all the customers at minimum cost is just the problem of finding the minimum cost Steiner tree that spans the customer nodes and the root node.

In our example, a feasible solution is a subset of edges $F \subseteq E$ forming a tree that spans every customer terminal and the root node. The optimal solution would be the feasible tree of minimum total cost. Goemans & Williamson explored the model of a general network design problem in [3], where a feasible solution is a subset of edges $F \subseteq E$ such that F crosses the cut induced by each S at least $f(S)$ times, i.e. $|\delta(S) \cap F| \geq f(S), \forall S \subseteq V$.

We will be considering a number of cut requirement functions in this thesis, restricting our focus to problems where the cut-requirement $f(S)$ is a $0,1$-function.

Definition 1. *A function* $f : 2^V \to \{0,1\}$ *satisfies* maximality *if for all disjoint* A, B *with* $f(A) = f(B) = 0$ *we have* $f(A \cup B) = 0$.

Notably, every cut requirement function in this thesis will satisfy the maximality property.

Definition 2. *A function f is* proper *if it satisfies both the maximality property and the symmetry property:* $f(S) = f(V \backslash S)$ *for all* $S \subseteq V$.

Another class of functions, from which we will draw our first few problems, is the class of downwards monotone functions.

Definition 3. *A function f is* downwards monotone *if* $\forall S, S' \subseteq V$ *where* $S' \subseteq S$ *we have* $f(S) \leq f(S')$.

It is necessary that we assume $f(V) = 0$ in every problem. Goemans & Williamson address the special case where $f(S)$ is a $0,1$-function and either proper or downwards monotone in [3]. Many of these network design problems, including the Steiner Tree problem, are NP-hard. Note that the Steiner Tree problem's cut requirement function is proper.

1.2 Network Design Games

We can make our example interesting from a game theoretic perspective by asking an obvious question: how much should each customer pay? ISPs are in the habit of charging a flat monthly fee to their customers, but due to the remoteness of the town it may not be cost-effective to charge the customers in the small town the same fee as those living in a large city. The ISP may want to pass on the excessive cost of laying cable to the customers who do use the service. It may also want to consider factors such as expected bandwidth usage, or more generally, how much each individual customer values access to the Internet. For instance, a household that wouldn't pay more than \$10 a month for cable Internet service is not a very desirable customer. On the other hand, the ISP would be losing out on revenue if it elected to charge \$50 a month to a business that was willing to pay upwards of \$200 a month.

A general network design game has a player set P, with $|P| = p \geq 1$ players. Each player $i \in P$ has their own cut requirement function f_i, and seeks to establish a forest $F \subseteq E$ that satisfies their cut requirement. In most games, we insist that a feasible F must satisfy all cut requirements $f_i, \forall i \in P$. For functions f_i satisfying the maximality property this is equivalent to satisfying the cut requirement $f(S) = \max_{i \in P} f_i(S)$. To see why, consider two disjoint sets A and B with $f_i(A) = f_i(B) = 0$ for all $i \in P$. Then by maximality, $f_i(A \cup B) = 0$ for all players, so $f(A \cup B) = 0$ as well for all disjoint sets A and B with $f(A) = f(B) = 0$.

From the perspective of the ISP we can define the set $P = \{1, ..., p\}$ of agents as the set of potential customers. In our example, we can identify each player with one node representing a household or business. We say that each player $i \in P$ has a private *utility* $u_i \geq 0$ for receiving the service, and utility 0 if they do not receive cable Internet service. The utility of a player can be interpreted as the most they are willing to pay for receiving the service. In our example, the cut requirement $f_i, \forall i \in P$ can be simply defined as $f_i(S) = 1$ if and only if S contains the node owned by player i and not the root node. That is, each player is only interested in ensuring they are connected to the root node. As all of these cut requirements satisfy the maximality property, a feasible solution to this network design game is also a feasible solution to the network design problem with cut requirement $f(S) = \max_{i \in P} f_i(S)$.

1.3 Cost-Sharing Mechanisms

If the ISP knew the utilities $\{u_i\}_{i \in P}$ of all the customers in the remote town, it could make educated decisions as to which customers are worth servicing; how to lay cable in order to service them; and how much to charge each customer. Unfortunately for the ISP, we assume these valuations are private. Instead, the ISP could solicit bids $\{b_i\}_{i \in P}$ from the players.

At this point we enter the field of mechanism design. In general, a *mechanism* is an algorithm that allocates resources among a set of agents. In this thesis we will be considering a smaller class of mechanism design problems where we try to allocate a service rather than resources.

Our ISP problem is a good example of a mechanism in action. The mechanism is tasked with three objectives:

1. Determine a set $Q \subseteq P$ that will receive the service;
2. Compute a solution $F \subseteq E$ to service Q;
3. Charge a price x_i to each player $i \in Q$.

A mechanism that accomplishes these goals is called a *cost-sharing mechanism.*

For notational convenience we define an indicator variable q_i such that $q_i = 1$ if $i \in Q$ and $q_i = 0$ otherwise. Define the *benefit* of player i as $u_i q_i - x_i$.

From the perspective of the players, the mechanism is a strategic game where we assume each player i is selfish, and seeks to maximize their benefit. We let B_i represent the set of all feasible bids for player $i \in P$. Let B_{-i} represent the set of vectors of feasible bids for the other players, and let the vector b_{-i} represent the chosen bids of all other players. Let $u_i(b_i, b_{-i})$ represent the benefit of player i following the outcome of the mechanism if she bids b_i given the other players bid b_{-i}.

Definition 4. *We say a bid $b_i \in B_i$ is a* dominant strategy *for player i if for all $b_{-i} \in B_{-i}$, we have $u_i(b_i, b_{-i}) \geq u_i(b'_i, b_{-i})$ for all $b'_i \in B_i$.*

An intuitive way to define this dominant strategy is as follows: regardless of the bids of the other players b_{-i}, player i cannot improve their benefit by instead submitting a new bid $b'_i \neq u_i$. A rational agent should always use a dominant strategy, if one exists at all.

Since the players are selfish, it may suit a player to lie about her utility u_i and submit a bid $b_i \neq u_i$. If this happens, it can only hurt the ISP. It could either lose money by not charging a customer enough, or could lose a customer entirely if the ISP decides the bid received is too low to bother providing service.

As a result, a very desirable property of cost-sharing mechanisms is that the dominant strategy of each player is to report their utility. This property is also desirable from the player's perspective as they can be confident they are using a simple yet optimal strategy.

Definition 5. *A cost-sharing mechanism is said to be* strategyproof *(or* truthful*) if bidding $b_i = u_i$ is a dominant strategy for each player $i \in P$*

We can generalize this concept to coalitions.

Definition 6. *Suppose the mechanism returns prices x and indicator vector q when every player $i \in P$ bids $b_i = u_i$, while it returns prices x' and indicator vector q' if a coalition of players $P' \subseteq P$ deviates from the truth and each player $i \in P'$ is permitted to bid some $b_i' \neq u_i$. We say the mechanism is* group-strategyproof *if for all $i \in P'$*

$$u_i q_i' - x_i' \geq u_i q_i - x_i$$

then for all $i \in P'$

$$u_i q_i' - x_i' = u_i q_i - x_i$$

In short, a mechanism is group-strategyproof if no member of a coalition can increase their benefit without decreasing the benefit of another member.

In [5] Jain & Vazirani describe a general model for cost-sharing mechanisms and list other desirable properties, including:

1. *No Positive Transfer (NPT)*: The prices are all non-negative ($x_i \geq 0, \forall i$)
2. *Voluntary Participation (VP)*: Each player i is never charged a price x_i greater than their bid b_i and only the players who receive service (those in Q) are charged a price;
3. *Consumer Sovereignty (CS)*: A player is only guaranteed to receive service if they make a large enough bid.

Voluntary Participation reflects the reality that someone who is being charged more for the service than they are willing to pay (i.e. has negative benefit) can instead choose not to play the game at all. Consumer Sovereignty prohibits the mechanism from guaranteeing service to players regardless of price, which encourages each player to bid their valuation instead of 0.

The service provider must charge prices to the players it services in order to recoup the cost of maintaining the network that services the players. Let the cost of servicing a player set Q be denoted by $c(Q)$. In our ISP example, if we are given edge costs c_e for each edge $e \in E$ then for a forest $F \subseteq E$ that services some subset of the customers Q we have that $c(Q) = \sum_{e \in F} c_e$. Note that while an optimal forest $F_{opt}(Q)$ certainly exists for servicing Q that the mechanism may not be able to find it in polynomial time. As mentioned earlier, some network design problems like the Steiner Tree problem are NP-Hard. As a result, we do not insist that our mechanism return an optimal solution, and instead relax the requirement to a solution that is at most a constant times the optimal solution. Let

$$opt_Q = \sum_{e \in F_{opt}(Q)} c_e$$

i.e. the cost of the optimal forest that services Q.

Once we have a solution that services Q, we seek to share the cost of the solution among the agents in Q. In the pursuit of fair prices, we try and design cost-sharing mechanisms that are *budget balanced*. Define a cost-sharing mechanism as *budget balanced* if:

$$c(Q) \leq \sum_{i \in Q} x_i \leq opt_Q$$

The first inequality is referred to as the cost recovery inequality and insists that the cost of servicing Q be recovered. The second inequality is referred to as the *competitiveness* inequality and promotes fairness by insisting the service provider not charge the players more than the optimal cost of servicing them. Since we are considering network design problems where we seek to minimize the objective function, a budget balanced mechanism naturally has $c(Q) = opt_Q$.

One final property we would like our mechanism to have is *efficiency*. We say a mechanism is *efficient* if the total benefit of the players in Q is maximized. Unfortunately, it is a classical result in game theory that no cost-sharing mechanism is both efficient and budget balanced. Recently, Roughgarden and Sundararajan [10] introduced an alternative measure of efficiency known as *Social Cost* that circumvents this difficulty somewhat. However, we will set this condition aside for the remainder of the thesis and focus on the remaining properties.

Now consider the problem of finding a budget balanced and group-strategyproof cost-sharing mechanism that also satisfies the NPT, VP and CS properties.

Immorlica, Mahdian and Mirrokni showed in [4] that the trivial cost-sharing mechanism that charges the entire cost $c(Q)$ to a somewhat arbitrary player (whose bid is at least $c(Q)$ for some $Q \subseteq P$) and nothing to the rest of the players results in a budget balanced and group-strategyproof cost-sharing mechanism. While this mechanism technically satisfies the properties listed above, it is undesirable for the following reasons. The first problem is that most of the players who do receive service do so without paying for it; we would like a cost-sharing mechanism that avoids free riders (those that pay nothing) whenever possible. The second problem is that a coalition of players could convince a free riding member $i \in Q$ whose utility is 0 to not submit a bid at all. Let $Q' = Q - \{i\}$. This could result in a lower cost of service $c'(Q') < c(Q)$ charged to the one player bearing the cost without changing the benefit of player i. The result is one member of the coalition increases their benefit (by paying less) while the player whose utility is 0 experiences no change in benefit. The mechanism is technically still group-strategyproof, but only in the way we defined earlier. It does not meet a stronger notion of group-strategyproofness suggested by Moulin in [8]. The authors suggest allowing utilities and bids to be negative so that the above mechanism is not a feasible alternative. The remainder of this thesis will take this approach so that we can focus our attention on 'sensible' cost-sharing mechanisms.

This brings us to a general cost-sharing mechanism developed by Moulin & Shenker in [8] & [9], sometimes referred to as a Moulin mechanism. This mechanism is a group-strategyproof cost-sharing mechanism that arises from any *cross-monotonic cost-sharing method* ξ.

Definition 7. *Given* $Q \subseteq P$*, define a* cost sharing method ξ *as an algorithm that returns a solution to service* Q *and prices* $\xi_Q(i) \in \mathbb{R}^+$ *(cost shares) such that* $\xi_Q(i) = 0$ *if* $i \notin Q$ *and* $\sum_{i \in Q} \xi_Q(i) = c(Q)$*.*

Intuitively, a cost-sharing mechanism selects a subset $Q \subseteq P$ and then uses a cost-sharing method to compute the solution F and prices $x_i = \xi_Q(i)$.

Definition 8. *A cost share is cross-monotonic if for sets* R *and* S *with* $R \subseteq S$ *and each player* $j \in R$ *we have*

$$\xi_S(j) \leq \xi_R(j)$$

That is, the cost share of each player should not increase if other players join the game; or analogously, the cost share of each player should not decrease if a player leaves the game.

Algorithm 1 Moulin Mechanism

```
Given: A cost-sharing method $\xi$, the set of players $P$ and their bids $\{b_i\}_{i \in P}$
$Q \leftarrow P$
repeat
    Compute $\xi_Q(i)$ using the cost-sharing method $\xi$
    $R = \{i \in Q : \xi_Q(i) > b_i\}$
    $Q \leftarrow Q \backslash R$
until $R = \emptyset$
Return: $x_i = \xi_Q(i), \forall i \in P$
```

Moulin & Shenker also showed that if the cost-sharing method ξ is budget-balanced then the Moulin mechanism is budget-balanced. However, if the underlying problem of finding the optimal solution that services Q is NP-hard then it is unlikely that a cost-sharing mechanism exists that is budget balanced and can compute prices in polynomial time.

We thus relax the budget balance requirement and focus our attention on finding cost-sharing mechanisms that are *approximately budget balanced.* Define a cost-sharing mechanism as *α-budget balanced* if:

$$\frac{1}{\alpha} c(Q) \leq \sum_{i \in Q} x_i \leq opt_Q$$

Just as we defined α-budget balanced for cost-sharing mechanisms we can define an *α-budget balanced cost-sharing method* ξ as one that satisfies:

$$\frac{1}{\alpha} c(Q) \leq \sum_{i \in Q} \xi_Q(i) \leq opt_Q$$

Jain & Vazirani in [5] extended the results of Moulin & Shenker with the following theorem

Theorem 1.3.1. *The Moulin mechanism is α-budget balanced, group-strategyproof and meets NPT, VP and CS if the underlying cost-sharing method ξ is cross-monotonic and α-budget balanced.*

We now turn to the problem of finding such cross-monotonic and α-budget balanced cost-sharing methods for some network design games by developing primal-dual approximation algorithms and cost-shares derived from dual solutions to these problems.

1.4 Primal Dual Algorithms

Recall the general linear program for network design problems we introduced earlier:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e & & \text{(IP)} \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S) & \forall S \subseteq V, & \\ & x_e \in \{0,1\} & \forall e \in E. & \end{aligned}$$

It is convenient to consider the linear programming relaxation (LP) of the above (IP), where the constraints $x_e \in \{0,1\}, \forall e \in E$ are replaced with the constraints $x_e \geq 0, \forall e \in E$.

Now consider the dual (D) of (LP), which has a dual variable y_S for each subset $S \subseteq V$ and a constraint for each edge $e \in E$:

$$\begin{aligned} \max \quad & \sum_{S \subseteq V} f(S) y_S & & \text{(D)} \\ \text{s.t.} \quad & \sum_{S \subseteq V : e \in \delta(S)} y_S \leq c_e & \forall e \in E, & \quad (1) \\ & y_S \geq 0 & \forall S \subseteq V. & \end{aligned}$$

There exist a few closely related primal-dual algorithms for solving general network design problems. Agrawal, Klein and Ravi first proposed a primal-dual algorithm (AKR) for the *Steiner Forest Problem* in [1]. In the Steiner Forest Problem we are given an undirected graph $G = (V, E)$, non-negative edge costs $c_e \forall e \in E$, and a set of $k > 0$ terminal pairs $R = \{(s_1, t_1), \ldots, (s_k, t_k)\} \subseteq V \times V$. A feasible solution to this problem is a forest $F \subseteq E$ such that the two vertices s_i, t_i of each terminal pair $(1 \leq i \leq k)$ are contained in the same tree of F. It is interesting to note that the Steiner Tree problem is just a special case of the Steiner Forest problem, where the root node $r \in \{s_i, t_i\}$ for all $1 \leq i \leq k$.

The appropriate cut requirement function for the Steiner Forest problem can be stated naturally in terms of Steiner cuts. A set $U \subseteq V$ is a *Steiner cut* if it contains exactly one vertex of an (s_i, t_i) pair, for some $1 \leq i \leq k$. We let $\mathcal{F}$ represent the set of all Steiner cuts. Now we can define the cut requirement function for the Steiner Forest problem as $f(S) = 1$ if and only if $S \in \mathcal{F}$, and $f(S) = 0$ otherwise.

1.4.1 AKR Algorithm

The AKR algorithm constructs a primal solution for (IP) and a dual solution for (D) with the cut requirement function f from the previous paragraph. The algorithm begins with an empty forest F and dual variables $y = 0$. During the execution of AKR the algorithm simultaneously computes a feasible forest for (IP) by augmenting the forest F and maintains a feasible dual solution to (D) by raising the value of certain dual variables.

We consider the execution of AKR as a process over time. At the initialization of the algorithm we set the time variable $\tau = 0$, and we increase τ and the dual variables at the same rate over the execution of the algorithm. Let x^τ represent the primal incidence vector, F^τ represent the forest induced by x^τ, and y^τ represent the dual solution vector at time $\tau \geq 0$. At the initiation of the algorithm, we have $x_e^0 = 0, \forall e \in E$ and $y_S^0 = 0, \forall S \subseteq V$.

For our example we use an instance of the Steiner Forest game. The different shapes represent three given terminal pairs. The corresponding cut requirement has $f(S) = 1$ if and only if S contains exactly one vertex of any shape (equivalent to S containing exactly one of the two vertices in any (s_i, t_i)-pair).

