Group Strategyproof Mechanisms via Primal-Dual Algorithms

Martin Pál * Éva Tardos †

April 7, 2003

Abstract

We develop a general method for cost-sharing that is approximately budget balanced and group strategyproof. We use our method to design cost sharing mechanisms for two NP-complete problems: metric facility location, and single source rent-or-buy network design. Both mechanisms are competitive, group strategyproof and recover a constant fraction of the cost. For the facility location game our cost-sharing method recovers a $1/3$rd of the total cost, while in the network design game the cost shares pay for a $1/15$ fraction of the cost of the solution.

*Computer Science Department, Cornell University, Ithaca, NY 14853. Email: mpal@cs.cornell.edu. Research supported in part by ONR grant N00014-98-1-0589.

†Computer Science Department, Cornell University, Ithaca, NY 14853. Email: eva@cs.cornell.edu. Research supported in part by ONR grant N00014-98-1-0589.

1 Introduction

Large networks, such as the Internet, are developed, built, and operated by cooperation of a large number of Autonomous Systems, all of whom act selfishly. These independent "agents" are willing to work together when such cooperation helps them achieve their goals. In this paper we study the *cost sharing* problem, when a group of agents agrees to install or maintain a jointly used facility, such as a shared network, and split the cost among themselves. We are interested in means of sharing the cost that would motivate the agents to cooperate by revealing their true value for the network, and such that no agent (or group of agents) would have incentives for an adversarial action against other players or against a service provider who, in essence, organizes the cooperation.

For any given set J of users (assumed to be a subset of a universe U of potential users), there is a minimum cost network that satisfies all users in J, and we denote the cost of such a network by $c^*(J)$. We assume that each potential user $i \in U$ has a private *utility* u_i of being connected to the network. The user is not willing to pay more than her utility for the benefit of being a member of the network, and if she is asked to pay a share that is higher than her utility, she prefers to stay disconnected.

The question we are trying to answer is how to agree on a set $J \subseteq U$ of users that will be part of the network, and how to split the cost so that all players would be content with the decision. In particular we want that no agent has an incentive to change her mind about being or not being in the network later, have incentives to lie about their utility, and that no group of agents is tempted to build a separate network of cost smaller than the sum of their current contributions.

Our proposed answer is to use cross-monotonic cost sharing methods. A *cost sharing method* is an algorithm to compute, given a set J of participating users, the cost share $\xi(J, j)$ of each user $j \in J$. A cost sharing method is *cross-monotonic* (also known as population monotonic), if the cost share of each user never goes up as the set of participating users increases, that is, $\xi(J, j) \geq \xi(J', j)$ for all $J \subseteq J'$. Cross-monotonicity is a crucial property that provides incentives for cooperation: indeed, no user j is not motivated to discourage other users from joining the network, as her cost share might only increase, but may be able to decrease her cost share by encouraging other users to join. Moulin and Shenker [29] show that cross-monotonic cost sharing leads to group strategyproof mechanisms for determining the set of agents that should be connected by the network.

Other desirable properties of a good cost sharing method include *competitiveness* and *cost recovery*. The former property states that the method is not allowed to charge the users more than the true cost, i.e. $\sum_{j \in J} \xi(J, j) \leq c^*(J)$, while the latter requires that the users do share the cost: $\sum_{j \in J} \xi(J, j) \geq c^*(J)$.

Unfortunately for many games of interest, cross-monotonic competitive and cost recovering cost sharing methods do not exist. It is well known that such methods provide cost allocations that lie in the *core*, a well studied concept in the game theory literature [28], and that the core is empty for these games. Motivated by the research effort on *approximate core* (e.g., [12, 7, 14]), we relax the last property to *α-approximate cost recovery*: $\sum_{j \in J} \xi J, j \geq c^*(J)/\alpha$, that is, the users are only required to recover an $1/\alpha$ fraction of the cost.[1]

Our Results. We give a general technique for using primal-dual approximation algorithms to generate cross-monotonic, competitive and approximately cost recovering cost shares. The cost shares are closely related to the dual solution generated by the algorithm, and hence the competitiveness and cost recovery properties are a consequence of dual feasibility and the approximation guarantee of the algorithm. However, cost-sharing directly based on linear programming duals does not usually deal well with individual utilities: the resulting cost-sharing is not cross-monotonic, and hence does not lead to group strategyproof

[1] Our choice to relax cost recovery was arbitrary; we could as well relax competitiveness. The only difference is that all our cost sharing methods would be multiplied by the factor α.

mechanisms. Our technique changes the primal-dual approximation algorithm by striving to make the cost-sharing (the dual growing process) as smooth as possible, avoiding sudden arbitrary jumps and changes in the growth that could harm cross-monotonicity, even at the cost of somewhat weakening the approximation ratio.

We apply our technique to two network design games. First, we illustrate it by giving a cross-monotonic cost sharing method for the *Facility Location* game, and then proceed to the main result of the paper, which is a cross-monotonic cost sharing method for *Single Source Rent-or-Buy Network Design* (also known as a simple Access Network Design game). Both cost sharing methods are competitive, while the former recovers $1/3$ and the latter $1/15$ of the cost of the optimal solution. The bounds on cost recovery are shown by explicitly constructing a feasible solution to the respective problems.

The games. The *facility location* problem is given by a set of potential servers (facilities) F, and a set $J \subseteq U$ of users. The problem is to open a subset of the facilities, paying amount f_p for each open facility p, and build a link from each user $j \in J$ to some open facility, given that the cost for connecting a client j to facility p is c_{jp}. We make the common assumption that c is a metric on $F \cup J$ (more precisely, on $F \cup U$). Let $c^*_{\mathrm{FL}}(J)$ denote the cost of the minimum cost solution for the set of users J.

In the *single source rent or buy* problem, we are given a set J of users and a source s, residing in a metric space defined by a graph G, and a number M. The goal is to build a tree that connects all users to the source, so that each edge e of the tree can either be *bought* at cost Mc_e, or *rented* at cost c_e times the number of users that use the edge on their way to the sink. Similarly, let $c^*_{\mathrm{RoB}}(J)$ be the cost of the optimum solution.