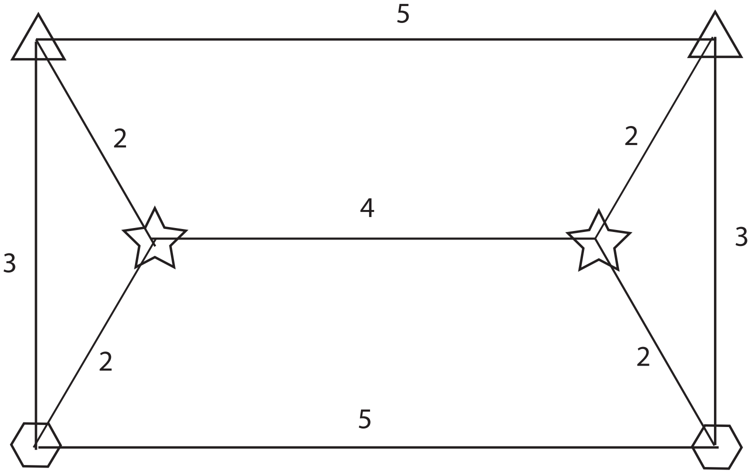

Figure 1.1: Sample Input Graph for AKR Algorithm.

We say a edge $e \in E$ becomes *tight* if the corresponding constraint (1) is satisfied with equality at time τ. We use $\bar{F^\tau}$ to represent the set of tight edges induced by the duals y^τ. For convenience, we refer to a connected component of $\bar{F}^\tau$ as a *moat*. We say a moat U of $\bar{F}^\tau$ is *active* if U separates at least one terminal pair, i.e., $U \in \mathcal{F}$ is a Steiner cut. Let $\mathcal{A}^\tau$ represent the set of active moats at time τ.

The AKR algorithm raises the dual variables of all active moats in $\mathcal{A}^\tau$ uniformly at all times $\tau \geq 0$. Whenever an edge becomes tight during the execution of an algorithm, we check for a *collision*. We say that moats U_1 and U_2 *collide* at time τ if

1. U_1 is an active moat at time τ ($U_1 \in \mathcal{A}^\tau$)
2. U_2 is an active moat at time τ
3. τ is the first time during the execution of the AKR algorithm at which forest $\bar{F}^\tau$ contains a connected component U such that $U_1 \subseteq U$ and $U_2 \subseteq U$.

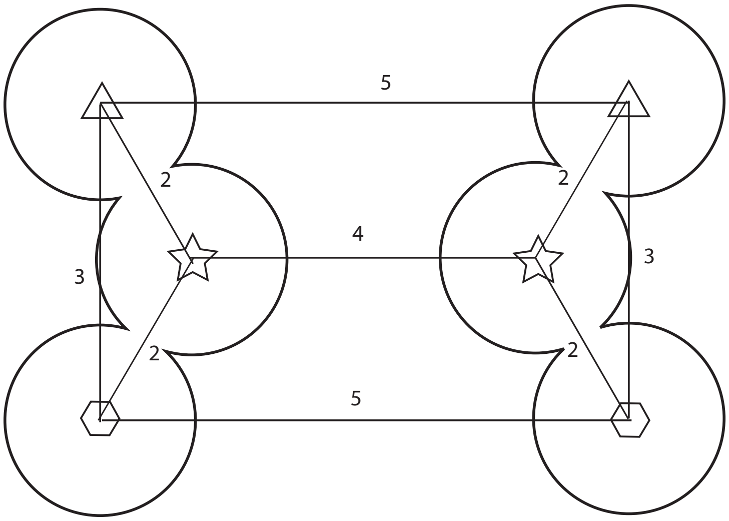

Figure 1.2: At this point, the edges with cost of 2 are tight and part of the current solution. The edges with cost 3 will soon be tight but will not be part of F, as they will not become tight as a result of two active moats colliding.

When a collision occurs at time τ between two active moats U_1 and U_2 , there must exist vertices $v_1 \in U_1$ and $v_2 \in U_2$ such that a path P_{v_1,v_2} between v_1 and v_2 becomes tight as a result of increasing y_{U_1} and y_{U_2}. When this happens, we add the path P_{v_1,v_2} to F^τ and continue the algorithm. The algorithm terminates at a time τ^* when the set of active moats $\mathcal{A}^{\tau^*}$ is empty. At this point the forest F^{τ^*} is feasible for the Steiner Forest problem.

We give a formal definition of the algorithm and present the main theorem of [1]:

Theorem 1.4.1. *Suppose that algorithm AKR outputs a forest* F *and a feasible dual solution* $\{y_U\}_{U\in\mathcal{F}}$*. Let* opt_R *represent the minimum cost of a Steiner forest for the terminal set* R*. Then*

$$c(F) \leq (2) \sum_{U\in\mathcal{F}} y_U \leq (2) opt_R$$

Algorithm 2 Primal-Dual AKR algorithm with uniform increase rule

```
$y \leftarrow 0$
$F \leftarrow \emptyset$
repeat
    $\mathcal{A} \leftarrow \{$ active moats $U\}$
    repeat
        Raise $y_U$ uniformly $\forall U \in \mathcal{A}$
    until Two moats $U_1$ and $U_2$ collide
    $P \leftarrow$ a path of tight edges joining some $v_1 \in U_1$ to some $v_2 \in U_2$
    $F \leftarrow F \cup P$
until F is feasible
Return: $(F, y)$
```

1.4.2 GW Algorithm

The GW algorithm presented in [3] extends the framework of the AKR algorithm presented in [1] to general downwards monotone and proper cut requirement functions. Throughout the remainder of this section we will refer to the algorithm as the GW algorithm, although we will be taking ideas from both algorithms as we develop our own in this thesis.

Recall the more general network design problem defined by (IP). The GW algorithm is also viewed as a process over time. The main difference between the two algorithms is in deciding which dual variables to raise during the execution. The GW algorithm is designed for more general cut requirement functions, so we need some new terminology and notation. For an infeasible forest F^τ, we say that the set S is *violated* if $\delta(S) \cap F^\tau < f(S)$. Unfortunately, there may be exponentially many violated sets so we are required to make some simplifying assumptions.

Recall we said we would restrict our focus to problems where the cut-requirement $f(S)$ is a $0,1$-function. Now we can say a set S is violated when $f(S) = 1$ and $\delta(S) \cap F^\tau = \emptyset$.

The GW algorithm only raises the dual variables of violated sets which do not contain violated subsets. It is easy to identify these minimal violated sets when the cut requirement function satisfies the maximality property, as the following lemma from [3] shows.

Lemma 1.4.2. *Let* $f(S)$ *be a* $\{0,1\}$ *function satisfying the maximality property, and let* $E \subseteq F$ *be any edgeset. Then a connected component* S *induced by the edgeset* E *with* $f(S) = 1$ *is a* minimal violated set.

Intuitively, a minimal violated set is a set S such that no subset $S' \subseteq S$ has $f(S') = 1$ and $\delta(S') \cap F^\tau = \emptyset$.

Let $\mathcal{C}^\tau$ represent the connected components of F^τ. Let $\mathcal{A}^\tau$ represent the set of minimal violated sets at time τ, i.e.

$$\mathcal{A}^\tau = \{S \in \mathcal{C}^\tau : f(S) = 1\}$$

The GW algorithm raises the dual variables of all minimal violated sets in $\mathcal{A}^\tau$ uniformly at all times $\tau \geq 0$. As the dual variables for minimal violated sets are raised, it could happen at time τ that the constraint of type 1 corresponding to an edge $e \in E$ becomes tight. At this point, the GW algorithm adds edge e to F^τ and continues.

At some time $\tau^* \geq 0$, the forest F^{τ^*} will be feasible for the given (IP). The GW algorithm now performs a reverse delete step by considering edges in the reverse order they were added to the forest. When considering the edge e, if the forest $F^{\tau^*} - \{e\}$ is feasible for (IP), then the GW algorithm removes e from F^{τ^*}. We let F represent the final forest after the reverse delete step.

Consider running the GW algorithm on the same example we presented for the AKR algorithm (see Figure 1.2).

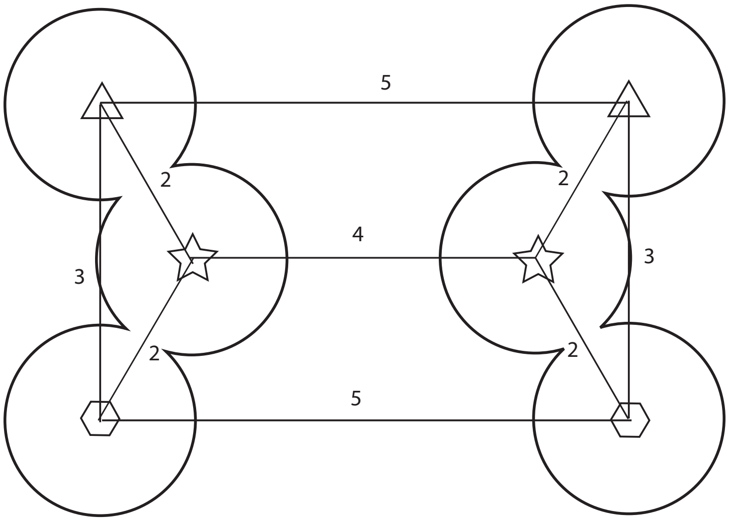

Figure 1.3: At this point, the edges with cost 2 are tight and are all part of the current solution F. Soon the edges of cost 3 will be tight and added to the solution.

It should be noted that in the original AKR algorithm we only add certain tight edges and do not perform a reverse-delete step. In the example, the edges of cost 3 will not be added in the AKR algorithm. In the case of the Steiner Forest problem, the execution of the two algorithms on the same graph and terminal set return the exact same solution. This is not the case for most other

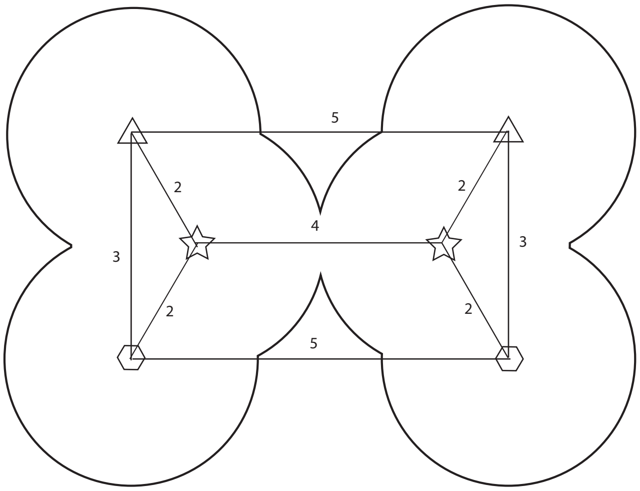

Figure 1.4: The edge with cost 4 is now tight, and in the reverse delete step, we remove both edges of cost 3

network design problems. We will use the more general GW framework for the first few algorithms presented in this thesis, and the AKR technique in the last algorithm presented.

The following theorem appears in [3] and its proof is included here for completeness:

Theorem 1.4.3. *Let F be the final forest produced by the GW Algorithm. Let $\{y_S\}_{S \subseteq V}$ be the corresponding feasible dual solution for (D). Let f be a 0,1-function satisfying the maximality property. Then F is feasible for (IP) and $c(F) \leq 2 \sum_{S \subseteq V} f(S) y_S$.*

Proof. We can show the result by considering the average degree of a forest. Consider an arbitrary time $\tau < \tau^*$ and the set of tight edges $E^\tau \subseteq F^\tau$ that will be retained in the final solution F (that is, $E^\tau \subseteq F$ as well). Define a *minimal augmentation* M of E^τ such that $E^\tau \subseteq M$, M is a feasible solution and $M - e$ is infeasible for all edges $e \in M$. By definition, the final forest F returned by GW is a minimal augmentation of E^τ.

Now consider the set $\mathcal{A}^\tau$ of minimally violated sets induced by the edgeset E^τ in G at time τ. If we increase the dual variables of all sets in $\mathcal{A}^\tau$ at time τ by ϵ, then the total incremental dual growth is $\epsilon|\mathcal{A}^\tau|$. At all times during the execution of GW, these minimally violated sets are incident to edges in $F \backslash E^\tau$, that is, edges that are not tight yet but will be added to the final solution at some later time. For a set $A \in \mathcal{A}^\tau$, let

$$\delta_{E^\tau}(A) = \delta(A) \cap F \backslash E^\tau$$

We need to show that the number of such edges is at most twice $|\mathcal{A}^\tau|$ at all times $\tau < \tau^*$. This implies we can share the dual growth of the minimally violated

Algorithm 3 Primal-Dual GW algorithm with uniform increase rule and reverse delete step

```
$y \leftarrow 0$
$F \leftarrow \emptyset$
$i \leftarrow 0$
repeat
    $i \leftarrow i + 1$
    $\mathcal{A} \leftarrow \{$ minimal violated sets $S\}$
    repeat
        Raise $y_S$ uniformly $\forall S \in \mathcal{A}$
    until An edge $e_i$ becomes tight
    $F \leftarrow F \cup \{e_i\}$
until F is feasible
for $j = i$ to 1 do
    if $F - \{e_i\}$ is feasible then
        $F \leftarrow F - \{e_i\}$
    end if
end for
Return: $(F, y)$
```

sets at any time τ among the final edges that 'feel' the dual growth, and the theorem will be shown.

Consider the auxiliary graph H^τ formed by taking G with edge set E^τ and shrinking the connected components of E^τ down to vertices. Every edge in H^τ now corresponds to an edge that will become tight at some future time $\tau' > \tau$ (and will not be removed in the reverse delete step). This is exactly the set of edges $F \backslash E^\tau$, so

$$\sum_{A \in \mathcal{A}^\tau} |\delta_{E^\tau}(A)| = \sum_{v \in H^\tau} d_v$$

Each vertex of H^τ corresponds to a connected component induced by E^τ. Let W^τ be the set of vertices of H^τ corresponding to sets in $\mathcal{A}^\tau$. Naturally,

$$|\mathcal{A}^\tau| = |W^\tau|$$

The cut requirement function f on G translates nicely to the graph H^τ by letting $f(v) = 1$ for $v \in V(H^\tau)$ if and only if v corresponds to a connected component S induced by E^τ and $f(S) = 1$.

Notice that H^τ has the property that no connected component contains more than one vertex x with $f(x) = 0$. Suppose, by contradiction, that $\exists x, y \in C \subseteq H^\tau$ with $f(x) = f(y) = 0$, C a connected component of H^τ. Then $\exists$ a path from x to y in H^τ and an edge e whose deletion disconnects x and y. Consider the resulting sets $S_1 \cup S_2 = C$ and suppose without loss of generality that $x \in S_1$ and $y \in S_2$. Then $f(S_1) = f(S_2) = 0$, and $E^\tau - e$ is still feasible, contradiction to the minimality of H^τ. Now suppose H^τ has c components. Thus

$$|V(H^\tau)| - c \leq |W^\tau|$$

Thus, we now have

$$\sum_{A \in \mathcal{A}^\tau} |\delta_{E^\tau}(A)| = \sum_{v \in H^\tau} d_v \leq 2(|V(H^\tau)| - c) \leq 2|W^\tau| = 2|\mathcal{A}^\tau|$$

since H^τ is a forest implies $\sum_{v \in H^\tau} d_v \leq 2(|V(H^\tau)| - c)$ and the result is shown. □

This implies that the GW algorithm is a 2-approximation for network design problems given by (IP) where the cut requirement function f satisfies the maximality property. In the remainder of this thesis, we apply this framework to some network design problems with special properties.

In the next chapter we will consider four different network design games. We will use variations of the primal-dual algorithms from this chapter to solve special instances of network design problems underlying the given network design game. We will then use the dual solutions to derive cross-monotonic cost-shares for the agents playing the game. We will also briefly discuss how to use these cost-shares in a Moulin Mechanism to ultimately compute which players to service and how much to charge them for the service. The final chapter contains discussion on directions for future work arising from this thesis.

Chapter 2

Algorithms for Network Design Games

Recall we introduced general network design games with a player set P, where $|P| = p \geq 1$. Each player $i \in P$ has their own cut requirement function f_i, and seeks to establish a forest $F \subseteq E$ that satisfies their cut requirement. As an example, consider the *Steiner Forest Game*, based on an instance of the Steiner Forest problem. Each terminal pair (s_i, t_i), $1 \leq i \leq k$ in the Steiner Forest problem is associated with a player $i \in P$. Each player i wishes to connect their terminal pair, so that a feasible solution to the Steiner Forest Game is a forest $F \subseteq E$ such that both vertices of each player's pair are contained in the same tree of F. We assume that each player $i \in P$ has a private utility u_i for establishing this connection.

To solve a network design game like the Steiner Forest Game we can use a cost-sharing mechanism. First we solicit bids from the players, run the cost-sharing methods we will develop in this chapter on the player set P and assign a cost share $\xi_P(i)$ for each player $i \in P$. Next we can use the Moulin mechanism to remove any players R whose assigned cost-shares are greater than their bids. We then rerun the cost-sharing method as a subroutine of the Moulin mechanism on the smaller player set $Q = P \backslash R$. The Moulin mechanism requires us to repeat this process until all the remaining players' cost-shares are at most their bids. In this manner every time we run the algorithm that computes the solution to service the current set Q we do so ignorant of the bids and the other iterations of the algorithm. Thus, we can effectively ignore the bids of the players while designing the approximation algorithms and cost-sharing methods for these network design games.