In the corresponding games users are trying to decide on the set J of users who should participate in the network, so that the participants can (approximately) share the cost.

Related work. There is a large body of literature on cooperative games, mostly focused on fair cost allocation (see e.g. [28]). A cost allocation is a vector x assigning each user $i \in U$ her share. It is *fair*, if no subset of users I is charged more than the optimum cost, i.e. $\sum_{j \in I} x(j) \leq c^*(I)$ for every $I \subseteq U$. If it also recovers or approximately recovers the cost, it is said to be in the *core* or *approximate core*. It is important to note that a cross-monotonic and competitive cost sharing method automatically gives us fair allocations for all $J \subseteq U$:

$$\sum_{j \in I} \xi(J, j) \leq \sum_{j \in I} \xi(I, j) \leq c(I).$$

The connection between the linear programming dual and cost-sharing is well known, and has been used to build core and approximate core allocations for various problems, including facility location [7, 14] and TSP games [12]. In fact, the classic Bondareva-Shapley theorem [6, 31] implies that for so called covering games (in which the cost $c^*(\cdot)$ is given by the minimum cost solution to a covering integer program), the core is nonempty if and only if the linear relaxation of the game-defining IP has no integrality gap.

Moulin and Shenker [29] show that cross-monotonic cost-sharing leads to group strategyproof mechanisms. They study the following simple mechanism for selecting the set of users J who should participate in the network: Start with assuming that $J = U$. Repeatedly offer every user j who remains in J membership in the network at price $\xi(J, j)$, and keep removing users who decline at the price offered. Stop when all remaining users in J accept the offer. Assuming that ξ is a cross-monotonic cost sharing method, they prove that this mechanism induces users to play truthfully, that is, accept whenever the price offered is less than or equal to u_j and decline otherwise. They prove that this mechanism is not only *strategyproof*, but also *group strategyproof*, that is, no user or coalition of users can benefit by deviating from the truthful strategy.

Moulin and Shenker [29] show that the classical Shapley value is in fact a cross-monotonic cost-sharing method whenever the cost function $c^*(\cdot)$ is a submodular function of the set J (see also [22]). The optimum solution to facility location, Rent-or-Buy network design, or their LP relaxations is in general not submodular.

The only known cross-monotonic cost-sharing for a non-submodular cost-function is the cost-sharing of Kent and Skorin-Kapov [24] and Jain and Vazirani [21] for the spanning tree game. The spanning tree heuristic gives a 2-approximation for the Steiner tree problem, hence the spanning tree cost-sharing method is also a 2-approximate budget-balanced cross-monotonic cost-sharing for the Steiner tree game, which is a special case of the above Network Design problem with $M = 1$.

Jain and Vazirani [21] pose the question of designing cross-monotonic cost sharing for other combinatorial games. Recently, Devanur, Mihail and Vazirani [8] used a standard primal-dual algorithm to provide a strategyproof cost sharing mechanism for facility location. However, the resulting cost-sharing is *not* cross-monotonic, and their mechanism is only individually strategyproof, not group-strategyproof. In contrast, our technique modifies the algorithm to make the cost shares cross-monotonic (and hence the resulting mechanism group strategyproof).

Much of the previous work on network design games [1, 4, 11, 19] focuses on the Nash equilibria of the corresponding non-cooperative games. In its nature, Nash equilibrium is a non-cooperative concept: the agents simply observe each others behavior, and respond accordingly. Typically Nash solutions break down if we allow the agents to cooperate. In contrast, our aim is to design a game that allows the agents to follow a very simple truthful strategy, and such that the final solution adopted by the agents is stable, meaning that no group of agents has interest to change it.

From the plethora of approximation algorithms for facility location we mention only the primal-dual 3-approximation algorithm of Jain and Vazirani [20], and the currently best, 1.52-approximation algorithm of Mahdian et al. [26].

The first constant factor approximation algorithm for the Single Source Rent-or-Buy Network Design problem is due to Karger and Minkoff [23] and Guha, Meyerson and Munagala [16] (see also [17]). Both algorithms use a variant of facility location to gather the clients, and then build a Steiner tree on the set of gathering points. Swamy and Kumar [32] gave a primal-dual 4.55-approximation algorithm, that is also built analogously as a 2-stage algorithm (gathering followed by a Steiner tree). Recently, Gupta et al. [18] gave a simple randomized 3.55-approximation algorithm.

Our Techniques. Any cost sharing method ξ can be made cross-monotonic by replacing it by a new function $\xi'(J, j) = \min_{I \subseteq J} \xi(I, j)$. It is not hard to see that $\xi'(J, j)$ is population monotonic. However, there are two obvious problems with this idea. First, computing the new cost-shares for a given set of users J requires considering all subsets $I \subseteq J$, and hence may take exponential time. Second, the new cost-shares $\xi'(J, j)$ may be much smaller than the original shares $\xi(J, j)$, and hence may not be able to recover a significant fraction of the budget.

In this paper we develop a general technique to use primal-dual algorithms for generating cross-monotonic cost-shares. Our algorithms are simple dual-growing processes that guarantee population monotonicity in a way analogous to the cross-monotone cost-shares $\xi'(J, j)$ above. Known primal-dual approximation algorithms all are based on a "fair" way of raising cost-shares: all users increase their cost-contributions uniformly until they are satisfied (the cost contributed so far pay for a facility they can connect to). The fact that these primal-dual algorithms raise all players cost-shares evenly suggests that they may be useful for building fair cost-shares. However, the resulting cost-shares are *not* population monotonic: adding new users close to an existing agent i, makes it easier for agent i to be satisfied, and once agent i stops raising his share, all remaining agents can be negatively effected.

The idea that allows us to compute monotone cost-shares is to have a "ghost" share for every user. Although in the original algorithm we are forced to stop raising the user's contribution in order not to overcharge some set of users, we still raise the ghost's share until all users are satisfied. For example in the facility location problem, considered in Section 2, the extra contribution from the ghosts results in opening more facilities and users getting connected earlier than in the original algorithm. This idea trivially ensures cross-monotonicity, as adding more users (and their ghosts) can only cause more facilities to be open, and each individual user satisfied earlier. The main challenge is to show that even that most facilities are only virtually paid for by the ghosts, the real cost shares still pay enough for constructing a feasible solution.