The resulting solution may not satisfy every player's cut requirement function. Instead, players whose utility is less than their assigned cost-shares are completely removed from the game. As explained above, on subsequent iterations we run the cost-sharing method on the player set Q. A feasible solution to a network design game is one that satisfies the cut requirements of every player

$i \in Q$.

2.1 Death-time concept

The notion of vertex death-time in the execution of a primal-dual algorithm was introduced in [7]. Könemann, Leonardi and Schäfer modified the AKR algorithm to develop a 2-budget balanced and cross-monotonic cost-sharing method (known as KLS) for the Steiner Forest Game. Recall we explored how the GW algorithm generalized the AKR algorithm to downwards monotone and proper cut requirement functions. This thesis takes the natural step of generalizing the KLS cost-sharing method to downwards monotone and proper cut requirement functions.

To see why we are interested in designing non-trivial cost-sharing methods, consider trying to obtain a cross-monotonic cost-sharing method directly from AKR. Suppose we run the AKR algorithm on our sample Steiner Forest problem and divide the dual growth of y_U among all players whose vertex pair is separated by U. We provide a specific example that shows the resulting duals do not give rise to a cross-monotonic cost-sharing method.

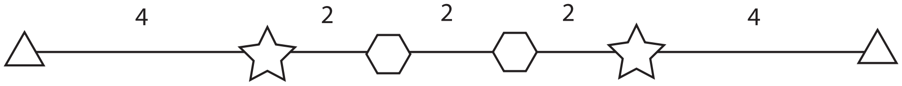

Figure 2.1: Our sample Steiner Forest game

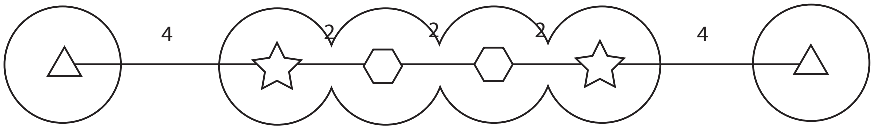

Figure 2.2: AKR at time 1

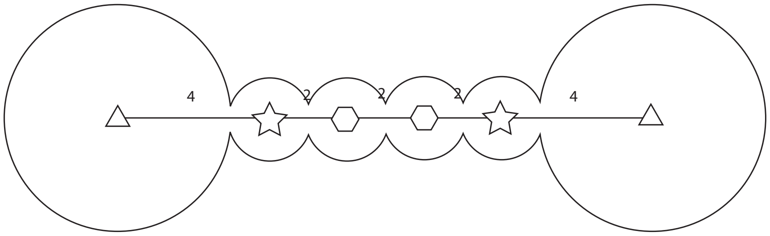

Figure 2.3: Uniform distribution of dual growth among active vertices would result in a total cost share to the triangle player of 3 + 3 = 6

This is because the amount of dual growth each set experiences may be dependent on the presence of other players.

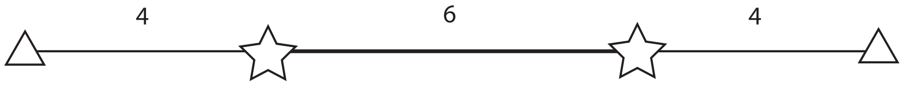

Figure 2.4: We remove the hexagon player from the game

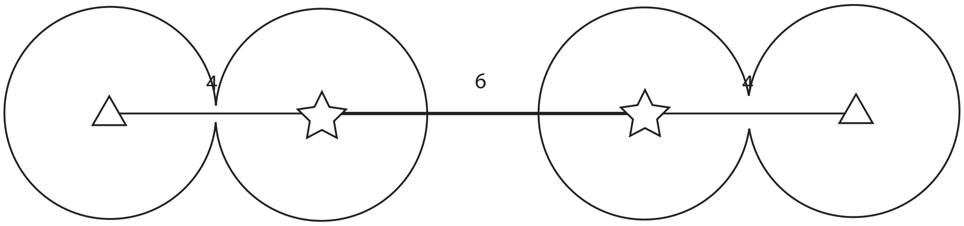

Figure 2.5: AKR running at time 2

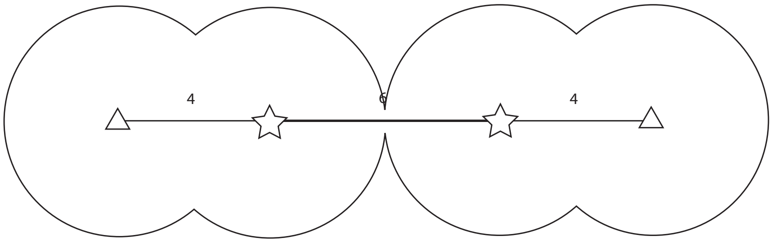

Figure 2.6: Uniform distribution of dual growth would now result in a total cost share of 2 + 2 + 1/2 + 1/2 = 5 for the triangle player

Comparing the example in Figure 2.3 with the example illustrated in Figure 2.6, the removal of the hexagon player results in a lower total cost share for the triangle player. Equivalently, observe that if we add the hexagon player back, the triangle player's cost share goes up. This is a violation of cross-monotonicity.

The concept of vertex death-time allows us to control the dual growth so that the resulting duals naturally give rise to a cross-monotonic cost-sharing method. Könemann, Leonardi and Schäfer used death-time to develop the KLS cost-sharing method for the Steiner Forest game in [6]. The death-time of each vertex of an s_i, t_i pair was defined as half the cost of the shortest s_i, t_i-path.

This definition arises naturally from a more general way to define vertex death-time. Suppose the current player set is $Q \subseteq P$. We run the GW algorithm presented earlier on the graph G. However, instead of growing the duals for all violated sets, we only grow the dual of sets that violate the cut requirement f_i of a specific player $i \in Q$. For example, to calculate the vertex death-time for player 1 in the Steiner Forest game, we would only grow the duals of the connected components that contained exactly one of s_1 or t_1. As a result, at all times during the execution of the algorithm, exactly two moats are growing and will continue growing until they collide. At this point, the shortest s_i, t_i-path is tight. Thus the sum of the duals of sets containing s_i returned by the i^{th} run of the GW algorithm is exactly half the cost of the shortest s_i, t_i-path. The same is true for the sum of the duals of sets containing t_i. After repeating this process for each player $i \in Q$, we will have computed the vertex death-time for all s_i, t_i pairs. The resulting death-times are exactly those used in [7].

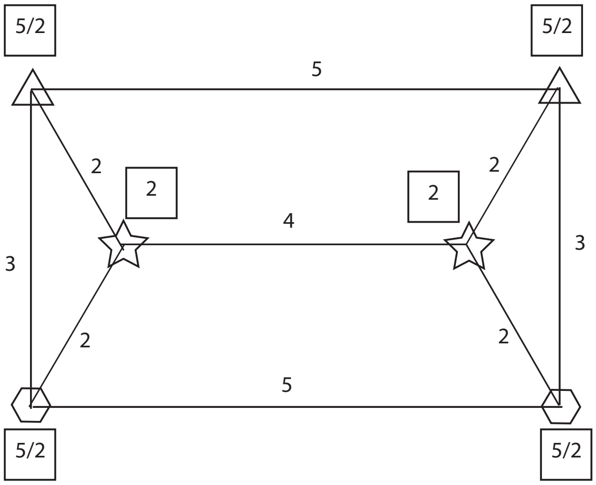

Figure 2.7: The death times for each vertex have been added to this example

Of course, we do not need to calculate the death-times this way since we can efficiently compute shortest paths, but this does help to motivate our choice of death-time for other games. If we extend this idea to a general network design game, the death-time of a vertex v for player i reflects the last time v is contained in a set that violates the cut-requirement f_i, if we were to run the GW algorithm in the absence of the other players.

We cannot give a general definition of death-time that is suitable for all network design games. Instead, in each game we consider we will give an explicit way of calculating the vertex death-time $d_i(v)$ of a vertex v for a player i. We can now define the notion of activity for this thesis and say a vertex v is *active* at time τ if $\tau \leq d_i(v)$ for some player $i \in Q$. Note that $d_i(v)$ could be positive for multiple players. In the Steiner Forest Game for example, it could be that v is a vertex of two or more terminal pairs. It will be convenient to define the death-time $D(v)$ of vertex v, without reference to any player, as

$$D(v) = \max_{i \in Q} d_i(v)$$

To summarize, we say a vertex v is *active* at time τ if $\tau \leq D(v)$.

Let $\mathcal{U}^\tau$ represent the connected components of $\bar{F^\tau}$, that is, the set of moats. If $f(S) = 1$ for a moat $S \in \mathcal{U}^\tau$, we say the moat S is unsatisfied. For consistency with the notation introduced with the GW algorithm, we let $\mathcal{A}^\tau$ represent the set of unsatisfied moats.

Define an *active moat* as a moat that contains an active vertex. Now we can define the set of active moats as:

$$\mathcal{M}^\tau = \{S \in \mathcal{U}^\tau : \exists v \in S \text{ such that } \tau < D(v)\}$$

The algorithms presented in the following sections will all follow a primal-dual framework based on the GW & AKR algorithms. However, instead of raising the duals of minimal violated sets, the algorithms will raise the dual variables of active moats to develop an intermediate forest F^{τ^*}, where τ^* is the time the last active terminal(s) become(s) inactive. Then the algorithms may add paths to augment the intermediate forest F^{τ^*} into a feasible forest F. The details of each algorithm will be presented in the section dedicated to each individual network design game.

In each section we will introduce the network design game, review the primal-dual method we will use to solve the underlying network design problem, and compute the death-times for each vertex. We will then introduce our cost-sharing method for each game, and show the resulting cost-shares recover a constant fraction of the cost of the solution returned by the algorithm. We will also show, where possible, that the cost-sharing method is cross-monotonic, and thus gives rise to a group-strategyproof cost-sharing mechanism.

2.2 Single-node Multi-player Edge-Cover Game

In this section we consider a game-theoretic version of the edge cover game. We are given an undirected graph G with node set V and edge set E, where each edge $e \in E$ has a non-negative cost c_e. We are also given a player set P, with $|P| = p \geq 1$ players. For now, we assume that each player owns exactly one node and that no node is owned by more than one player, so we can identify each player $i \in P$ with their node $v_i \in V$. Let $O \subseteq V$ be the set of nodes owned by the players. Note that if there are more nodes than players, some nodes will remain unowned.

Each player $i \in P$ wants their node $v_i \in O \subseteq V$ to be covered by an edge added in the solution. In the context of the mechanism, each player $i \in P$ has a private utility u_i for satisfying their vertex, and will submit a bid b_i to the service provider. All other information, including who owns which nodes, is public. On a given iteration of this algorithm with current player set $Q \subseteq P$ and associated vertices $O_Q \subseteq O$, a feasible solution to the single-node multi-player edge-cover game is a set of edges $F \subseteq E$ such that $\delta(v) \cap F \neq \emptyset, \forall v \in O_Q$, i.e. F must contain at least one edge incident to each node v that is owned by one of the players. The goal is to find a minimum-cost feasible solution to this problem.

In the remainder of this section, we present a cross-monotonic cost-sharing method that is 2-budget balanced for the above network design game. This result is the best possible, due to a result in [4] where the authors proved that there does not exist a $(2-\epsilon)$-budget balanced cross-monotonic cost-sharing method for the edge cover problem, for any $\epsilon > 0$. Our result confirms their bound is tight for the general edge cover game.

2.2.1 Primal-dual method

To solve this problem we use a primal-dual algorithm based on the framework provided by Goemans & Williamson [3]. Consider the following integer programming formulation for the edge-cover game:

$$
\begin{aligned}
\min \quad & \sum_{e\in E} c_e x_e & & \quad\text{(IP-1)}\\
\text{s.t.} \quad & \sum_{e\in\delta(v)} x_e \geq 1 & \forall v \in O, &\\
& x_e \in \{0,1\} & \forall e \in E. &
\end{aligned}
$$

This linear program fits nicely into the framework we set up for general network design problems by defining $f(\{v\}) = 1$ if and only if $v \in O$. Again, it will be convenient to consider the linear programming relaxation (LP-1) of the above (IP-1), where the constraints $x_e \in \{0,1\}, \forall e \in E$ are replaced with the constraints $0 \leq x_e, \forall e \in E$. Note that in an optimal solution to the linear relaxation no edge variable x_e would be assigned a value of more than 1. We also present the linear programming dual (D-1) of (LP-1), which has a dual variable y_v for each node $v \in O$ and a constraint for each edge $e \in E$:

$$
\begin{aligned}
\max \quad & \sum_{v\in O} y_v & & \quad\text{(D-1)}\\
\text{s.t.} \quad & \sum_{v\in O: e\in\delta(v)} y_v \leq c_e & \forall e \in E, & \quad(1)\\
& y_v \geq 0 & \forall v \in O. &
\end{aligned}
$$

Note that in the edge-cover game, the only candidates for violated sets are the nodes $v \in O$. Also note that on subsequent iterations with player set Q and active nodes O_Q we can rebuild these LPs by substituting O_Q for O.

We present the Könemann-Wheatley (KW) Algorithm for the Single-node Multi-player Edge-Cover Game (KW-1). The cut requirement f_i for player i is very simple:

$$
f_i(S) = \begin{cases} 1 & \text{if } S = \{v_i\} \\ 0 & \text{otherwise} \end{cases}
$$

The general cut requirement for the current iteration, captured in the problem (IP-1), is

$$f(S) = \max_{i \in Q} f_i(S)$$

Note that since each player i controls only one node, and each node $v \in O_Q$ is owned by exactly one player, for vertex $v_i \in O_Q \subseteq V$ we can actually compute the death-time:

$$D(v_i) = d_i(v_i) = \frac{1}{2} \text{ cost of the shortest edge incident to } v_i$$

For all other vertices $u \notin O$, we automatically set $D(u) = 0$. Once the death-times are computed we consider a fresh instance of the graph G with $y_v = 0, \forall v \in V$ and $F = \emptyset$. The algorithm first uniformly raises the dual variables of every active vertex until some edge becomes tight. As in the GW algorithm, the KW-1 algorithm adds this edge to the forest F^τ, and then continues to uniformly raise the dual variables of connected components of F^τ that contain active vertices. In this version of the edge-cover game, a connected component of F^τ containing more than one vertex cannot contain any active vertices (Why? Suppose u, v are two vertices in the same connected component, and u is active. Then $\tau < D(u)$ implies that the dual variables $y_u = y_v$ are each less than half the length of the shortest edge incident to u. Thus, no edge incident to u is tight). This implies the only candidates for active moats are active vertices. The KW-1 algorithm continues in this fashion until a time τ^* when there are no more active vertices. It is important to note that the dual variable y_v is raised uniformly until time $\tau = D(v)$, so at time τ^*, we have $y_v = D(v)$ for all $v \in V$. Let $F^0 = F^{\tau^*}$.

Here we can justify with a quick example why we must use half the cost of the shortest edge so that the cost-sharing method remains competitive. Consider a graph with two vertices, each owned by one of two players. The vertices are joined by an edge of unit length. Suppose instead we set the death time to be the length of the shortest edge incident to v_i. The death-times of both vertices would be 1, and then the total dual growth using the above method is $\frac{3}{2}$. This does not immediately lead to a competitive cost-sharing method. By defining the death-times as we do here, we can ensure the resulting cost-sharing method is competitive, but we may have to add some edges.

Next the KW-1 algorithm adds edges to connect any unsatisfied vertices. We refer to this as the edge-buying phase. We proceed with a simple greedy algorithm which considers the shortest edge from unsatisfied vertices in G to their nearest neighbour. For this game, the shortest edge for an unsatisfied vertex v is just the shortest edge incident to v. In practice, we could create a table with one entry for each unsatisfied vertex and the shortest edge we can buy to satisfy it. Then we can buy each edge in order from shortest to longest, updating the table as we go. We can resolve ties by using the player ordering implied by the player labels $1, ..., n$, buying the edge of the vertex owned by the lower-numbered player first. We add the 1^{st} edge to F^0, call the new forest F^1 and repeat for each unsatisfied vertex. For notation purposes, we say we add the k^{th} edge to F^{k-1} and call the new forest F^k. After repeating this simple edge-buying process for each unsatisfied vertex, adding some p edges, F^p will be a feasible solution to the problem.

Finally, we perform the reverse delete step that is standard in primal-dual algorithms. Suppose $F^p = \{e_1, ..., e_l\}$ and that edge e_i was added before e_j if and only if $i < j$. We consider every edge $e \in F^p$ in the reverse order they were added to F^p (throughout the entire algorithm) and delete an edge e from F^p if $F^p - e$ is feasible for the problem. Some tight edges may be removed that were added during the dual-growth step. Let the final forest be F. Just as F^p was feasible for the problem, so is F.

Algorithm 4 Primal-Dual KW-1 algorithm for edge cover game

```
Given: $D(v)$ for each $v \in O_Q$
$y \leftarrow 0$
$F \leftarrow \emptyset$
$i \leftarrow 0$
repeat
    $i \leftarrow i + 1$
    $A \leftarrow \{v \in O_Q : y_v \leq D(v)\}$
    repeat
        Raise $y_v$ uniformly $\forall v \in A$
    until An edge $e_i$ becomes tight or all vertices are inactive
    $F \leftarrow F \cup \{e_i\}$
until $A = \emptyset$
repeat
    $i \leftarrow i + 1$
    $e_i \leftarrow$ the edge connecting $v$ to its nearest neighbour
    $F \leftarrow F \cup \{e_i\}$
until $\nexists v \in V$ such that $F \cap \delta(v) = \emptyset$
for $j = i$ to 1 do
    if $F - \{e_i\}$ is feasible then
        $F \leftarrow F - \{e_i\}$
    end if
end for
Return: $(F, y)$
```

2.2.2 Analysis: Cost Recovery

Consider the set of edges $F^0 \cap F$, i.e., the set of tight edges after the dual-growth phase that were not removed in the reverse delete phase. Let $X \subseteq O_Q$ be the set of vertices that are owned by one of the players and are incident to one (or more) of these tight edges. Formally, we can define:

$$X = \{v \in O_Q : \delta(v) \cap \{F^0 \cap F\} \neq \emptyset\}$$

Note that a vertex $v \in X$ is simply a vertex with $f(v) = 1$ that is now satisfied by an edge added in the dual growth phase. We can use the following lemma to bound the cost of the edges added in the dual-growth phase:

Lemma 2.2.1. *The total cost of the edges of F added during the dual growth stage, denoted $c(F^0 \cap F)$, satisfies $c(F^0 \cap F) \leq 2\sum_{v\in X} y_v$*

Proof. The only edges added during the dual growth stage are those edges that become tight. Since a moat for vertex v only grows until the dual y_v attains half the value of the shortest edge incident to v, an edge $e = uv$ becomes tight if and only if u is v's nearest neighbour and likewise v is u's nearest neighbour.

Since the reverse delete phase will remove any unnecessary edges, the set of edges $F^0 \cap F$ we are considering clearly form a forest. The vertices incident to the edges $F^0 \cap F$ are exactly the vertices in X. Since $F^0 \cap F$ is a forest, $|F^0 \cap F| \leq |X| - 1$. The average degree of a forest is at most two, so each vertex with a positive dual value is incident to an average of at most 2 edges. Thus

$$c(F^0 \cap F) \leq 2\sum_{v\in X} y_v$$

as each edge $e = uv \in \{F^0 \cap F\}$ has cost $c(e) = 2y_v = 2y_u$ and the result is shown. □

Let $Z \subseteq O_Q$ be the remaining set of unsatisfied vertices, such that $X \cup Z = O_Q$ and $X \cap Z = \emptyset$. We will satisfy these vertices in the edge-buying phase, and thus we need the following simple lemma to bound the cost of the edges bought in the edge-buying phase:

Lemma 2.2.2. *The total cost of the edges of F added during the edge-buying stage, denoted $c(F^p - F^0)$, satisfies $c(F^p - F^0) \leq 2\sum_{v\in Z} y_v$*

Proof. The cost of the edge bought to satisfy a vertex $v \in Z$ with $f(v) = 1$ is at most the cost of the shortest edge incident to v. Since $D(v)$ is half this cost, the shortest satisfying edge $P(v)$ satisfies

$$P(v) \leq 2D(v)$$

Also recall that $y_v = D(v)$, so we have

$$P(v) \leq 2y_v$$

Which gives

$$\sum_{v\in Z} c(P(v)) = c(F^p - F^0) \leq 2\sum_{v\in Z} y_v$$

by summing over all vertices in Z, and the result is shown. □

These two lemmas give rise to the following theorem bounding the cost of the forest outputted by KW-1:

Theorem 2.2.3. *The forest F and duals $y_v, \forall v \in V$ returned by the KW-1 algorithm satisfy $c(F) \leq 2\sum_{v \in V} y_v$*

Proof. Recall that $X \cup Z = O_Q$, so each vertex $v \in O_Q$ is now satisfied and F is a feasible solution to the original problem. Note that the forest $F = \{F^0 \cap F\} \cup \{F^p - F^0\}$, since the right-hand side is just the set of edges added in the first two phases and not subsequently deleted in the reverse-delete phase. Thus, we have:

$$c(F) \leq c(F^0 \cap F) + c(F^p - F^0) \leq 2\sum_{v \in X} y_v + 2\sum_{v \in Z} y_v \leq 2\sum_{v \in V} y_v$$

since $X \cap Z = \emptyset$ and the result is shown. □

2.2.3 Analysis: Cross-monotonic Cost-Sharing Method

For each vertex $v \in O_Q$, we define the cost-share for vertex v as $\xi_Q(v) = D(v)$. Since each player only owns one vertex, define the cost-share of the player $i \in Q$ who owns v as $\xi_Q(i) = \xi_Q(v)$. We will show this cost-sharing method ξ is cross-monotonic and 2-budget balanced for the single-node edge cover game.

It is important to note that we could have just defined a very simple cost-sharing method for the edge cover game that does not require a primal-dual algorithm. That is, we can compute $D(v)$ just by finding the shortest edge incident to v, add this edge to the solution F, and then define the cost-share as above. We presented the primal-dual algorithm with path-buying phase as an introductory exercise to illustrate how the our cost-sharing method will work on more complex problems. Immorlica, Mahdian & Mirrokni give a simple 2-budget balanced cross-monotonic cost-sharing method for the unweighted edge cover game in [4], and as mentioned before, show the result is best possible.

We continue by proving our cost-sharing method is indeed cross-monotonic. Consider an arbitrary player $i \in Q$, their node $i \in O_Q$ and let $Q_0 = Q \backslash \{i\}$.

Lemma 2.2.4. *The cost-sharing method ξ arising from KW-1 is cross-monotonic, i.e., for each $v \in Q_0$ we have $\xi_{Q_0}(v) \geq \xi_Q(v)$*

Proof. Recall that $\xi_Q(v) = D(v)$ for a vertex $v \in O_Q$, and $D(v)$ is just 1/2 the length of the shortest edge incident to v. Specifically, $D(v)$ is calculated in absence of any of the other players, including the player i we arbitrarily removed from Q. We then have $\xi_{Q_0}(v) = D(v)$ and in fact $\xi_{Q_0}(v) = \xi_Q(v)$ for each $v \in Q_0$. Thus ξ is cross-monotonic. □

Lemma 2.2.5. *The cost-sharing method ξ arising from KW-1 is competitive, i.e., $\sum_{i \in Q} \xi_Q(i) \leq opt_Q$*

Proof. Recall $\xi_Q(i) = \xi_Q(v) = y_v$ for the player i who owns vertex v. For any edge $e = uv \in E$ we have that $y_u \leq c(e)$ (and $y_v \leq c(e)$ respectively) since y_u (resp. y_v) is at most half the length of the shortest edge incident to vertex u (resp. v). Thus equation (1) in (D-1) reduces to $y_u + y_v \leq c_e$ for each $e \in E$, implying the duals $\{y_v\}_{v \in O_Q}$ outputted by KW-1 are feasible to (D-1). Thus

$$\sum_{i \in Q} \xi_Q(i) = \sum_{v \in O_Q} y_v \leq opt_Q$$

where the last inequality follows from Weak Duality theory. □

Theorem 2.2.6. *Suppose that algorithm KW-1 outputs a forest F and a dual solution $\{y_v\}_{v \in O_Q}$. Then ξ is a 2-budget balanced cost-sharing method, i.e.*

$$\frac{1}{2}c(F) \leq \sum_{i \in Q} \xi_Q(i) \leq opt_Q$$

Proof. Recall result 2.2.3 which bound the cost of the forest F by $c(F) \leq 2\sum_{v \in V} y_v$. Also note that when running KW-1 with player set Q and active vertices O_Q we only increase the dual variables of the vertices in O_Q, so

$$c(F) \leq 2\sum_{v \in V} y_v = 2\sum_{v \in O_Q} y_v$$

By design $y_v = D(v) = \xi_Q(v) = \xi_Q(i)$ for each vertex $v \in O_Q$ (owned by player $i \in Q$) so

$$\sum_{v \in O_Q} y_v = \sum_{i \in Q} \xi_Q(i) \leq opt_Q$$

where the last inequality follows from the previous lemma on competitiveness. □

2.3 Multi-player Multi-node Edge Cover Game

In this section we consider a subtly different game-theoretic version of the edge cover game. We are given an undirected graph G with node set V and edge set E, where each edge $e \in E$ has a non-negative cost c_e. We are also given a player set P, with $|P| = p \geq 1$ players. Now we allow each player to own multiple nodes, and remove the restriction that each node is owned by at most one player. Let O_i represent the set of nodes owned by player $i, \forall i \in P$, and let $O = \cup_{i \in P} O_i$ represent the set of nodes owned by at least one player.

Each player $i \in P$ wants to satisfy all their nodes $v \in O_i$ by buying a set of edges such that each of the nodes $v \in O_i$ is incident to at least one of the edges in the final solution. In the context of the mechanism, each player has a private utility u_i for satisfying all their nodes and will submit a bid b_i to the service provider. Again, we focus on one iteration of the algorithm, with current player set $Q \subseteq P$ and active nodes $O_Q = \cup_{i \in Q} O_i$. A feasible solution to the multi-node multi-player edge-cover game is a set of edges $F \subseteq E$ such that $\delta(v) \cap F \neq \emptyset, \forall v \in O_Q$, i.e. F must contain at least one edge incident to each node v that is owned by one of the players. The goal is to find a minimum-cost feasible solution to this problem.

In the remainder of this section, we will see that the KW-1 algorithm also gives rise to a cross-monotonic cost-sharing method that is 2-budget balanced for the above network design game.

2.3.1 Primal-Dual Method

The IP presented for the single-node multi-player edge-cover game (IP-1) is the same IP for this game, as the formulation of the IP does not depend on who owns the nodes. Similarly, the dual problem (D-1) is the same for the multi-node multi-player edge-cover game. Again, on subsequent iterations with player set Q and active nodes O_Q we can rebuild these IPs by substituting O_Q for O. Also note that in the edge-cover game, the only candidates for violated sets are the nodes $v \in O_Q$.

The cut requirement f_i for player i is again very simple:

$$f_i(S) = \begin{cases} 1 & \text{if } S = v \text{ such that } v \in O_i \\ 0 & \text{otherwise} \end{cases}$$

The general cut requirement for the current iteration, captured in the problem (IP-1), is

$$f(S) = \max_{i \in Q} f_i(S)$$

As for the first version of edge-cover we considered, the death-time of vertex $v \in O_i$ for player $i \in Q$ is

$$d_i(v) = \frac{1}{2} \text{ cost of the shortest edge incident to } v$$

Since each node may be owned by more than one player, we can compute the Death-time of vertex v, without reference to a player, as

$$D(v) = \max_{i \in Q} d_i(v)$$

which the reader may recall defines the time τ when vertex v becomes inactive.

From this point on the KW-1 algorithm runs just as was described for single-node edge cover. Once the death-times are computed we consider a fresh instance of the graph G with $y_v = 0, \forall v \in V$ and $F = \emptyset$. Please see section 2.2.1 for the details of the execution of KW-1.

2.3.2 Analysis: Cost Recovery

Again we consider the set of edges $F^0 \cap F$, and let $X \subseteq O_Q$ be the set of vertices that are owned by one of the players and are incident to one (or more) of these tight edges. Recall we defined this formally as:

$$X = \{v \in O_Q : \delta(v) \cap \{F^0 \cap F\} \neq \emptyset\}$$

We will bound the cost of the edges $c(F^0 \cap F)$ by using result 2.2.1 from the previous section.

Lemma 2.3.1. *The total cost of the edges of F added during the dual growth stage, denoted $c(F^0 \cap F)$, satisfies $c(F^0 \cap F) \leq 2\sum_{v \in X} y_v$*

Proof. Note that for a vertex v owned by players $\{1, ..., j\} \in Q$ we have that $d_1(v) = ... = d_j(v) = D(v)$ since the death-time $d_i(v)$ is just half the cost of the shortest edge incident to v. In this sense, we can instead assume that each vertex is owned by only one player. Consider then the execution of the KW-1 algorithm with active vertex set X, each one owned by one of $|X|$ distinct players. This will cause the same set of edges $c(F^0 \cap F)$ to become tight during the dual growth phase. We can apply lemma 2.2.1 and the result follows. □

Recall we let $Z \subseteq O_Q$ be the remaining set of unsatisfied vertices, such that $X \cup Z = O_Q$ and $X \cap Z = \emptyset$. We will satisfy these vertices in the path-buying phase, and again we can bound the cost of the bought edges with:

Lemma 2.3.2. *The total cost of the edges of F added during the path-buying stage, denoted $c(F^p - F^0)$, satisfies $c(F^p - F^0) \leq 2\sum_{v \in Z} y_v$*

Proof. As in the proof for the previous lemma, we can assume each vertex in Z is owned by only one player and apply lemma 2.2.2. The result follows immediately. □

2.3.3 Analysis: Cross-monotonic Cost-Sharing Method

For each vertex $v \in O_Q$, we define the cost-share for vertex v as $\xi_Q(v) = D(v)$. For a vertex $v \in O_Q$ owned by k players $\{1, ..., k\} \in Q$ define the cost-share of player i for vertex v as $\xi_Q^i(v) = \frac{1}{k}\xi_Q(v)$. Then the total cost-share for player $i \in Q$ can be defined as

$$\xi_Q^i = \sum_{v \in O_i} \xi_Q^i(v)$$

We will show this cost-sharing method ξ is cross-monotonic and 2-budget balanced for the multi-node edge cover game. We note again that we can avoid the primal-dual phase when developing our cost-sharing method for the edge cover game.

Consider an arbitrary player $i \in Q$, let $Q_0 = Q \backslash \{i\}$ and consider the set of vertices O_{Q_0} owned by the players in Q_0.

Lemma 2.3.3. *The cost-sharing method ξ is cross-monotonic, i.e., for each player $j \in Q_0$ we have $\xi_{Q_0}^j \geq \xi_Q^j$*

Proof. Recall that $\xi_Q(v) = D(v)$ for a vertex $v \in O_Q$, and $D(v)$ is just 1/2 the length of the shortest edge incident to v. Specifically, $D(v)$ is calculated in absence of any of the other players, including the player i we arbitrarily removed from Q. We then have $\xi_{Q_0}(v) = D(v)$ if $v \in O_{Q_O}$.

Suppose that $v \in O_i$ and v is owned by $k > 0$ other players in Q_0. Then while $\xi^i_Q(v) = \frac{1}{k+1}\xi_Q(v)$ we now have for each player j such that $v \in O_j$:

$$\xi^j_{Q_0}(v) = \frac{1}{k}\xi_{Q_0}(v) > \xi^i_Q(v)$$

Now we get $\xi^j_{Q_0} \geq \xi^j_Q$ by summing over all vertices owned by player $j \in Q_0$. Thus ξ is cross-monotonic. □

Lemma 2.3.4. *The cost-sharing method ξ is competitive, i.e.,* $\sum_{i\in Q} \xi_Q(i) \leq$ opt_Q

Proof. Recall $\xi_Q(v) = y_v$ for every $v \in O_Q$. We split up the cost-share for vertex v among all the players that own it, so we maintain $\sum_{i\in Q} \xi_Q(i) = \sum_{v\in V} \xi_Q(v)$.

For any edge $e = uv \in E$ we have that $y_u \leq c(e)$ (and $y_v \leq c(e)$ respectively) since y_u (resp. y_v) is at most half the length of the shortest edge incident to vertex u (resp. v). Thus equation (1) in (D-1) reduces to $y_u + y_v \leq c_e$ for each $e \in E$, implying the duals $\{y_v\}_{v\in O_Q}$ outputted by KW-1 are feasible to (D-1). Thus

$$\sum_{i\in Q} \xi_Q(i) = \sum_{v\in V} y_v \leq opt_Q$$

where the last inequality follows from Weak Duality theory. □

Theorem 2.3.5. *Suppose that algorithm KW-1 outputs a forest F and a dual solution $\{y_v\}_{v\in O_Q}$. Then ξ is a 2-budget balanced cost-sharing method, i.e.*

$$\frac{1}{2}c(F) \leq \sum_{i\in Q} \xi_Q(i) \leq opt_Q$$

Proof. This proof is identical to theorem 2.2.6 □

2.4 Downwards Monotone Set Cover Game

Recall from Definition 3 that a function $f : 2^V \to \{0, 1\}$ is downwards monotone if $\forall S, S' \subseteq V$ where $S' \subseteq S$ we have $f(S) \leq f(S')$. Note the edge-cover cut requirement functions were downwards monotone.

Consider an extension of the edge-cover game where we are given an undirected graph $G = (V, E)$ with a cost c_e for each edge $e \in E$. Each player i of the the player set P owns a set of vertices $O_i \subset V$ not necessarily disjoint from other players' sets. Again let $O = \cup_{i\in P} O_i$ represent the set of nodes owned by at least one player.

In one iteration of the algorithm, with current player set $Q \subseteq P$ and active nodes $O_Q = \cup_{i \in Q} O_i$, a feasible solution to the downwards monotone set cover game is a set of edges $F \subseteq E$ such that $\delta(S) \cap F \neq \emptyset, \forall S \subseteq O_i, \forall i \in Q$. The following lemma gives us a more intuitive way of thinking about a feasible solution to this problem.

Lemma 2.4.1. *A set of edges F is feasible for the downwards monotone set cover game with a set of players Q if and only if for each vertex $v \in O_i$, F contains a v, u-path for some vertex $u \notin O_i$, for all players $i \in Q$*

Proof. Suppose we have a feasible solution $F \subseteq E$ such that $\delta(S) \cap F \neq \emptyset, \forall S \subseteq O_i, \forall i \in Q$. Suppose for the sake of contradiction that there exists a vertex $v \in O_i$ such that F does not contain a v, u-path for some vertex $u \notin O_i$, for some player $i \in Q$. Let $C(v)$ represent the (maximally) connected component of F that contains v. Then $C(v) \subseteq O_i$ since we assumed F does not contain a v, u-path to any vertex $u \notin O_i$. However, since F is feasible, $\delta(C(v)) \cap F \neq \emptyset$, a contradiction to the maximality of $C(v)$.