In Section 3 we consider the Rent or Buy network design game. Most known approximation algorithms for this problem build the solution in a two phase process: first they use a facility location algorithm to gather (or cluster) users into larger groups, and then build a Steiner-tree on the set of gathering points. It would be natural to suggests to combine the cross-monotonic cost-sharing for the facility location problem of Section 2 with the spanning tree cost-sharing of Kent and Skorin-Kapov [24] and Jain and Vazirani [21]. However, combining cost-sharing for such a two phase process does not lead to cross-monotonic cost-shares: decisions made in the gathering stage affect the way players contribute towards building the tree in an unpredictable way, making it impossible to guarantee cross-monotonicity of the second stage.

In Section 3 we extend the idea of ghosts to the Rent or Buy network design game. We use the idea of a two stage process, gathering demand followed by building a Steiner tree on the gathering points. To overcome the difficulty making a two-phase process cross-monotonic, we run the two processes simultaneously, adding all potential gathering points into the second stage as soon as they are created by the first stage. Again, cross-monotonicity is easy: the more users we have, the more gathering points get opened, and earlier. More gathering points means that also the tree gets built faster, only reducing the cost share of each individual user.

As in the facility location case, the main challenge is to prove that the cost shares can pay for a feasible solution. Note that since the first phase may open much more gathering points than necessary (in fact, every point reachable by enough users eventially becomes a gathering point), the cost of building a Steiner tree on all of them would be prohibitively large. We thus need to select only a suitable subset of the points, and relate the processes that builds a tree on the selected points to the process on all potential points to show that the cost shares generated by the latter process can pay for the tree built by the former.

2 The Facility Location Game

In this section we develop our cross-monotonic cost-sharing method for the metric facility location problem. The facility location problem gives rise to very simple and elegant primal-dual algorithms, and hence it provides a clean example to illustrate how our "ghost" method turns a primal-dual algorithm into a cross-monotonic cost-sharing method. In the next section we use these ideas to develop a cost-sharing method for a more complex network design problem.

The *Metric Facility Location game* is given by a set F of possible *facilities*, a universe U of users and a metric $c(\cdot, \cdot)$ on $F \cup U$. Each facility $p \in F$ has *opening cost* f_p. Given a subset $J \subseteq U$ of users, the problem is to open a subset Q of facilities, and connect each user in J to an open facility. The cost of a solution for a given set of users J is the sum of the costs of facilities we open plus the sum of the costs for connecting clients to facilities. The cost of connecting user j is equal to her distance from the closest open facility. Formally, the cost $c(J)$ of a solution for the set of users J is

$$\sum_{q \in Q} f_q + \sum_{j \in J} \min_{q \in Q} c(j, q),$$

where Q is the set of facilities we open. The players in this game are the users, and our goal is to construct a solution and make each user pay certain amount to recover a constant fraction of the cost of the solution constructed.

Our main contribution for the facility location game is to build cross-monotonic cost-shares that recover a constant fraction of the cost of the optimal solution. To define the cost-shares we use a "ghost-process" that can be viewed as a smoothed version of a primal-dual approximation algorithm, like Jain and Vazirani [21]. We will use the algorithm by Mettu and Plaxton [27] for building the facilities. Note that the Mettu and Plaxton [27] is *not* a primal-dual algorithm, and hence does not come with cost-shares. As a byproduct, our analysis shows that the Mettu and Plaxton [27] algorithm can be interpreted as a primal-dual algorithm. Archer, Rajagopalan, and Shmoys [2] have also independently given a primal-dual analysis of this algorithm.

The ghost process. First we define a *ghost process*, and use it to define cost shares for each user. For each facility p we define a neighborhood S_p of clients, the smallest set of clients such that charging them all the same share $\alpha = t(p)$ each can afford to pay for the connection, and enough is left over to pay for opening the facility. We will not be able to charge all users in this set the amount α as they may have closer other facility that they would rather contribute to. But our plan is to make the users in the set S_p pay enough so that if we choose to open facility p we will be able to recover a constant fraction of the cost.

The *ghost process* grows a ghost for every user. The ghost of user j is a ball with j as its center that is growing uniformly to infinity. The radius of the ghost at any time $t \geq 0$ is equal to t. The ghost of a user j *touches* a facility p at time t if $c(j,p) \leq t$. All ghosts that touch a facility start *filling* the facility. The contribution of the ghost j at time t towards filling facility p is $\kappa_{jp} = \max\{0, t - c(j,p)\}$. A facility p is declared *full*, if $\sum_{j \in J} \kappa_{jp} \geq f_p$. Let $t(p)$ denote the time when p becomes full. Let S_p be the set of users that contributed towards filling p, i.e. all users that are at distance less than $t(p)$ from p. Note that charging all users in S_p the amount $\alpha = t(p)$ we would recover exactly the cost of opening facility p and the cost of connecting all users in S_p to p, as $f_p = \sum_{j \in S_p} (t(p) - c(j,p))$, by definition.

The cost shares. We raise the cost share α_j of each user j at unit rate until j's ghost touches some full facility or some facility it is touching becomes full. More formally,

$$\alpha_j = \min\left(\min_{p: j \in S_p} t(p), \min_{p: j \notin S_p} c(j,p)\right).$$

Note that by adding more users, each facility can only get filled more quickly, and hence each individual user can only become satisfied earlier. Hence

Lemma 2.1 *The cost shares generated by the above process are cross-monotonic.*

Deciding which facilities to open. We use a simple rule, that can be viewed as a restatement of the facility location algorithm due to Mettu and Plaxton [27] to build a feasible. We first decide which facilities to open, and then assign each user to the closest open facility. For each facility p we decide whether we open it or not at the time p gets filled. We open the facility p if and only if there is no already open facility q such that $c(p,q) \leq 2t(p)$. This will keep open facilities sufficiently far away from each other.

The analysis Mettu and Plaxton [27] showed that the cost of the solution their algorithm obtains is at most 3 times the optimum cost $c^*(J)$. They did not consider their algorithm being primal-dual, and their analysis does not relate the cost of the solution with the dual LP. We provide a stronger guarantee: the cost shares do correspond to a feasible dual LP solution, and they recover 1/3 of the solution constructed.