Now suppose for each vertex $v \in O_i$, F contains a v, u-path to some vertex $u \notin O_i$, for all players $i \in Q$. Suppose for the sake of contradiction $\exists S \subseteq O_i$ such that $\delta(S) \cap F = \emptyset$ for some player $i \in Q$. But if $S \subseteq O_i$ and the cut induced by S is empty, then F cannot contain a v, u-path for any vertex $v \in S$ to a vertex $u \notin O_i$. This is also a contradiction, so the two statements of feasibility are equivalent. □

Note that in a feasible solution F two arbitrary vertices $u, v \in O_i$ for some player i need not lie in the same component of F. All that is required is that each vertex $v \in O_i$ lie in a connected component of F that contains a vertex not in O_i.

The goal is to find a minimum-cost feasible solution to this problem. In the remainder of this section, we will present the KW-2 algorithm that gives rise to a cross-monotonic cost-sharing method that is 2-budget balanced for the above network design game.

2.4.1 Downwards Monotone Cut Requirements

The connectivity demand of player i in the downwards monotone set cover game can be captured by the downwards monotone cut requirement function:

$$f_i(S) = \begin{cases} 1 & \text{if } S \subseteq O_i \\ 0 & \text{otherwise} \end{cases}$$

As usual, we define the general cut requirement function for the current iteration as:

$$f(S) = \max_{i \in Q} f_i(S)$$

Lemma 2.4.2. *The function $f(S)$ is downwards monotone.*

Proof. If $f(S) = 0$ then $f(S) \leq f(S')$ for all subsets $S' \subseteq S$ since f is a $\{0,1\}$ function. If $f(S) = 1$ then $\exists i \in Q$ such that $f_i(S) = 1$. By definition of f_i it must be that $S \subseteq O_i$, and so for all subsets $S' \subseteq S$ we have that $S' \subseteq O_i$ and thus $f_i(S') = 1$. Then $f(S') = 1 \geq f(S)$ so f is downwards monotone. □

It is interesting to note a stronger property: every downwards monotone $0,1$-function on the vertex set V can be captured using these cut requirements.

Lemma 2.4.3. *Suppose we are given a function* $f : 2^V \rightarrow \{0,1\}$ *that is downwards monotone. Then there exists a family of pairwise disjoint sets* $\mathcal{O} \subseteq 2^V$ *such that a forest* F *is feasible for* f *if and only if* $\forall O \in \mathcal{O}$ *and* $\forall S \subseteq O$*, we have* $\delta(S) \cap F \neq \emptyset$.

Proof. The construction is straightforward. The family of sets $\mathcal{O}$ we are considering is the set of maximal vertex subsets $O \subseteq V$ such that $f(O) = 1$.

Suppose F is feasible for f. Consider an arbitrary set $O \in \mathcal{O}$ and an arbitrary subset $S \subseteq O$. Since $f(O) = 1$ by construction and f is downwards monotone, we have $f(S) = 1$, which implies that $\delta(S) \cap F \neq \emptyset$, since F is feasible for f.

Now suppose we have that $\delta(S) \cap F \neq \emptyset$, for all $S \subseteq O$, for all $O \in \mathcal{O}$. Now consider some arbitrary set $S' \subseteq V$, such that $f(S') = 1$. By construction, there exists $O \in \mathcal{O}$ such that $S \subseteq O$. This implies that $\delta(S') \cap F \neq \emptyset$, which means F is feasible for f. □

These two lemmas shows that we could have equivalently defined this network design game as each person owning a general downwards monotone function. That is, each player $i \in P$ owns a set of sets $\mathcal{O} \subseteq 2^V$ such that for each $O \in \mathcal{O}$, they have $f_i(O) = 1$.

2.4.2 A Lazy Primal-Dual Method

Let $\mathcal{A} = \{S \subseteq V : f(S) = 1\}$ represent the set of violated sets. The Integer Programming problem for the downwards monotone set cover game (IP-2) is the original (IP) presented in the previous chapter using the downwards monotone cut requirement f from the previous section.

We present the KW Algorithm for the Downwards Monotone Set Cover Game (KW-2). The algorithm first calculates the Death-time $D(v)$ for each node $v \in O_Q$ by considering the cost of the shortest path from v to a vertex $u \notin O_i$. Let $P_{v,i} \subseteq E$ represent this path. We let

$$d_i(v) = \frac{1}{2} c(P_{v,i})$$

and

$$D(v) = \max_{i \in Q} d_i(v)$$

as usual. If $v \notin O_Q$, we can simply set $D(v) = 0$.

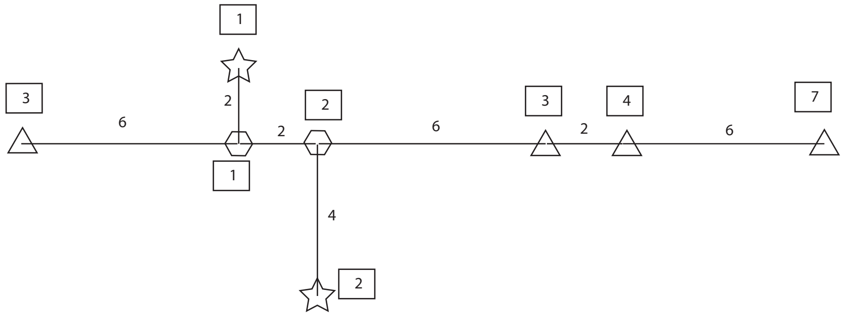

Figure 2.8: Death times included for each vertex

Here we define the crucial notion of *responsibility* introduced by Könemann, Leonardi & Schäfer in [6]. First we define a precedence order over the vertices in V. Let $v \succ w$ if $D(v) > D(w), \forall v, w \in V$. Furthermore, if $D(v) = D(w)$, then we define an arbitrary precedence order for v and w. Observe then that $v \succ w$ if and only if $D(v) \geq D(w)$.

Definition 9. *We say a vertex v is* responsible *for a set $S \subseteq V$ if $v \in S$ and $v \succ w, \forall w \in S$.*

In this fashion we can associate each set with a single vertex (of largest death-time) for ease of analysis. We will use this concept frequently as we explore the algorithm.

Once the death-times are computed we consider a fresh instance of the graph G with $y_v = 0, \forall v \in V$ and $F = \emptyset$. Just like the AKR algorithm described in Section 1.4.1, the KW-2 algorithm first uniformly raises the dual variables of every active moat until some edge becomes tight. We deviate from the previous algorithm here by insisting we only add edges contained in paths that become tight as a result of two active moats colliding.

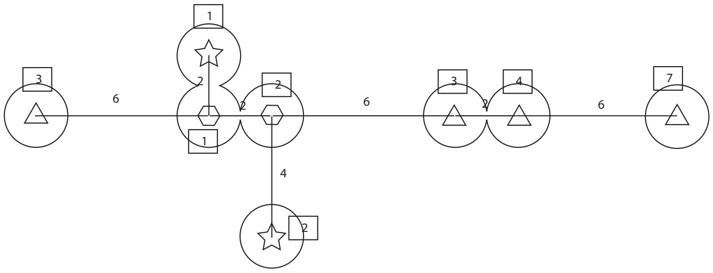

Figure 2.9: KW-2 algorithm at the first collision

Recall we let $\mathcal{U}^\tau$ represent the set of moats, i.e., the set of connected components of $\bar{F^\tau}$, the forest of tight edges at time τ.

Recall our definition of collision from Section 1.4.1. We say that moats S_1 and S_2 *collide* at time τ if

1. S_1 is an active moat at time τ ($S_1 \in \mathcal{U}^\tau$ and S_1 contains an active vertex v with $D(v) \geq \tau$)
2. S_2 is an active moat at time τ
3. τ is the first time during the execution of the algorithm at which forest $\bar{F}^\tau$ contains a connected component S such that $S_1 \subseteq S$ and $S_2 \subseteq S$.

Suppose two moats S_1, S_2 of $\bar{F}^\tau$ collide at time τ, where v_1 is responsible for S_1 and v_2 is responsible for S_2. If $D(v_1) \geq \tau$ and $D(v_2) \geq \tau$ then we add the shortest path $P \subseteq E$ between S_1 and S_2 to the forest F^τ. If either of the moats is inactive (either $D(v_1) < \tau$ or $D(v_2) < \tau$) then we do not add this path.

We also do not add paths that create cycles. If multiple moats collide simultaneously at time τ then we process the collisions in an arbitrary order (we may join S_1 to S_2, and then S_3 to the union of S_1 and S_2). These two modifications will allow us to do away with the reverse delete-step without affecting the performance guarantee of the algorithm.

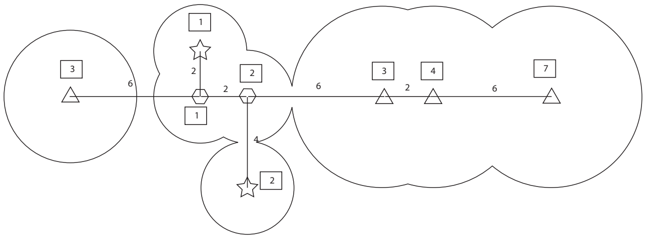

Figure 2.10: The KW-2 algorithm in progress

The KW-2 algorithm then continues to uniformly raise the dual variables of active moats until a time τ^* when there are no more active vertices. Let $F^0 = F^{\tau^*}$.

Next the KW-2 algorithm buys paths to connect any unsatisfied connected components of F^0. As before, this will be an iterative process where we add at most p paths. At each step, among all the connected components C of the current forest with $f(C) = 1$, we pick the component C_1 whose responsible vertex has the lowest precedence order. We then buy a path that connects C_1 to another component C_2 of the current forest such that at least one player $i \in P$ that had $f_i(C_1) = 1$ has $f_i(C_1 \cup C_2) = 0$. We will describe this process in more detail below, and show that such a path always exists.

The actual number of paths p we buy depends on the order we add the paths, but we can guarantee that $p \leq |\mathcal{A}^{\tau^*}|$, where $|\mathcal{A}^{\tau^*}|$ is the number of unsatisfied connected components of the forest F^{τ^*}. Let F^k represent the forest after we have added the k^{th} path. For $0 \leq k \leq p$, let $\mathcal{A}^k = \{S \in \mathcal{A} : S$ is a connected component of $F^k\}$, i.e. the set of unsatisfied connected components.

We define a precedence order for the sets in $\mathcal{A}^k$ by using their responsible vertices. Specifically, if v_1 is responsible for $C(v_1) \in \mathcal{A}^k$ and v_2 is responsible for $C(v_2) \in \mathcal{A}^k$ with $v_1 \succ v_2$ then $C(v_1) \succ C(v_2)$.

Note that $f(C) = 1$ for all components $C \in \mathcal{A}^k$. Hence, each component $C \in \mathcal{A}^k$ contains a vertex $v \in O_i$ for a player $i \in Q$ such that $f_i(C) = 1$. Beginning with the *lowest ranked* component of this precedence order, say $C(v)$, with responsible vertex v of *smallest* $D(v)$, we identify the path $P_{v,j}$ which was used to define $d_j(v)$ for one of the players j such that $f_j(C(v)) = 1$ and $d_j(v) = D(v)$. Since v is one endpoint of this path, let $u \notin C(v)$ be the other endpoint of this path. Since $P_{v,j}$ was used to define $d_j(v)$, we know that $u \notin O_j$, so by downwards monotonicity, $f_j(C(v) \cup \{u\}) = 0$. We add the path $P_{v,j}$ by setting $F^{k+1} = F^k \cup P_{v,j}$.

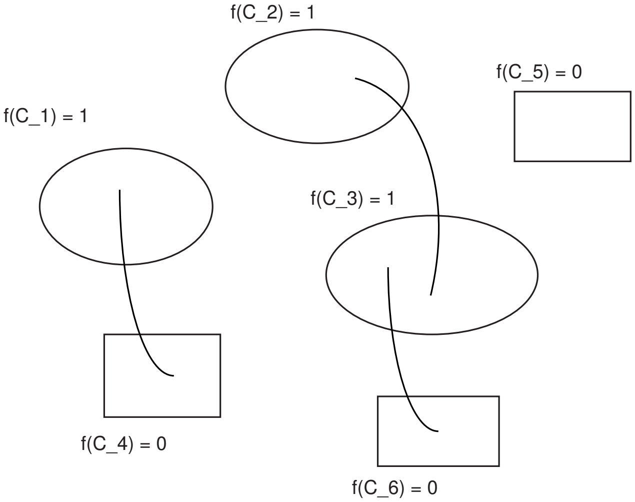

Figure 2.11: A sketch to illustrate why buying one path per unsatisfied component is always enough.

It is important when buying subsequent paths that we not create a cycle of paths connecting components of $\mathcal{A}^k$. We accomplish this goal by considering connected components of $\mathcal{A}^k$, instead of the intermediate forest F^{τ^*}. The reader will note in the formal presentation of the algorithm that we update $\mathcal{A}$ to represent the unsatisfied connected components of F^k. We repeat this process until we have $\mathcal{A}^p = \emptyset$ and the edgeset F^p. The following lemma shows that the final forest F^p is a feasible solution to the given network design game:

Lemma 2.4.4. *If $\mathcal{A}^0$ has p components, then the path-buying phase terminates after buying at most p paths (one path for each component $C \in \mathcal{A}^0$), and $F^p \subseteq E$ has the property that $\delta(S) \cap F^p \neq \emptyset, \forall S \subseteq O_i, \forall i \in Q$*

Proof. We proceed by mathematical induction on the number of paths added.

The induction hypothesis: Let $\mathcal{A}^k$ be the set of violated sets after k paths have been added. Then $\mathcal{A}^k$ has at most $p - k$ components.

The base case: take $k = 0$. Then by definition $\mathcal{A}^0$ has p components, so the base case holds.

We assume the induction hypothesis holds for $0 < k < p$, and we consider the point in the algorithm after we have bought the first k paths. By the inductive hypothesis, we suppose $\mathcal{A}^k$ has at most $p - k$ components, $C_1, ..., C_{p-k}$, whose responsible vertices are $v_1, ..., v_{p-k}$, respectively. The algorithm will find the responsible vertex (from this list) of *lowest* precedence order, which we assume is v_1 without loss of generality.

Case 1: If the path $P_{v_1,j}$ for some player j bought to satisfy C_1 connects to one of the other components in $\mathcal{A}^k$, say C_2, then the new forest F^{k+1} has one less component than F^k. The new component $C_1 \cup V(P_{v_1,j}) \cup C_2$ may still be unsatisfied, and thus in the set $\mathcal{A}^{k+1}$. However, while $\mathcal{A}^k$ has $p - k$ components, $\mathcal{A}^{k+1}$ has at most $p - k - 1$ components. Notably, if the new component $C_1 \cup V(P_{v_1,j}) \cup C_2$ is in fact satisfied, $\mathcal{A}^{k+1}$ will have at most $p - k - 2$ components.

Case 2: If the path $P_{v_1,j}$ bought to satisfy C_1 connects to a connected component S of F^k with $f(S) = 0$, then by downwards monotonicity $f(C_1 \cup S) = 0$. The new forest F^{k+1} has one less component than F^k, and $\mathcal{A}^{k+1}$ has $p - k - 1$ components. In either case, we have reduced the number of unsatisfied components by at least one. The result follows by mathematical induction, as each path bought reduces the number of unsatisfied components by at least one. □

2.4.3 Analysis: Cost Recovery

Recall we say a vertex v is responsible for a set $S \in V$ if $v \in S$ and $v \succ w, \forall w \in S$. Let $C^\tau(v)$ be the connected component of F^τ at time τ that contains v. Following the example of [6] we define an indicator variable:

$$r^\tau(v) = \begin{cases} 1 & \text{if } v \text{ is responsible for the component} C^\tau(v) \\ 0 & \text{otherwise} \end{cases}$$

and define the *responsibility time* of vertex $v \in O_Q$ as:

$$r(v) = \int_{\tau=0}^{D(v)} r^\tau(v) d\tau$$

Just as in [6], we can show that each vertex $v \in O_Q$ is responsible for a unique moat at all times $0 \leq \tau < r(v)$.

Lemma 2.4.5. *For a vertex $v \in O_Q$ with responsibility time $r(v)$, v is responsible for the connected component $C^\tau(v)$ at all times $0 \leq \tau < r(v)$ in the execution of KW-2.*

Proof. Suppose, for the sake of contradiction, that there is a time τ' when v is not responsible for $C^{\tau'}(v)$, for $0 \leq \tau' < r(v)$. But the connected component $C^{\tau'}(v)$ contains the active vertex v, so it must be some other vertex $u \in O_Q$ is

Algorithm 5 Primal-Dual KW-2 algorithm for downwards monotone set cover game

```
Given: $D(v)$ for each $v \in O_Q$
$y \leftarrow 0$
$F \leftarrow \emptyset$
repeat
    $\mathcal{A} \leftarrow \{C \subseteq V : C$ is a connected component (moat) of $\bar{F}$, $\exists v \in C$ with $\sum_{S:v \in S} y_S < D(v)\}$
    repeat
        Raise $y_S$ uniformly $\forall S \in \mathcal{A}$
    until An path becomes tight between two active moats $S_1, S_2 \in \mathcal{A}$
    $P \leftarrow$ the shortest path from $S_1$ to $S_2$
    $F \leftarrow F \cup E(P)$
until $\mathcal{A} = \emptyset$
repeat
    $\mathcal{A} \leftarrow \{C \subseteq V : C$ is a connected component with $f(C) = 1\}$
    $C \leftarrow$ the component of lowest precedence order of $\mathcal{A}$
    $v \leftarrow$ the vertex responsible for $C$
    $j \leftarrow$ an arbitrary player in $Q$ such that $d_j(v) = D(v)$ and $f_j(C) = 1$.
    $P_v \leftarrow$ the path connecting $v$ to the closet vertex $u \notin C$ where $f_j(C \cup \{u\}) = 0$.
    $F \leftarrow F \cup P_v$
until $\mathcal{A} = \emptyset$
Return: $(F, y)$
```

responsible for $C^{\tau'}(v)$ at time τ'. Then $u \succ v$ and $D(u) \geq D(v)$. However, u and v are both contained in $C^{\tau'}(v)$ and will continue to be in the same moat for all times $\tau \geq \tau'$. Thus v will never be responsible again, implying $r(v) < \tau'$, a contradiction. □

This lemma also implies that once a vertex stops being responsible, it will never be responsible again during the execution of KW-2.

We use the notion of responsibility time to associate the dual growth of a set with the unique vertex that is responsible for the set. Let $\mathcal{S}(v)$ represent the set of sets $S \subseteq V$ such that v is responsible for S. Since each active set contains exactly one active vertex that is responsible for it, it is clear from the definition of responsibility time that for any connected component C of the forest F^0 we have:

$$\sum_{S \in S(v): v \in C} y_S = \sum_{v \in C} r(v) \tag{2}$$

This equation will prove to be very useful later.

For now, consider the set of edges F^0 added during the dual-growth stage of KW-2 and the set of connected components

$$\mathcal{C}^0 = \{C \subseteq V : C \text{ is a connected component induced by the edgeset } F^0\}$$

We partition the vertices of O_Q into two sets: those that are responsible for a connected component $C \in \mathcal{C}^0$ and those that are not. Let

$$X = \{v \in O_Q : v \in C \in \mathcal{C}^0, \exists u \in C \text{ such that } u \succ v\}$$

represent the set of vertices that are not responsible for the connected components in $\mathcal{C}^0$. We will show that the total cost of the edges added during the dual-growth stage of KW-2, denoted $c(F^0)$, satisfies $c(F^0) \leq 2\sum_{v \in X} r(v)$. We will then show the path bought to satisfy each unsatisfied connected component has cost at most twice the responsibility time of the vertex responsible for that component.

To show the first result, we need the following lemma from [1]. Let $\mathcal{C}^\tau$ represent the set of connected components induced by the forest F^τ at time τ. Consider a single component $C \in \mathcal{C}^\tau$ and the set of edges $F_C^\tau = \{e = uv \in F^\tau : u, v \in C\}$. We let $c(F_C^\tau)$ denote the total cost of these edges. Clearly, the edges of F_C^τ form a tree (since we avoid adding edges that form cycles). Let u be the vertex responsible for C.

Lemma 2.4.6. *For all times* $0 \leq \tau \leq \tau^*$ *and any connected component* $C \in \mathcal{C}^\tau$ *we have* $c(F_C^\tau) \leq 2\sum_{S \in S(v): v \in C} y_S^\tau - 2\max(\tau, D(u))$

Proof. Proof is contained in [1] and omitted here. Notably, we have replaced their notation of *age*(C) with the equivalent $\max(\tau, D(u))$. □

Now we return to the result we were first trying to prove:

Lemma 2.4.7. *The total cost of the edges added during the dual-growth stage of KW-2, denoted $c(F^0)$, satisfies $c(F^0) \leq 2\sum_{v \in X} r(v)$*

Proof. Consider a arbitrary connected component $C \in \mathcal{C}^0$, where the reader will recall $\mathcal{C}^0$ is the set of connected components after the dual growth step. Let u be the vertex responsible for C. Let $M^{D(u)}(C)$ represent the moat that contains C at time $D(u)$. Since $M^{D(u)}(C)$ was active up to and including time $\tau = D(u)$, the previous lemma gives

$$c(F_C^{D(u)}) \leq 2 \sum_{S \in S(v): v \in C} y_S^{D(u)} - 2D(u)$$

Note that the moat $M^{D(u)}(C)$ that contains C at time $D(u)$ became inactive at time $D(u)$, but it may be that $M^{D(u)}(C)$ is a subset of a larger moat at the end of the dual-growth phase. This could happen if an edge became tight between $M^{D(u)}(C)$ and some active moat at some time $\tau > D(u)$, but this would not trigger a collision as $M^{D(u)}(C)$ would be inactive by assumption. Thus, the edges of $F_C^{D(u)}$ are the only edges we will charge to the dual growth of sets whose responsible vertices are in C. would not trigger a collision as $M^{D(u)}(C)$ would be inactive by assumption. Thus, the edges of $F_C^{D(u)}$ are the only edges we will charge to the dual growth of sets whose responsible vertices are in C.

Since we know C is a component of the forest F^{τ^*}, let $F_C = F_C^{D(u)} = F_C^{\tau^*}$, the tree spanning C after the dual growth phase. Now we can use equality (2) to observe

$$c(F_C) \leq 2 \sum_{v \in C} r(v) - 2D(u)$$

Note the responsible vertex u is responsible for C at time $\tau = D(u)$ so we know $r(u) = D(u)$. If we separate this term from the summation we get:

$$\begin{array}{rcl} c(F_{M(C)}) & \leq & 2\sum_{v \in (C - \{u\})} r(v) + 2r(u) - 2D(u) \\ & \leq & 2\sum_{v \in (C - \{u\})} r(v) \end{array}$$

and summing over every moat $C \in \mathcal{C}^0$ gives the result, since X is just the set of vertices not responsible for a component $C \in \mathcal{C}^0$. $\square$

Next we bound the cost of the paths bought during the path-buying phase using the responsibility times of the vertices responsible for the components $\mathcal{C}^0$. Let

$$Z = \{v \in O_Q : v \text{ is responsible for some component } C \in \mathcal{C}^0\}$$

so that $X \cup Z = O_Q$ and $X \cap Z = \emptyset$. Recall P_v is the path bought to satisfy the component $C(v)$ whose responsible vertex is v. Let $P = \cup_{v \in Z} P_v$ represent the set of paths bought during the path buying phase.

Lemma 2.4.8. *The total cost of the edges bought during the path buying phase, denoted $c(P)$, satisfies $c(P) \leq 2\sum_{v \in Z} r(v)$*

Proof. Since vertex v is responsible for component $C(v)$ at the end of the dual-growth phase we have that $r(v) = D(v)$ for all $v \in Z$. When we add path $P_{v,j}$ from v to a vertex u in another component $C(u)$ of the current forest, where $f_j(C(v) \cup \{u\}) = 0$ for a player $j \in Q$ with $f_j(C(v)) = 1$ and $d_j(v) = D(v)$, we know it costs at most $2D(v)$ by the way we choose $P_{v,j}$. Thus each path $P_{v,j}$ has the property that $c(P_{v,j}) \leq 2D(v) = 2r(v), \forall v \in Z$.

Using result 2.4.4 we know that we buy at most one path per component. Furthermore, in the situation where a path connects two unsatisfied components, the order in which we buy the paths guarantees that when we buy a path for the responsible vertex of lowest precedence order from the components of $\mathcal{A}^k$, this vertex is removed from the list of responsible vertices of components of $\mathcal{A}^{k+1}$. In this manner we can be sure that the responsible vertices we are using to pay for paths is a subset of the original set Z. Furthermore, the $2r(v)$ savings from each connected component of the intermediate forest F^{τ^*} are used at most once. The result:

$$c(P) \leq 2 \sum_{v \in Z} r(v)$$

follows after summing over all components $C \in \mathcal{C}^0$. □

Together the previous two lemmas give us the following result:

Theorem 2.4.9. *Suppose the KW-2 algorithm outputs the forest F and the duals $\{y_S\}_{S \subseteq V}$. Then $c(F) \leq 2 \sum_{S \subseteq V} y_S$*

Proof. Since $X \cup Z = O_Q$ and $X \cap Z = \emptyset$ we have that

$$2 \sum_{v \in X} r(v) + 2 \sum_{v \in Z} r(v) = 2 \sum_{v \in O_Q} r(v)$$

and using the equation $\sum_{v \in C} r(v) = \sum_{S \subseteq C} y_S$ for every component $C \in \mathcal{C}^0$ gives

$$2 \sum_{v \in O_Q} r(v) = 2 \sum_{S \subseteq V} y_S$$

Finally, note that the forest $F = F^0 \cup P$ so that

$$c(F) = c(F^0) + C(P) \leq 2 \sum_{v \in X} r(v) + 2 \sum_{v \in Z} r(v) = 2 \sum_{S \subseteq V} y_S$$

as required. □

2.4.4 Analysis: Cross-monotonic Cost-Sharing Method

Consider the following cost-sharing method that follows the methodology of [6] derived from the KW-2 Algorithm: when we grow the dual variable of an active moat we share the cost of this growth among all active vertices contained in the moat. Recall we let $M^\tau(v)$ be the connected component of $\bar{F^\tau}$ that contains vertex v, i.e. that active moat that contains v. Now let $a^\tau(v)$ be the number of active vertices contained in $M^\tau(v)$, i.e., the number of vertices $u \in M^\tau(v)$ with $D(u) \geq \tau$. Furthermore, we share the cost-share of one vertex among all players for whom the vertex is still active. Let $b^\tau(v)$ be the number of players j for whom $d_j(v) \geq \tau$. Now we can define the cost-share of vertex $v \in O_Q$ for player $i \in Q$ as

$$\xi^i_Q(v) = \int_{\tau=0}^{d_i(v)} \frac{1}{a^\tau(v) b^\tau(v)} d\tau$$

The cost-share of player $i \in Q$ is just the sum of the cost-shares of their terminals:

$$\xi_Q(i) = \sum_{v \in O_i} \xi^i_Q(v)$$

We will show the cost-sharing method ξ is a cross-monotonic and 2-budget balanced cost-sharing method for the downwards monotone set cover game.

To prove cross-monotonicity, we study the effect of the removal of one player on the cost-shares of the remaining players. Consider an arbitrary player $i \in Q$ and let $Q_0 = Q - \{i\}$. Let KW-2(Q) represent the run of the KW-2 algorithm using the set of players Q and KW-2(Q_0) represent the run of the KW-2 algorithm using the set of players Q_0. Suppose KW-2(Q) has moats $\mathcal{U}^\tau$ induced by the forest $\bar{F^\tau}$ and terminates at time τ^*; while KW-2(Q_0) has moats $\mathcal{U}^\tau_0$ induced by the forest $\bar{F^\tau_0}$ and terminates at time τ^*_0. The following lemma is from [7] and shows that $\mathcal{U}^\tau_0$ is a *refinement* of $\mathcal{U}^\tau$:

Lemma 2.4.10. *For all times $0 \leq \tau \leq \tau*$ and for every moat $U_0 \in \mathcal{U}^\tau_0$ there must be a set $U \in \mathcal{U}^\tau$ such that $U_0 \subseteq U$.*

Proof. We proceed by mathematical induction on the time τ.

The claim is trivially true at time $\tau = 0$ since $\mathcal{U}^\tau = \mathcal{U}^\tau_0$.

Now suppose the claim holds for some time $0 \leq \tau \leq \tau*$ and consider what happens when we grow the duals of active moats from τ to $\tau' = \tau + \epsilon$ for a small $\epsilon > 0$. Let $U_0 \in \mathcal{U}^\tau_0$ be an active moat at time τ in the run of KW-2(Q_0) and let $v \in U_0$ be an active vertex ($d_j(v) \geq \tau$ for some player $j \in Q_0$). Thus KW-2(Q_0) grows the moat U_0 from time τ to τ'. By the induction hypothesis, we know at time τ that $\exists U \in \mathcal{U}^\tau$ such that $U_0 \subseteq U$. Then v is active in the run of KW-2(Q) and contained in the moat U, so U is an active moat. Thus KW-2(Q) grows the dual of U at time τ to time τ' and the claim follows. $\square$

This lemma implies cross-monotonicity, as the removal of one player can not possibly reduce the cost-shares of the remaining players. To see this explicitly, let $\xi^i_{Q_0}(v)$ represent the cost-share of vertex v for player i with player set Q_0.

Lemma 2.4.11. *The cost-sharing method ξ arising from KW-2 is cross-monotonic, i.e., for each player $j \in Q_0$ we have $\xi_{Q_0}(j) \geq \xi_Q(j)$*

Proof. We will prove a stronger result, that is for each player $j \in Q_0$ and each vertex $v \in O_{Q_0}$ we have $\xi^j_{Q_0}(v) \geq \xi^j_Q(v)$. Recall we let $U^\tau(v)$ be the moat that contains v in the execution of KW-2(Q), so similarly let $U_0^\tau(v)$ be the moat that contains v in the execution of KW-2(Q_0). Recall we let $a^\tau(v)b^\tau(v)$ be the number of active vertices contained in $U^\tau(v)$ times the number of players for whom v is still active, so similarly let $a_0^\tau(v)b_0^\tau(v)$ be the number of active vertices contained in $U_0^\tau(v)$ times the number of players for whom v is still active.

The previous lemma shows that $U_0^\tau(v) \subseteq U^\tau(v)$, so it must be that $a_0^\tau(v) \leq a^\tau(v)$ and $b_0^\tau(v) \leq b^\tau(v)$ for all times $0 \leq \tau \leq \tau *$. Thus

$$\xi_Q^j(v) = \int_{\tau=0}^{d_j(v)} \frac{1}{a^\tau(v)b^\tau(v)} d\tau \leq \int_{\tau=0}^{d_j(v)} \frac{1}{a_0^\tau(v)b_0^\tau(v)} d\tau = \xi_{Q_0}^j(v)$$

for all players $j \in Q_0$ and vertices $v \in O_{Q_0}$. Summing over all vertices proves cross-monotonicity. □

Note that the sum of the cost shares $\sum_{i\in Q} \xi_Q(i) = \sum_{S\subseteq V} y_S$, the sum of the duals by construction. Recall that $\sum_{v\in O_Q} r(v) = \sum_{S\subseteq V} y_S$, so we have the following useful equation:

$$\sum_{i\in Q} \xi_Q(i) = \sum_{v\in O_Q} r(v) \tag{3}$$

We will prove the cost-sharing method is competitive by using the responsibility times of the vertices in a tree of the optimal solution. First we will need a supporting lemma which shows that every maximal tree T of an optimal solution has diameter at least twice the death-time of v, the tree's responsible vertex.

Lemma 2.4.12. *Let $P \subseteq T$ be the longest path in the tree that contains the responsible vertex v. Then the cost of the path P must satisfy*

$$c(P) \geq 2D(v)$$

Proof. Recall $D(v) = \max_{i\in Q} d_i(v)$, so there exists a player $j \in Q$ for which $d_j(v) = D(v)$. By construction, $d_j(v)$ is half the length of the shortest path from v to a vertex $u \notin O_j$. By result 2.4.1 we know a feasible solution to our problem requires v to lie in a tree that contains a vertex not in O_j. Thus, an optimal tree T that contains v must also contain a vertex not in O_j, of which we know u is the closest. It may be that u is not in T, but we do know that any path from v to a vertex not in O_j has cost at least $2D(v)$. Thus the longest path $P \subseteq T$ that contains v must have cost at least $2D(v)$. □

Lemma 2.4.13. *The cost-sharing method ξ arising from KW-2 is competitive, i.e., $\sum_{i\in Q} \xi_Q(i) \leq opt_Q$*

Proof. Consider the optimal forest F_{opt} and a maximal tree T in the optimal solution. Let $V(T)$ represent the vertices of T. Let v be the vertex responsible for T, i.e., the vertex of highest death-time in $V(T)$. Let $\mathcal{M}_T^\tau$ represent the set of active moats whose responsible vertices are in $V(T)$. Recall we let $M^\tau(v)$

represent the active moat that contains v. It is important to note that if v is responsible for $M^\tau(v)$ and $u \neq v$ is responsible for $M^\tau(u)$, then the two moats must be disjoint. A short proof is provided in [7]. If the active moats $\mathcal{M}_T^\tau$ intersect the tree T at all times $\tau \leq \tau^*$, then by the disjointness property we have that $\sum_{u \in V(T)} r(u) \leq c(T)$.

However, once $|\mathcal{M}_T^\tau| = 1$, it may be that the last active moat $M^\tau(v)$ outgrows T. Let $\tau' < \tau^*$ represent the last time that the moat $M^{\tau'}(v)$ containing v intersected the tree T in the run of KW-2. It must be that $V(T) \subseteq M^{\tau'}(v)$. In this case, we can still show that the sum of the responsibility times of the vertices in $V(T)$ does not exceed the total cost of the tree by tracking the active moats which intersect T in two or more places simultaneously.

We let $V(P)$ represent the vertices on the longest path P in T that contains v. The previous result 2.4.12 implies that $c(P) \geq 2D(v)$. Now consider an arbitrary active moat $M^\tau(u)$ whose responsible vertex is $u \in V(P)$, for some time $\tau < \tau'$. By assumption $|M^\tau(u) \cap T| \geq 1$ for all times $\tau < \tau'$. We are particularly interested in the cases where $|M^\tau(u) \cap T| > 1$.