First note that the cost-shares are competitive: It is easy to see that the sum of the cost-shares does not exceed the cost of any solution, as the set S_p of clients assigned to a facility p in a optimal solution never pay more than the cost of serving S and opeining p.

Theorem 2.2 (Competitiveness) *Every solution to the facility location problem has cost at least* $\sum_{j \in J} \alpha_j$.

Next we will relate the cost-shares to the cost of the Mettu-Plaxton solution. Recall that for each facility p we defined the sets S_p of closest clients who would fully pay for the facility (if they did not have other options). Consider two open facilities p and q, and let p be opened later than q. Since p was not closed, it must be far enough from facility q: by definition $c(p,q) \geq 2t(p) \geq t(p) + t(q)$, and hence

Fact 2.3 *If facilities p and q are both open, then S_p and S_q are disjoint.*

This means that every user contributes towards filling of at most one open facility. For the purposes of the analysis, we assign all users in the set S_p to the facility p, for every open facility p. We assign each user that does not belong to the set S_p of any open facility p to its closest open facility.

We claim that the users in the set S_p of every open facility p have paid enough to cover $1/3$ of the cost of p as well as $1/3$ of their connection costs. By definition of S_p, $f_p = \sum_{j \in S_p}(t(p) - c(j,p))$. After rearranging, we obtain that the opening cost f_p plus the connection cost $\sum_{j \in S_p} c(j,p)$ is equal to $|S_p|t(p)$. To prove our claim it suffices to show that the cost share of each user is at least $t(p)/3$.

Lemma 2.4 *Let p be an open facility, and let $j \in S_p$ be a user. Then $3\alpha_j \geq t(p)$.*

Proof. Consider a user $j \in S_p$. For contradiction, assume that $\alpha_j < t(p)/3$. Let q be the first full facility that j touched, i.e. $c(j,q) \leq \alpha_j$ and $t(q) \leq \alpha_j$. If q is open, we get a contradiction, since $c(p,q) \leq t(p) + \alpha_j < 2t(p)$. If q is not open, there is an open facility q' such that $c(q,q') \leq 2t(q) \leq 2\alpha_j$. See Figure 1. Again, we get a contradiction, because $c(p,q') \leq c(p,j) + c(j,q) + c(q,q') \leq t(p) + \alpha_j + 2t(q) \leq t(p) + 3\alpha_j < 2t(p)$. Note that $p \neq q'$, since $t(q') \leq \alpha < t(p)$. ∎

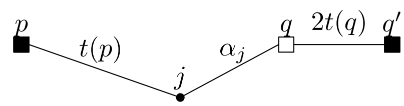

Figure 1: If $\alpha_j < t(p)/3$, then $c(p,q') < 2t(p)$.

Theorem 2.5 *The cost of the solution constructed is at most $3\sum_{j \in J} \alpha_j$, and hence above algorithm is a 3-approximate cross-monotonic cost sharing method for metric facility location.*

Proof. Lemma 2.4 shows that the clients in the sets S_q pay for a 1/3rd of their connection costs and the cost of the open facilities. In a similar fashion we can also show that the cost shares of users that do not belong to any S_p for an open facility p can pay for $1/3$ of their connection costs. ∎

3 Single Source Rent-or-Buy

The *Single Source Rent-or-Buy Network Design game* is given by an undirected graph $G = (V,E)$ with costs c_e on edges, a designated *source* vertex s, and a parameter M. We are given a set J of *users*, each user j being located at some vertex $v \in V$ (we allow more users to be located at the same node). If no confusion arises, we use j instead of its location v.

The goal is to build a network that connects each user to the source s by a path. Each edge e in the network can either be *bought* at cost Mc_e, or rented at cost c_e. A bought link can support any number of paths, while on a rental link we must pay the cost c_e for each path that uses the edge. A network is feasible, if it can support a path from each user j to the source s. The cost of the network is the sum costs of all bought and rented edges. Let $c^*(J)$ denote the cost of the optimal network supporting the set of users J.

In this section we present our main result, a cost sharing method for the rent or buy game.

Theorem 3.1 *There is a cross-monotonic cost sharing method for the single source rent or buy game that is competitive and recovers at least* $1/15$ *of the cost of a feasible network that we can find in polynomial time.*

It is not hard to see that the bought edges in the optimum rent or buy network must form a Steiner tree with s at its root, and that the rental edges form a shortest path connection from each user J to the closest point in the tree. Several approximation algorithms [23, 16, 32] for this problem exploit the optimal structure to construct a solution in two phases: first they use a facility location-like algorithm to gather the users in a few *center* locations, and rent a path from each user to such a center. When enough demand is gathered at each center, a Steiner tree is built, connecting all centers to the source.

It would be tempting to use our scheme for facility location to share the rental costs, and use the cost sharing method of Jain and Vazirani [21] to share the cost of buying the tree among the centers. Naturally, the contribution of each center would be split among the users connected to it. However, the gathering algorithm has to make decisions on where to choose the centers and how to assign users to them. These decisions (necessarily influenced by addition of new users) may have arbitrary effects on what center a particular user is assigned to and the number of other users it is sharing the center with, that could make the share of an individual user heavily non-crossmonotonic.

We solve this problem by giving a process that avoids making decisions altogether. As in the facility location case, we grow a ball around every user. Since there is no opening cost, we open a center at any location p that has been reached by M or more user balls. A regular primal-dual algorithm, such as [32] would freeze all users that got connected to the center, and stop growing their balls.

To ensure cross-monotonicity, we use the same trick as in the previous section: continue growing a ghost ball around every user, even after the user gets connected. Note that since the ghost balls keep contributing to opening new centers, much more centers are opened than necessary. This poses a new problem: each user may be connected to multiple centers, and the cost of building a Steiner tree on all centers may be prohibitively large.

We deal with this problem in two different ways. When computing the cost shares, we pretend that we are building a Steiner tree on all centers. We split the cost of each center and try to charge it equally to all users connected to the center. Now, since a user may be connected to multiple centers, she gets to choose which center to contribute to (we assume each user wants to contribute only to the center where the required contribution is smallest).