For a moat $M^\tau(u)$ with responsible vertex $u \in O_Q$, let $P^\tau(u) = |\delta(M^\tau(u)) \cap P|$. That is, $P^\tau(u)$ counts the number of edges of the path P that the moat $M^\tau(u)$ intersects at time τ.

Now we consider the sum of the responsibility times of the vertices up until time τ'. Let $\rho = \sum_{u \in V(T)} \max\{r(u), \tau'\}$ represent this quantity. Since the moats that the vertices in the tree are responsible for intersect the tree until time τ', we can say

$$c(T) \geq \rho + \int_0^{\tau'} (\sum_{u \in O_Q} \max\{0, P^\tau(u) - 1\}) d\tau \tag{2.4.1}$$

where $\int_0^{\tau'} (\sum_{u \in V} P^\tau(u) - 1) d\tau$ are the savings we get from moats that intersect P in more place than one.

Observe that the path P must be tight at time τ' since we assumed that τ' is the last time the moat $M^{\tau'}(v)$ containing the responsible vertex v intersected the tree T. This implies that

$$c(P) = \int_0^{\tau'} (\sum_{u \in O_Q} P^\tau(u)) d\tau$$

Further observe that since a path has only two endpoints, the number of moats that intersect P in exactly one place at any time $0 \leq \tau \leq \tau'$ is at most two. By pulling the terms corresponding to these moats out of the summation we get the following expression

$$c(P) \leq 2\tau' + \int_0^{\tau'} (\sum_{\{u \in O_Q : P^\tau(u) \geq 2\}} P^\tau(u)) d\tau$$

Now every term in the summation is at least two, so we can subtract one from each term and multiply by 2 to get

$$c(P) \leq 2\tau' + 2\int_0^{\tau'} (\sum_{u \in O_Q} \max\{0, P^\tau(u) - 1\})d\tau$$

We can rearrange this inequality as follows

$$\int_0^{\tau'} (\sum_{u \in O_Q} \max\{0, P^\tau(u) - 1\})d\tau \geq \frac{c(P) - 2\tau'}{2}$$

which, combined with 2.4.1 gives us

$$c(T) \geq \frac{c(P) - 2\tau'}{2} + \rho$$

Finally, recall the previous result 2.4.12 implies that $D(v) \leq \frac{c(P)}{2}$, so we have

$$c(T) \geq D(V) - \tau' + \rho$$

We know the growth of the final moat after time τ' is at most $D(V) - \tau'$. Recall ρ is the sum of the responsibility times up to time τ'. Thus, we have shown that the time the last active moat spends growing beyond the tree T can be paid for by the savings we get from the active moats that intersect the path P in two or more places at once. Thus

$$\sum_{u \in V(T)} r(u) \leq c(T)$$

and summing over all trees T of F gives

$$\sum_{i \in Q} \xi_Q(i) = \sum_{u \in V} r(u) \leq c(F) = opt_Q$$

where the first equality comes from the distribution of cost-shares. □

We are now ready for the final result of this section:

Theorem 2.4.14. *Suppose that algorithm KW-2 outputs a forest F and a dual solution $\{y_v\}_{v \in O_Q}$. Then ξ is a 2-budget balanced cross-monotonic cost-sharing method, i.e.*

$$\frac{1}{2}c(F) \leq \sum_{i \in Q} \xi_Q(i) \leq opt_Q$$

Proof. Result 2.4.9 and 2.4.13 imply 2-budget balanced since $\sum_{i \in Q} \xi_Q(i) = \sum_{S \subseteq V} y_S$, while result 2.4.11 implies cross-monotonicity. The theorem follows. □

2.5 Even Parity Connection Game

A function f is *proper* if it satisfies both the maximality property and the symmetry property: $f(S) = f(V \backslash S)$ for all $S \subseteq V$. It is also necessary that we assume $f(V) = 0$. Proper functions encompass a wide variety of network design problems, including Steiner Tree, T-Joins and Perfect Matchings. The work of Könemann, Leonardi, Schäfer, and van Zwam in [7] addresses the Steiner Forest problem whose associated cut requirement function is proper.

The minimum-weight perfect matching problem is the problem of finding a minimum cost set of edges that cover all the vertices, with each vertex incident to exactly one edge. The T-Join problem considers an even subset $T \subseteq V$ of vertices with the goal of finding a minimum cost set of edges that has odd degree at vertices in T and even degree at all other vertices.

Consider a network design game based on the T-Join problem where we are given an undirected graph $G = (V, E)$ with a cost c_e for each edge $e \in E$. Each player i of the the player set P owns a set of vertices $O_i \subseteq V$ not necessarily disjoint from other players' sets, where $|O_i|$ is *even* for all $i \in P$. Again let $O = \cup_{i \in P} O_i$ represent the set of nodes owned by at least one player.

As usual, we will be considering one iteration of the algorithm, with current player set $Q \subseteq P$ and the set of nodes owned by the players $O_Q = \cup_{i \in Q} O_i$. Before we can define a feasible solution we need to explore the idea of a T-cut. A set $S \subseteq V$ is a *T-cut* if $|O_i \cap S|$ is odd for some player $i \in Q$. Formally, we can define the set of T-cuts $\mathcal{T}_Q$ for the player set Q as:

$$\mathcal{T}_Q = \{S \subseteq V : \exists i \in Q \text{ such that } |O_i \cap S| \text{ is odd}\}$$

Now we can say a feasible solution to the even parity connection game is a set of edges $F \subseteq E$ such that for all T-Cuts $S \in \mathcal{T}_Q, \delta(S) \cap F \neq \emptyset$.

The following cut requirement for a player $i \in P$ arises naturally from T-cuts:

$$f_i(S) = \begin{cases} 1 & \text{if } |S \cap O_i| \text{ is odd} \\ 0 & \text{otherwise} \end{cases}$$

Since every player owns an even number of nodes, the function f_i satisfies symmetry for each player $i \in P$. As well, if $f_i(A) = f_i(B) = 0$ for disjoint sets $A, B \subseteq V$, then neither set is a T-Cut for player i, so $A \cup B$ is also not a T-Cut for player i. This implies the function f_i is indeed proper, as shown by Goemans and Williamson in [3].

As usual, we define the general cut requirement function for the current iteration as:

$$f(S) = \max_{i \in Q} f_i(S)$$

Note that a feasible solution F may not be a T-Join for the set of nodes O_Q in the traditional sense. However, a feasible solution does guarantee that each connected component induced by F contains an even number of vertices owned by each player.

The goal is to find a minimum-cost feasible solution to this problem. In the remainder of this section, we will modify the KW-2 algorithm into a cross-monotonic cost-sharing method that satisfies 2-approximate cost recovery for the above network design game. We conjecture that the method is competitive, and thus 2-budget balanced, but a proof is not provided in this thesis.

2.5.1 Primal-Dual Method

Recall we let $\mathcal{T}_Q$ represent the set of T-cuts. The Integer Programming problem for the even parity connection game fits the framework of the general network design problem (IP) using the above cut requirement function f.

Recall the AKR & GW algorithms discussed earlier. AKR was designed for the Steiner Forest problem while GW generalized the framework to run on any proper cut requirement. In the special case of the Steiner Forest problem, the algorithms return the same solutions to both primal and dual problems. The difference lies in how edges are added. While GW adds many edges and then removes unnecessary edges in a reverse delete step, AKR selectively adds edges and retains them all for the final solution. We adapt the AKR method of only adding certain paths (and not performing a reverse delete step) into the KW-3 algorithm specialized for the Even Parity Connection game.

To calculate the death-times in this game we cannot simply use shortest paths as in the previous algorithms. Instead, we have to use the more general idea presented in Section 2.1 and run the GW algorithm once for each player $i \in P$.

Let GW(i) represent the run of the GW algorithm where we grow only the moats that contain an odd number of vertices owned by player i. This is equivalent to only raising the dual of a moat U if $f_i(U) = 1$. Recall we say a moat U is active if $f_i(U) = 1$. The run of GW(i) for a particular player i begins by uniformly growing the dual y_v for each of the vertices $v \in O_i$. At some time $\tau \geq 0$, two (or more) active moats will collide when a path between two (or more) vertices becomes tight.

We seek to pair up the vertices of O_i through this process. To this end, if more than two active moats collide simultaneously, we choose two appropriate moats and pair the vertices up arbitrarily among active vertices of the colliding moats that do not already have mates.

When the first collision takes place it will be between two active moats (perhaps chosen by the above rule) that each contain only one vertex owned by player i. In this situation, for a moat $U(v)$ that contains vertex $v \in O_i$ colliding with a moat $U(u)$ that contains vertex $u \in O_i$, we define u to be v's *i-mate* and v to be u's *i-mate*. For convenience, we also refer to v and u as simply *mates*. We define $d_i(v) = d_i(u) = \tau(u, v)$, where $\tau(u, v)$ is the time the active moats $U(v)$ and $U(u)$ collide during the run of GW(i).

At all times during the run of GW(i), moats that contain an even number of vertices of O_i are inactive. Moats that contain an odd number of vertices are active, although every active moat contains exactly one vertex of O_i that does not have a mate. When we process subsequent collisions of active moats, the pairing of mates is always defined naturally by pairing the two vertices of the colliding moats that do not already have mates.

We continue growing the duals of active moats (connected components of the forest $\bar{F}^{\tau}$ of tight edges) until a time τ_i^* when every vertex of O_i has a mate. When this happens we terminate the run of GW(i), store the death times computed for player i, and reset all the dual values to 0 so that we may repeat this process for the other players. Note that a single vertex v may have as many mates as the number of players that own v.

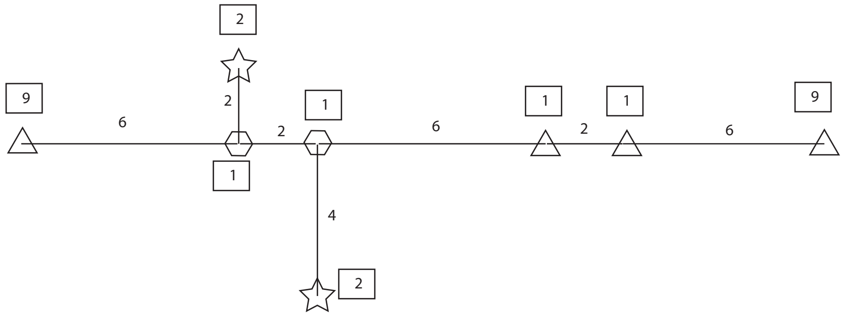

Figure 2.12: Death times for Even Parity Connection Game

Once all the death times $d_i(v)$ have been computed for each $v \in O_i, \forall i \in P$, we are ready to run KW-3. In the run of KW-3 with the set of players $Q \subseteq P$, we define

$$D(v) = \max_{i \in Q} d_i(v)$$

as usual. Just as in AKR, GW, KW-1 and KW-2, we view this algorithm as a process over time and say a vertex v is active at time τ in the run of KW-3 if $D(v) \geq \tau$. We say a moat is active in the run of KW-3 if it contains an active vertex. The KW-3 algorithm will uniformly grow the duals corresponding to active moats at all times $\tau \geq 0$.

We define a precedence order over the vertices. Let $v \succ w$ if $D(v) > D(w), \forall v, w \in V$. Otherwise, we define an arbitrary precedence order for v and w if their death-times are equal.

The execution of KW-3 is very similar to KW-2, in that we only add a path to the forest F^{τ} when two active moats collide. Specifically, when we are processing the collision of two moats U_1 and U_2 at time τ, we add the shortest path P to the forest F^{τ} such that the connected components C_1 contained in U_1 and C_2 contained in U_2 are now joined by P.

We continue growing active moats and processing collisions of active moats just as in previous algorithms, until a time τ^* when there are no more active moats. Since each run of GW(i) terminates only after satisfying the cut requirement f_i, the aggregate run of KW-3 must satisfy every cut requirement $f_i, \forall i \in Q$ and return a forest F^{τ^*} that is feasible for the Even Parity Connection game. We present the KW-3 algorithm formally below, noting there is no need to buy paths or perform a reverse-delete step (although one may by used in practice).

Algorithm 6 Primal-Dual KW-3 algorithm for even parity connection game

```
Given: $D(v)$ for each $v \in O_Q$
$y \leftarrow 0$
$F \leftarrow \emptyset$
$i \leftarrow 0$
repeat
    $\mathcal{A} \leftarrow \{U \subseteq V : U$ is a connected component (moat) of $\bar{F}$, $\exists v \in U$ with $\sum_{S:v \in S} y_S < D(v)\}$
    repeat
        Raise $y_S$ uniformly $\forall S \in \mathcal{A}$
    until A path becomes tight between two active moats $S_1, S_2 \in \mathcal{A}$ that contains connected components $C_1$ and $C_2$ of $F$
    $P \leftarrow$ the shortest path from $C_1$ to $C_2$
    $F \leftarrow F \cup E(P)$
until $\mathcal{A} = \emptyset$
Return: $(F, y)$
```

2.5.2 Analysis: Cost Recovery

The execution of KW-3 is essentially an execution of the AKR algorithm with only one major modification: we use death-times to determine how long a moat will continue growing. This is the methodology used in [6]. We will show that despite this change, we can reduce the execution of KW-3 on the Even Parity Connection game to an execution of AKR on an instance of the Steiner Forest problem.

Theorem 2.5.1. *If KW-3 returns the forest F, then F is feasible for the Even Parity Connection game and the total cost of the forest F, denoted $c(F)$, satisfies $c(F) \leq 2\sum_{S \subseteq V} y_S$*

Proof. Suppose F is infeasible and there exists a connected component C of F with $f(C_1) = 1$. This implies for some player $i \in Q$ that $f_i(C) = 1$. Thus, C must contain an odd number of vertices of O_i, so let $v \in O_i$ be the vertex of highest precedence order in C whose i-mate $u \in O_i$ is not in C.

Consider the run of GW(i) we used to define the death-times of the vertices of O_i. GW(i) terminates when every vertex of O_i has a mate, and thus returns a solution that is feasible for player i. Since we define death-times for KW-3 using GW(i), if a moat U grows in GW(i) at time τ, then there exists a vertex $v \in U$ with $d(v) > \tau$, and thus there exists a moat U' containing U that grows in KW-3 at time τ.

We know by time $\tau = d_i(v) = d_i(u)$ that v is in the same component as its i-mate u in the run of GW(i). We also know that in the run of KW-3, v

and u are both contained in active moats until time $\tau = d_i(v) = d_i(u)$ (and possibly beyond). This, together with the previous paragraph, implies that at some time $\tau' \leq \tau$, the two active moats containing v and u must have collided in the execution of KW-3. When this occurred, KW-3 would have added a path between the moats. However, this is a contradiction to our assumption that KW-3 terminated and returned the forest F with a component C that contained v and not u. Thus F is indeed feasible.

We will show that $c(F)$ is at most twice the total dual growth by a reduction to the Steiner Forest problem, using an argument found in [6]. We construct an instance of the Steiner Forest problem from the Even Parity Connection game as follows: for each vertex $v \in O_Q$, we introduce a new vertex $\tilde{v}$ and a new edge $\tilde{e}_v = v\tilde{v}$ with $c(\tilde{e}) = 2D(v)$. Let $\tilde{E}$ represent this set of new edges. We run the AKR algorithm on the new graph with terminal set $\tilde{R} = \{(v, \tilde{v}) : v \in O_Q\}$. Let $\tilde{\mathcal{S}}$ represent the set of Steiner cuts with terminal set $\tilde{R}$.

Observe that in the run of AKR the edge $\tilde{e}_v$ will become tight at time $\tau = D(v)$, for each $v \in O_Q$. This implies that the component containing v will be active in the run of AKR for precisely the same amount of time as the moat containing v in the run of KW-3. Let $\{y_S^{AKR}\}_{S \in \tilde{\mathcal{S}}}$ represent the set of duals returned by the AKR algorithm on the new graph, and let $\{y_S^{KW}\}_{S \subseteq V}$ represent the set of duals returned by the KW-3 algorithm on the original graph. We thus have

$$\sum_{S \in \tilde{\mathcal{S}}} y_S^{AKR} = \sum_{S \subseteq V} y_S^{KW} + \sum_{v \in O_Q} y_{\{\tilde{v}\}}^{AKR}$$

Furthermore, since collisions are defined in the same manner for both AKR and KW-3, when a collision takes place and we add a path in the run of KW-3, the same collision takes place and the same path is added in the run of AKR. Thus, the solution computed by AKR, restricted to the original graph, must be the same solution as the one computed by KW-3. By result 1.4.1, the forest returned by AKR on the new graph is at most twice the total AKR dual growth, that is

$$\sum_{e \in E \cup \tilde{E}} c(e)x_e \leq 2 \sum_{S \in \tilde{\mathcal{S}}} y_S^{AKR}$$

Finally, we know that each edge $\tilde{e}_v = v\tilde{v}$ is added by AKR at time $\frac{c(\tilde{e})}{2}$, which implies

$$\sum_{e \in \tilde{E}} c(e)x_e = 2 \sum_{v \in O_Q} y_{\{\tilde{v}\}}^{AKR}$$

Putting these results together gives us

$$\begin{aligned} c(F) = \sum_{e \in E} c(e)x_e = \sum_{e \in E \cup \tilde{E}} - \sum_{e \in \tilde{E}} c(e)x_e \\ \leq 2 \sum_{S \in \tilde{\mathcal{S}}} y_S^{AKR} - 2 \sum_{v \in O_Q} y_{\{\tilde{v}\}}^{AKR} = 2 \sum_{S \subseteq V} y_S^{KW} \qquad (2.5.1) \end{aligned}$$

which proves the result. □

2.5.3 Analysis: Cross-monotonic Cost-Sharing Method

We use the same cost-sharing method derived from the KW-2 Algorithm for our KW-3 Algorithm: when we grow the dual variable of an active moat we share the cost of this growth among all active vertices contained in the moat. The definitions are reproduced here for completeness. Recall we let $M^\tau(v)$ be the connected component of $\bar{F}^\tau$ that contains vertex v, i.e. that active moat that contains v. Now let $a^\tau(v)$ be the number of active vertices contained in $M^\tau(v)$, i.e., the number of vertices $u \in M^\tau(v)$ with $D(u) \geq \tau$. Furthermore, we share the cost-share of one vertex among all players for whom the vertex is still active. Let $b^\tau(v)$ be the number of players j for whom $d_j(v) \geq \tau$. Now we can define the cost-share of vertex $v \in O_Q$ for player $i \in Q$ as

$$\xi^i_Q(v) = \int_{\tau=0}^{d_i(v)} \frac{1}{a^\tau(v) b^\tau(v)} d\tau$$

The cost-share of player $i \in Q$ is just the sum of the cost-shares of her terminals:

$$\xi_Q(i) = \sum_{v \in O_i} \xi^i_Q(v)$$

We will show the cost-sharing method ξ is a cross-monotonic and recovers at least half the cost of the solution for the Even Parity Connection Game.

To prove cross-monotonicity, we again study the effect of the removal of one player on the cost-shares of the remaining players. Consider an arbitrary player $i \in Q$ and let $Q_0 = Q - \{i\}$. Let KW-3(Q) represent the run of the KW-3 algorithm using the set of players Q and KW-3(Q_0) represent the run of the KW-3 algorithm using the set of players Q_0. Suppose KW-3(Q) has moats $\mathcal{U}^\tau$ induced by the forest $\bar{F}^\tau$ and terminates at time τ^*; while KW-3(Q_0) has moats $\mathcal{U}_0^\tau$ induced by the forest $\bar{F}_0^\tau$ and terminates at time τ_0^*. The following lemma should be familiar to the reader (see Section 2.4.4) and shows that $\mathcal{U}_0^\tau$ is a *refinement* of $\mathcal{U}^\tau$:

Lemma 2.5.2. *For all times $0 \leq \tau \leq \tau*$ and for every moat $U_0 \in \mathcal{U}_0^\tau$ there must be a set $U \in \mathcal{U}^\tau$ such that $U_0 \subseteq U$.*

Proof. We proceed by mathematical induction on the time τ.

The claim is trivially true at time $\tau = 0$ since $\mathcal{U}^\tau = \mathcal{U}_0^\tau$.

Now suppose the claim holds for some time $0 \leq \tau \leq \tau*$ and consider what happens when we grow the duals of active moats from τ to $\tau' = \tau + \epsilon$ for a small $\epsilon > 0$. Let $U_0 \in \mathcal{U}_0^\tau$ be an active moat at time τ in the run of KW-3(Q_0) and let $v \in U_0$ be an active vertex ($d_j(v) \geq \tau$ for some player $j \in Q_0$). Thus KW-3(Q_0) grows the moat U_0 from time τ to τ'. By the induction hypothesis, we know at time τ that $\exists U \in \mathcal{U}^\tau$ such that $U_0 \subseteq U$. Then v is active in the run of KW-3(Q) and contained in the moat U, so U is an active moat. Thus KW-3(Q) grows the dual of U at time τ to time τ' and the claim follows. □

Again, this refinement lemma immediately implies cross-monotonicity, as the removal of one player cannot decrease the cost-share of any of the remaining players. The following lemma shows this explicitly:

Lemma 2.5.3. *The cost-sharing method ξ arising from KW-3 is cross-monotonic, i.e., for each player $j \in Q_0$ we have $\xi_{Q_0}(j) \geq \xi_Q(j)$*

Proof. We will prove a stronger result, that is for each player $j \in Q_0$ and each vertex $v \in O_{Q_0}$ we have $\xi^j_{Q_0}(v) \geq \xi^j_Q(v)$. Recall we let $U^\tau(v)$ be the moat that contains v in the execution of KW-3(Q), so similarly let $U^\tau_0(v)$ be the moat that contains v in the execution of KW-3(Q_0). Recall we let $a^\tau(v)b^\tau(v)$ be the number of active vertices contained in $U^\tau(v)$ times the number of players for whom v is still active, so similarly let $a^\tau_0(v)b^\tau_0(v)$ be the number of active vertices contained in $U^\tau_0(v)$ times the number of players for whom v is still active.

The previous lemma shows that $U^\tau_0(v) \subseteq U^\tau(v)$, so it must be that $a^\tau_0(v) \leq a^\tau(v)$ and $b^\tau_0(v) \leq b^\tau(v)$ for all times $0 \leq \tau \leq \tau*$. Thus

$$\xi^j_Q(v) = \int_{\tau=0}^{d_j(v)} \frac{1}{a^\tau(v)b^\tau(v)} d\tau \leq \int_{\tau=0}^{d_j(v)} \frac{1}{a^\tau_0(v)b^\tau_0(v)} d\tau = \xi^j_{Q_0}(v)$$

for all players $j \in Q_0$ and vertices $v \in O_{Q_0}$. Summing over all vertices proves cross-monotonicity. □

Chapter 3

Discussion

3.1 Future Work

We conjecture that the cost-sharing method ξ that arises from the KW-3 algorithm for the even parity connection game is competitive. To see the motivation for this conjecture, consider the following lifted cut relaxation.

3.1.1 Lifted Cut Relaxation

To prove the cost-sharing method ξ is competitive, we will consider a lifted-cut relaxation for our even parity connection game. Recall we let $\mathcal{T}_Q$ represent the set of T-cuts. Let $\mathcal{N}_Q$ represent the set of subsets of V that are not T-cuts. Formally, we can define $\mathcal{N}_Q$ as:

$$\mathcal{N}_Q = \{S \subseteq V : \forall i \in Q, |O_i \cap S| \text{ is even}\}$$

Let $\mathcal{S} = \mathcal{T}_Q \cup \mathcal{N}_Q$. As defined, $\mathcal{S}$ is simply the set of all possible subsets of V. In practice, it is the set of sets that may experience dual growth.

In order to create a useful lifted-cut relaxation, we form a non-disjoint partition of the T-cuts according to special vertices contained within each set. Recall we defined two vertices to be mates when they were paired together during a run of GW(i). Define:

$$T(v) = \{S \in \mathcal{T}_Q : v \text{ is responsible for } S \text{ OR } v \in S \text{ has a mate } u \notin S\}$$

We call this partition non-disjoint as a T-Cut S may belong to many such sets. For example, suppose $S = \{v, v', u, w, w'\}$ where $v \succ v' \succ w \succ w'$, v and v' are each other's unique mates, w is w''s unique mate but w has another mate not in S, and at least one of u's mate is not in S. Then $S \in T(v)$, $S \in T(u)$ and $S \in T(w)$.

We also want to partition the non T-cuts, but before we do that we need to introduce the concept of a friend. Unlike in the Steiner Forest game, in the run of KW-3, two mates may not end up in the same component of a feasible solution F. As a result, we must generalize the concept for the Even Parity Connection game.

First we say a player $i \in Q$ *controls* $v \in O_i$ if $D(v) = d_i(v)$. We can then introduce notation for a set of players that control v as

$$\mathcal{K}_Q(v) = \{i \in Q : D(v) = d_i(v)\}$$

Now for a non T-cut $S \in \mathcal{N}_Q$ and responsible vertex v we can define the set of v's *friends* $\mathcal{F}_Q(v, S)$ as

$$\mathcal{F}_Q(v, S) = \{u \in \{\cup_{i \in \mathcal{K}_Q(v)} O_i \cap S\}\}$$

The friends of v in a non T-cut S are just the set of vertices that are owned by at least one player who controls v. Note that since S isn't a T-cut, v will always have at least one friend in S.

We choose the *best friend* u of v in a non T-cut S as the most responsible friend that isn't v itself. Formally, $u \in \mathcal{F}_Q(v, S)$ is the best friend of the responsible vertex v of the set $S \in \mathcal{N}_Q$ if $\nexists w \in \mathcal{F}_Q(v, S)$ with $w \succ u$. Now we can form a cover of the non T-cuts by defining:

$$\begin{aligned} N(v) = \{S \in \mathcal{N}_Q : \ & v \text{ is responsible for } S \\ & \text{OR } v \text{ is the best friend of the responsible vertex } u\} \end{aligned} \tag{3.1.1}$$

We say this is a cover because each non T-cut S with responsible vertex v and best friend u is actually a member of two sets: $N(v) \subseteq \mathcal{N}_Q$ and $N(u) \subseteq \mathcal{N}_Q$.

We now have all the notation we need to introduce the dual of the lifted-cut relaxation for the even parity connection game:

$$\begin{aligned} \max \quad & \sum_{S \in \mathcal{S}} y_S & & & \text{(LC-D)} \\ \text{s.t.} \quad & \sum_{S \in \mathcal{S} : e \in \delta(S)} y_S & \le c_e & \quad \forall e \in E, & (6) \\ & \sum_{S \in T(v)} y_S + \sum_{S \in N(v)} y_S & \le D(v) & \quad \forall v \in O_Q, & (7) \\ & y_S \ge 0 & \forall S \in \mathcal{S}. & & \end{aligned}$$

Notice that a feasible solution to (LC-D) may assign positive values to non T-cuts, just as the algorithm may grow the duals of non T-cuts. The constraints of type 7 are required to bound the objective function of (LC-D), which sums over the duals of all subsets of V, not just the T-cuts.

We now consider the lifted-cut primal (LC-P) (dual to (LC-D)) which has an additional variable z_v for each $v \in O_Q$:

$$\begin{aligned}
\min \quad & \sum_{e\in E} c_e x_e + \sum_{v\in O_Q} D(v) z_v & & \text{(LC-P)}\\
\text{s.t.} \quad & \sum_{e\in\delta(S)} x_e + \sum_{v:S\in T(v)} z_v \ge 1 & \forall S \in \mathcal{T}_Q \quad & (8)\\
& \sum_{e\in\delta(S)} x_e + \sum_{v:S\in N(v)} z_v \ge 1 & \forall S \in \mathcal{N}_Q \quad & (9)\\
& x_e, z_v \ge 0 & \forall e \in E, \forall v \in O_Q. &
\end{aligned}$$

Given a feasible solution F to the underlying even parity connection game we would like to construct a solution (x, z) that is feasible for (LC-P). We will also show the associated duals $\{y_S\}_{S\in\mathcal{S}}$ returned by the algorithm are feasible for (LC-D). We would like to then show that the optimal solution OPT_{LC-P} is bounded by the cost of F. By weak duality, the optimal solution to the lifted-cut dual OPT_{LC-D} is at most OPT_{LC-P}, so this value too is bounded by the cost of F. By using the optimal solution F^*, we would finally show that the sum of the duals returned by our algorithm is bounded by the cost of the optimal solution, implying our cost-sharing method is competitive.

Lemma 3.1.1. *Any set of duals $\{y_S\}_{S\in\mathcal{S}}$ returned by the KW-3 algorithm is feasible to LC-D*

Proof. The duals returned by KW-3 naturally satisfy $\sum_{S\in\mathcal{S}:e\in\delta(S)} y_S \le c_e, \forall e \in E$, as whenever an edge $e \in E$ becomes tight between two moats we merge the moats into one. By only growing the duals corresponding to connected components of tight edges (moats), we ensure that we never overload an edge.

To prove the duals also satisfy $\sum_{S\in T(v)} y_S + \sum_{S\in N(v)} y_S \le D(v), \forall v \in O_Q$, we first note that at any given time $0 \le \tau \le \tau^*$, v is contained in at most one moat $S \in \mathcal{S}$ whose dual y_S is growing at time τ. To complete the proof, we just need to show that the dual corresponding to any set $S \in (T(v) \cup N(v))$ will not grow after time $\tau = D(v)$.

Suppose $S \in T(v)$, then either v is responsible for S or v has a mate $u \notin S$. Now suppose for the sake of contradiction that algorithm KW-3 grows the dual y_S at a time $\tau > D(v)$. Suppose v is responsible for S, then $D(v) \ge D(w), \forall w \in S$. But then at time $\tau > D(v)$ the set S does not contain any active vertices, which contradicts the assumption that KW-3 was growing the dual after time $\tau = D(v)$. Instead, suppose v has a mate $u \notin S$. By time $\tau = D(v)$ in the run of GW(i) (for each player $i \in Q$), by definition v and u must be in the same moat. As a result, v and u must also be in the same moat in the execution of KW-3 by time $\tau = D(v)$. Thus, either KW-3 grows the dual y_S at a time $\tau < D(v)$, or never at all. This also contradicts the assumption that KW-3 was growing the dual after time $\tau = D(v)$.

Now suppose $S \in N(v)$, so that either v is responsible for S or v is the best friend of u, the vertex responsible for S. Suppose for the sake of contradiction that algorithm KW-3 grows the dual y_S at a time $\tau > D(v)$. If v is responsible for S, then again $D(v) \geq D(w), \forall w \in S$. But at time $\tau > D(v)$ the set S does not contain any active vertices, which contradicts the assumption that KW-3 was growing the dual after time $\tau = D(v)$.

Instead, suppose v is the best friend of u, the vertex responsible for S. If v is also the mate of highest precedence order of u, then $D(v) \geq D(u)$. But this would again imply that the set S does not contain any active vertices. Suppose then that another vertex w is the mate of highest precedence order of u. This implies $D(v) \leq D(u)$. Let player i be the player for whom the run of GW(i) established v and w as mates. Since v is the best friend of u, we know $w \notin S$ and that $v \in O_i$. Now consider the mates of v. If any mate of v is not in S, by the same reasoning as above KW-3 would not grow the dual y_S after time $\tau = D(v)$.

Instead, suppose that all of v's mates are in S. Since S is a non-T-Cut, there are an even number of vertices of O_i in the set S. We know that v and u were not established as mates in the run of GW(i), so it must be v's mate from the run of GW(i) is a third vertex x owned by player i in the set S. This immediately implies there is a fourth vertex y owned by player i in S. Since v is u's best friend, we know $v \succ x$ and $v \succ y$. But the vertex y also has a mate from the run of GW(i) that can't be any of v, u or x. If this mate is not in S, then again KW-3 would not grow the dual y_S after time $\tau = D(y) \leq D(v)$. This establishes a recursive argument which will continue until we exhaust the vertices of O_i. In summary, if v is the best friend of u but not its mate of highest precedence order and all of v's mates are in S, then either S is actually a T-Cut, or KW-3 would never grow the dual y_S anyway. □

3.1.2 Competitive Conjectures

We conjecture that there always exists a constructed solution to the lifted cut primal whose objective value is at most the cost of the optimal solution to the underlying network design problem. In an attempt to prove this powerful result, we tried to patch up the given feasible solution by buying some edges. By creating a forest where every vertex is in the same component as its' mates, it would be easy to construct a solution that is feasible to the lifted cut primal and whose objective value is at most the cost of the augmented solution.

We conjecture that the cost of these edges required to augment any feasible solution to the even parity connection game is at most a constant factor times the cost of the optimal solution, which implies the total cost shares would be at most a constant factor times the optimal solution. For a constant α, this property is known as α-approximate competitiveness (as in [5]).

3.2 Conclusions

Our main result is a 2-budget balanced and cross-monotonic cost sharing method for the downwards monotone set cover game, which arises naturally from any downwards monotone $0,1$-function. We have also designed a 2-budget balanced and cross-monotonic cost sharing method for two versions of the edge cover game arising from the edge cover problem. These games are special cases of the downwards monotone set cover game. By a result by Immorlica, Mahdian & Mirrokni in [4] our result is best possible for the edge cover game.

We also designed a cross-monotonic cost sharing method for a network design game we call the Even Parity Connection game arising from the T-Join problem that generalizes proper cut requirement functions. We have shown our algorithm returns cost shares that recover at least half the cost of the solution. We conjecture that our cost sharing method for the even parity connection game is competitive and thus 2-budget balanced.

Bibliography

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. (Preliminary version in *23rd STOC*, 1991).

[2] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *J. Comput. System Sci.*, 63(1):21–41, 2001. Special issue on Internet algorithms.

[3] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. (Preliminary version in *5th SODA*, 1994).

[4] N. Immorlica, M. Mahdian, and V. S. Mirrokni. Limitations of cross-monotonic cost sharing schemes. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 602–611. ACM Press, 2005.

[5] K. Jain and V. Vazirani. Applications of approximation algorithms to cooperative games. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 364–372, 2001.

[6] J. Könemann, S. Leonardi, and G. Schäfer. A group-strategyproof mechanism for Steiner forests. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 612–619. ACM Press, 2005.

[7] J. Könemann, S. Leonardi, G. Schäfer, and S. van Zwam. From primal-dual to cost shares and back: a stronger LP relaxation for the Steiner forest problem. In *Automata, languages and programming*, volume 3580 of *Lecture Notes in Comput. Sci.*, pages 930–942. Springer, Berlin, 2005.

[8] H. Moulin. Incremental cost sharing: Characterization by coalition strategyproofness. *Social Choice and Welfare*, 16:279–320, 1999.

[9] H. Moulin and S. Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Econom. Theory*, 18(3):511–533, 2001.

[10] T. Roughgarden and M. Sundararajan. New trade-offs in cost-sharing mechanisms. In *STOC*, 2006.