Since most centers are not contributed to by enough users (or any users at all), we can not hope that the user contributions would recover a constant fraction of the cost of a Steiner tree on all centers. To build a tree that the cost shares can pay for, we select only a subset of the centers, and assign each user to only one center. Now we have forced the user to contribute only to the center he was assigned to, but we still can only ask a contribution that is equal to the minimum of his potential contributions to any possible center.

Technically, the hard part is to relate the two processes: one that pretends to build a Steiner tree on all centers, used to compute the cost shares, and a standard primal-dual process that we use to build a real tree on a subset of the centers, and show that the cost shares grown by the first process pay for the solution constructed by the second.

After a simplifying assumption, we describe the heart of the algorithm, which is the ghost growing process. Then we describe how to generate the cost shares, and how to find a feasible solution that can be partially paid for by the cost shares. Finally, we show competitiveness by giving a lower bound on the cost of any feasible solution.

A Simplifying Assumption. It will be convenient to think of each edge e as a line segment of length c_e, containing a continuum of points. We use the term *location* to refer to both original vertices and intermediate points. We extend the shortest path metric c to a metric on locations in the obvious way. For the rest of this section, we will assume that *an edge segment can be bought or rented between arbitrary pair of locations*.

As shown in [32], a solution that is allowed to buy and rent edge segments arbitrarily can be transformed into one that does not use intermediate points of no greater cost. This is because any intermediate point p on edge $e = (uv)$ that is an endpoint of some paths can be shifted towards u or towards v without increasing the cost. We move p until it meets another intermediate endpoint or until it reaches one of the ends of the edge. This way, we can remove all intermediate endpoints one by one.

3.1 The ghost process

The ghost process we consider here is similar to the process considered in Section 2. Hence, each user j has a ghost, which is a ball $B(j,t)$ with j as its center and radius equal to the current time t.

The difference is that instead of thinking of the ghosts as contributing towards opening a facility, we will be looking for locations that are potential gathering points. Good candidates are locations that have been reached by M or more ghosts. Let us denote the (time varying) set of all such locations by $\mathcal{C}$. Note that at any time, the set $\mathcal{C}$ can be represented as a collection of vertices, edges and segments of edges.[2]

A *connected component* of $\mathcal{C}$ is any inclusion-maximal subset C of $\mathcal{C}$ such that any two locations $p_1, p_2 \in C$ are joined by a path (that is, a continuous curve) lying completely within C.

Let us take a closer look at the behavior of the set $\mathcal{C}$ in time. In the beginning, $\mathcal{C}$ is empty, and there are no components. As time progresses, components begin to appear. Each component starts as a single location p, that has been first reached by some set S of users, $|S| \geq M$. With increasing time, the set of locations reachable by the users in S locally looks like a uniformly growing ball with center p (note that the set $\bigcap_{j\in S} B(j,t)$ of locations reachable by all users in S may look as a union of several balls, as the same set S may initiate multiple components of $\mathcal{C}$). As these balls grow, two or more of them may eventually touch, joining into a single component. In general, pairs of components will be connecting up and merging. Hence at any time, each component C of $\mathcal{C}$ can be expressed as an union of uniformly growing balls of different radii, each of which started as an independent component at a single location p.[3] Note that since the ghosts grow indefinitely, eventually the set $\mathcal{C}$ will consist of a single component.

We would like to point out a correspondence between the way the set $\mathcal{C}$ grows and the standard primal-dual algorithm for Steiner tree (see [3], and also [15]), that will be useful later. The algorithm maintains a list of components. Initially, each terminal vertex is in a separate component. It grows a ball uniformly around every vertex, and whenever the balls of two vertices touch, it builds the edge joining the two vertices, merging the two components. The only difference is that while the Steiner tree algorithm starts with a fixed set of components, in the ghost growing process components may get created at arbitrary points in time. We will make use of this connection in Section 3.4 to show that the cost shares can pay for a constant fraction of the buying cost in a solution we construct.

3.2 Cost shares

To specify how the cost shares are computed, we need some definitions. We say that a user j at time t is *connected* to a component C of $\mathcal{C}$, if $B(j,t) \cup C \neq \emptyset$. The user is *satisfied* if it is connected to a component containing the source vertex s. For each user j, let $t(j)$ denote the time when the user first becomes connected to a component and let $t'(j)$ the time when j becomes satisfied.

Roughly speaking, we should make each user pay for the rental costs associated with him. This suggests that we should charge each user amount proportional to the time the user spent unconnected. Further, each user connected to a component C should contribute an equal share to the cost of growing the component. If

[2] It is easy to see that each edge is subdivided into at most $2\min\{M, |J|\} + 1$ segments, and hence this representation is of polynomial size.

[3] At most $\min\{M, |J|\}$ separate components can be created on any edge e, hence the number of balls in the union is polynomial.

a user is connected to multiple components, we let him contribute only to the component where his share is smallest.

For each component C of $\mathcal{C}$, let $J(C)$ denote the set of users connected to C. For a connected user j, let $a_j(t)$ be the maximum size $|J(C)|$ over all components C that j is connected to at time t. Let us define a function f_j for each user j as follows: $f_j(t) = 1$ for $0 \leq t < t(j)$, $f_j(t) = M/a_j(t)$ for $t(j) \leq t < t'(j)$, and $f_j(t) = 0$ for $t > t'(j)$. We define two versions of cost shares α_j and α'_j for each user j. The final cost share, determined in Theorem 3.8, will be a weighted sum of the shares we now define, and balanced to optimize the fraction of the cost recovered.

$$\alpha_j = \int_0^{t'(j)} f_j(t)dt \quad \text{and} \quad \alpha'_j = \int_0^{t(j)} f_j(t)dt = t(j).$$

It is easy to establish the first of the three desired properties of our cost sharing method.

Theorem 3.2 *The cost shares α and α' are cross-monotonic functions of the set J.*

Proof sketch. By adding more users, we can only make the set $\mathcal{C}$ larger, and hence make each user connect earlier. Moreover, each component of $\mathcal{C}$ can only increase by adding more users, as increases the number of users connected to that component, hence the cost shares can only grow slower. ■

3.3 Building a solution

The goal of the following two sections is to prove that the cost shares α and α' can recover a constant fraction of the cost of a network supporting the given set J of users. In this section, we give a way to construct such a feasible network, and in the next section we use the similarity between the behavior of the set $\mathcal{C}$ and the primal-dual Steiner tree algorithm to compare the cost of the solution constructed with the cost shares.

Remember from the discussion in Section 3.1 that each new component of the set $\mathcal{C}$ started out as a single point p. We will call such starting points *centers*. For a location p, let $t(p)$ denote the first time p appeared in the set $\mathcal{C}$. Note that $\mathcal{C}$ at a given time t is just a union of balls $B(p, t - t(p))$, with a ball of radius $t - t(p)$ around each center p.

We want to use the centers as gathering points. However, to be able to pay for the Steiner tree on the centers, we must select only a subset of the centers to open, and assign each user to contribute to at most one open center. We do the selection in a similar way we did for facility location.

For each center p we decide whether to open it at the time the center is created. We declare p open, if at time $t(p)$, there is no open center within radius $2t(p)$ of p. As in the facility location case, this ensures that no user j can be in the set $S_p = \{j \in J \mid c(j,p) \leq t(p)\}$ of users contributing towards creating p for two distinct open centers p. The following result carries over from the previous section.

Lemma 3.3 *Let p be an open center, and let $j \in S_p$. Then $3\alpha'_j \geq t(p)$. For a j that does not belong to any S_p, there is an open ghost center p such that $c(j,p) \leq 3\alpha'_j$.*

The network we construct is thus a follows: we buy a Steiner tree that connects opened centers to the source s, and rent a path from each user to its closest open center. Lemma 3.3 shows that the cost shares α' can pay for (1/3 of) the rental cost of the network. The next section is devoted to showing that the shares α can pay for a constant fraction of the tree we build.

3.4 Bounding the tree cost

For the purposes of analysis we assume that the primal-dual algorithm of Agarwal et al. is used [3] (see also [15]) to build a Steiner tree on the open centers. The algorithm starts with each center in a separate component. It grows a ball uniformly around every center. Every time two balls touch, a check is performed if the centers of the balls are in the same component. If not, a shortest path between the ball centers is bought, merging the two components into one. In addition, the center p of the first ball to reach the source vertex s is allowed to buy a shortest path between p and s.

We can think of the algorithm as incurring cost of M per time unit for growing each component that does not contain the root vertex s. (Growing the component containing s is free.) That is, if $a(t)$ denotes the number of non-root components at time t, the total cost of growing the components is $M\int_0^\infty a(t)dt$, where the integral is over the entire execution of the algorithm. By standard arguments, the cost of the tree constructed is at most two times the growing cost.

For each Steiner component C' we define a (time varying) set $\mathrm{Contrib}(C')$ of users that will be responsible for maintaining a steady flow of funding at least $M/3$ per time unit from the time increments $f_j(t)$ of their cost shares. The contributor sets are disjoint.

Consider a user j that belongs to a contributor set of a Steiner component C'. In the ghost process, this user may be connected to several components of $\mathcal{C}$, and some of these components may have a large number of users connected to them. So even if the component C' is "small", that is, does not have many contributors, it may happen that most of its contributors all have a connection to a large $\mathcal{C}$-component, and hence their contributions $f_j(t)$ are not enough to support the growth of the component C' at time t. However, what we will be able to say is that later, namely at time $3t$, the component C' will have grown large enough to cover the area that the original "large" components occupied at time t and connect to all users that were connected to the "large" components at time t. These users together are able to pay enough for the growth of C' at time $3t$ with their growth rates $f_j(t)$ at time t.

Lemma 3.4 *Let j be a user such that at time t, $j \in \mathrm{Contrib}(C')$ for some Steiner component C'. Then $f_j(t/3) \geq M/|\mathrm{Contrib}(C')|$.*

The exact definition of the contributor sets, as well as the proof of this lemma appears in the Appendix. Using the lemma, we can bound the cost of the tree built.

Lemma 3.5 *The cost of the tree we buy is at most $6\sum_j \alpha_j$.*

Proof. as follows. At any time t, the users are able to collect $Ma(t)$ revenue from their contributions at time $t/3$. Hence

$$\int_0^\infty Ma(t)dt \leq \int_0^\infty \sum_j f_j(t/3)dt = 3\int_0^\infty \sum_j f_j(t)dt = 3\sum_{j\in J}\alpha_j.$$

Since $M\int_0^\infty a(t)dt$ can pay for half of the tree, the buying costs of our solution does not exceed $6\sum_j \alpha_j$. ∎

3.5 Cost Recovery and Competitiveness

Theorem 3.6 (Cost recovery) *The cost of the solution constructed is at most $\sum_j(3\alpha'_j + 6\alpha_j)$.*

Proof. Lemma 3.5 bounds the buying cost by $6\sum_j \alpha_j$, while by Lemma 3.3, the rental costs can be bounded by $3\sum_j \alpha'_j$. ∎

Theorem 3.7 (Competitiveness) *Every feasible solution to the single source rent or buy instance has cost at least* $\max\left(\frac{1}{2}\sum_{j\in J}\alpha_j,\ \sum_{j\in J}\alpha'_j\right)$.

One way to prove Theorem 3.7 is to write down an IP formulation of the single source rent or buy problem and give feasible dual solutions of cost $\sum_j \alpha_j/2$ and $\sum_j \alpha'_j$ respectively. In fact, the vectors $\alpha/2$ and α' can be completed to a feasible solution of the dual given by [32]. In the Appendix we give a direct proof of this theorem, which we believe is more intuitive.

Combining Lemma 3.2, Theorem 3.6 and Theorem 3.7 we obtain

Theorem 3.8 *The cost shares* $\xi(J,j) = 1/5\alpha'_j + 2/5\alpha_j$ *are cross-monotonic, competitive and recover* $1/15$ *fraction of the cost of the solution constructed.*

Acknowledgments

We would like to thank Chaitanya Swamy and David Shmoys for useful discussions and suggestions.

References

[1] E. Anshelevich, A. Dasgupta, É. Tardos, and T. Wexler. Near-Optimal Network Design with Selfish Agents. To appear in STOC'03.

[2] A. Archer, R. Rajagopalan, D. B. Shmoys. Lagrangian relaxation for the k-median problem: new insights and continuity properties. Manuscript.

[3] A. Agrawal, P. Klein, R. Ravi. When Trees Collide: An approximation algorithm for the generalized Steiner Tree problem on networks. *SIAM Journal on Computing*, 24(3):440-456, 1995.

[4] V. Bala and S. Goyal. A Non-Cooperative Model of Network Formation. In *Econometrica* 68(2000), 1181–1229.

[5] M. L. Balinski. On finding integer solutions to linear programs. *Proc. IBM Scientific Computing Symp. on Combinatorial Problems*, 225-248. IBM, 1966.

[6] O. N. Bondareva. Some applications of linear programming to cooperative games. Problemy Kibernetiki, 10:119–139, 1963

[7] P. Chardaire. On the core of facility location games. Unpublished manuscript available from `http://www.sys.uea.ac.uk/people/pc/`.

[8] Nikhil Devanur, Milena Mihail and Vijay V. Vazirani. Strategyproof cost-sharing Mechanisms for Set Cover and Facility Location Games. To appear In Proc. ACM Conference on Electronic Commerce, 2003.

[9] N. Devanur, C. Papadimitriou, A. Saberi, V. Vazirani. Market Equilibrium via a Primal-Dual-Type Algorithm. *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, 2002.

[10] J. Feigenbaum, C. H. Papadimitriou, S. Shenker. Sharing the cost of multicast transactions *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, 2000, pp 218–227.

[11] A. Fabrikant, A. Luthra, E. Maneva, S. Papadimitriou, and S. Shenker. On a Network Creation Game. Unpublished manuscript.

[12] U. Faigle, S.P. Fekete, W. Hochstättler, W. Kern. On approximately fair cost allocation for the Euclidean Traveling Salesman Problem. *OR Spektrum* 20, 1998, pp. 29-37.

[13] N. Garg, R. Khandekar, G. Konjevod, R. Ravi, F. S. Salman, A. Sinha. On the Integrality Gap of a Natural Formulation of the Single-Source Buy-at-Bulk Network Design Problem. *Lecture Notes in Computer Science*, vol. 2081, 2000.

[14] Michel X. Goemans and Martin Skutella. Cooperative Facility Location Games. *Proceedings of the 11th Symposium on Discrete Algorithms*, 2000, pp 76–85.

[15] Michel X. Goemans and David Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM Journal on Computing*, 24, 1995, pp 296–317.

[16] S. Guha, A. Meyerson, K. Munagala. Hierarchical Placement and Network Design Problems *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[17] S. Guha, A. Meyerson, K. Munagala. A Constant Factor Approximation for the Single Source Edge Installation Problem. 33rd Annual ACM Symposium on the Theory of Computing, (2001), pp 383–388.

[18] Anupam Gupta, Amit Kumar, Tim Roughgarden. Simpler and Better Approximation Algorithms for Network Design. To appear in STOC 2003.

[19] H. Heller and S. Sarangi. Nash Networks with Heterogeneous Agents. Working Paper Series 2001, E-2001-1, Virginia Tech.

[20] Kamal Jain, Vijay V. Vazirani. Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Problems. *Journal of the ACM*, 48:274–296, 2001.

[21] Kamal Jain, Vijay V. Vazirani. Applications of Approximation Algorithms to Cooperative Games. 33rd Annual ACM Symposium on the Theory of Computing, 2001, pp 364–372.

[22] Kamal Jain, Vijay V. Vazirani. Equitable Cost Allocation via Primal-Dual Type Algorithms. *34th Annual ACM Symposium on the Theory of Computing*, 2002.

[23] D. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[24] K. Kent, D. Skorin-Kapov. Population monotonic cost allocation on mst's. *Operational Research Proceedings KOI*, 1996, pp 43–48.

[25] Amit Kumar, Anupam Gupta, Tim Roughgarden. A Constant Factor Approximation Algorithm for the Multi-commodity Rent-or-Buy Problem. To appear in FOCS 2002.

[26] M. Mahdian, Y. Ye, and J. Zhang Improved Approximation Algorithms for Metric Facility Location Problems. APPROX 2002.

[27] R. R. Mettu, C. G. Plaxton. The online median problem. *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 339-348, 2000.

[28] H. Moulin. Cooperative Microeconomics: A Game-Theoretic Introduction. Princeton University Press. 1995

[29] H. Moulin, S. Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Economic Theory* 18, pp. 511-533, 2001.

[30] R. Ravi, F.S. Salman. Approximation Algorithms for the Traveling Purchaser Problem and its variants in network design. *European Symposium on Algorithms*, 1999.

[31] L.S. Shapley. On balanced sets and cores. Naval Research Logistics Quaterly, 14:453–460.

[32] Chaitanya Swamy, Amit Kumar. Primal-dual Algorithms for Connected Facility Location Problems. APPROX 2002.

A Appendix

In the appendix, we give proofs of Lemma 3.4 and Theorem 3.7.

The contributor sets, and proof of Lemma 3.4

First, let us define the contributor sets. For an open center p, let $S_p(t)$ denote the set of users that are within distance t from p. Note that $S_p = S_p(t(p))$. Let C' be a Steiner component at some given time t. We define

$$\mathrm{Contrib}(C') = \bigcup_{p \in C'} S_p(\max\{t, t(p)\}).$$

Lemma A.1 *At every time t, the contributor sets of two distinct components C'_1, C'_2 of H' are disjoint.*

Proof. Suppose $\mathrm{Contrib}(C_1') \cap \mathrm{Contrib}(C_2') \neq \emptyset$. Then there are centers $p \in C_1'$, $q \in C_2'$, such that $S_p(\max\{t, t(p)\}) \cap S_q(\max\{t, t(q)\}) \neq \emptyset$. Without loss of generality assume $t(p) \geq t(q)$. If $t \geq t(p) \geq t(q)$, then $c(p, q) \leq 2t$ and hence p and q must belong to the same component of H'. If $t \leq t(p)$, then $c(p, q) \leq t + t(p) \leq 2t(p)$, which means that we could not have declared p open in the first place. ■

The following facts essentially carry over from the facility location section.

Fact A.2 *For location $p \in \mathcal{C}$, there is an opened center q (possibly $q = p$) such that $c(p, q) \leq 2t(p)$.*

Fact A.3 *If centers p and q are both open, then S_p and S_q are disjoint.*

Lemma A.4 *Suppose at time t, users i and j are connected to the same component C of the set $\mathcal{C}$. Then at time $3t$, i and j belong to the contributor set of the same Steiner component C'.*

Proof. Let i and j be connected to a component C of $\mathcal{C}$ at some time t. Note that C is a union of balls around (opened or unopened) centers. User i is connected to C, hence there must be a center q with a ball $B(q, t - t(q))$ that intersects the ghost of i. Similarly there is a center q' and a ball $B(q', t - t(q'))$ intersecting the ghost of j. Since the centers q and q' belong to the same component, there must be a sequence of centers $q_1 = q, q_2, \ldots, q_k = q'$, with balls $B_\ell = B(q_\ell, t - t(q_\ell))$ such that the balls of every two consecutive centers intesect.

By Fact A.2, there is a sequence of (not necessarily distinct) open centers $p_1, \ldots, p_k$, such that for every ℓ, $c(p_\ell, q_\ell) \leq 2t(q_\ell)$. See Figure 2 for an example. It is easy to verify that the distance $c(i, p_1) \leq 2t(q_1) + t \leq 3t$, and analogously $c(p_k, j) \leq 3t$. The distance $c(p_\ell, p_{\ell+1})$ can be bounded by $4t$ as

$$c(p_\ell, p_{\ell+1}) \leq c(p_\ell, q_\ell) + c(q_\ell, q_{\ell+1}) + c(q_{\ell+1}, p_{\ell+1}) \leq 2t(q_\ell) + (t - t(q_\ell)) + (t - t(q_{\ell+1})) + 2t(q_{\ell+1}) \leq 4t.$$

Hence each pair of consecutive balls $B(p_\ell, 3t)$ must intersect, proving that p_1 and p_k are in the same Steiner component at time $3t$. By definition of Contrib, both i and j must be in the contributor set of this Steiner component. ■

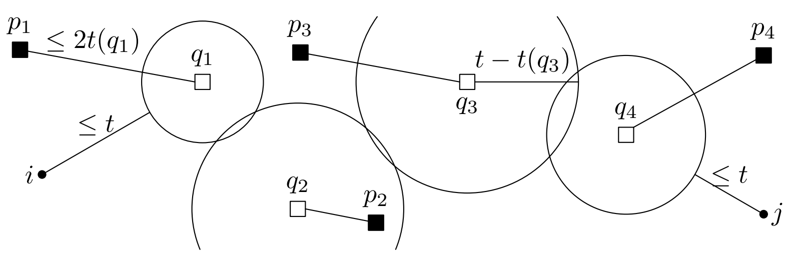

Figure 2: Clients i and j connected to a component C at time t.

Proof of Lemma 3.4. Note that since $|S_p| \geq M$ for any open center p, $|\mathrm{Contrib}(C')|$ is always at least M, and hence $M/|\mathrm{Contrib}(C')| \leq 1$. If j is not connected at time $t/3$, then $f_j(t/3) = 1$ and then the inequality trivially holds.

If j is connected at time $t/3$, $f_j(t/3) = M/|J(C)|$ for some component C of $\mathcal{C}$. By Lemma A.4, $\mathrm{Contrib}(C')$ at time t contains all users that were contributing to C at time $t/3$. ■

Proof of Theorem 3.7

Proof of Theorem 3.7. Consider an arbitrary solution SOL to the Single Source Rent-or-Buy instance. We will show that its cost must be greater or equal to both $\sum_{j\in J}\alpha'_j$ and $\frac{1}{2}\sum_{j\in J}\alpha_j$.

Consider the bound using $\sum_{j\in J}\alpha'_j$ first. Consider an open ball of radius α'_j drawn around each client j. Note that by construction, no location is covered by more than M balls and the source s lies outside all balls. The solution SOL must provide a path p_j from each user j to the source s. Each path p_j must get from the center of j's ball to its outside, hence it must travel at least α'_j distance inside the ball. We will charge each path p_j the distance it travels inside the ball corresponding to j. We claim that the sum of charges to all paths is less than or equal to the cost of SOL. This is because each rented edge segment was charged at most once, while each bought segment was charged at most M times. Since the sum of radii is less than or equal to the sum of charges, we have $\sum_j \alpha'_j \le \mathrm{cost}(\mathrm{SOL})$.

Now we turn our attention to the second bound. Suppose we came up with a set of lengths $d_{j,e}$ of edges for each user j with the following properties:

1. For every edge e and every user j, $d_{j,e} \le c_e$.
2. For every edge e, $\sum_{j\in J} d_{je} \le 2Mc_e$.
3. The shortest path from every user j to the source s in the metric d_j has length at least α_j.

Again, consider an arbitrary solution SOL. Since every user must have a path to s, we have

$$\alpha_j \le \sum_{e \text{ bought}} d_{j,e} + \sum_{e \text{ rented for } j} d_{j,e}.$$

Summing the above inequalities over all $j \in J$ we obtain

$$\sum_j \alpha_j \le \sum_{e \text{ bought}} \sum_j d_{j,e} + \sum_j \sum_{e \text{ rented for } j} d_{j,e}.$$

We can write the cost of the solution as

$$\mathrm{cost}(\mathrm{SOL}) \;=\; \sum_{e \text{ bought}} Mc_e + \sum_j \sum_{e \text{ rented for } j} c_e.$$

¿From properties (1) and (2) we obtain that

$$\sum_{e \text{ bought}} \sum_j d_{j,e} \le 2 \sum_{e \text{ bought}} Mc_e \quad \text{and} \quad \sum_j \sum_{e \text{ rented for } j} d_{j,e} \le \sum_j \sum_{e \text{ rented for } j} c_e.$$

Combining these inequalities we obtain $\sum_j \alpha_j \le 2\,\mathrm{cost}(\mathrm{SOL})$.

It remains to show that such $d_{j,e}$ exist. For a user j, let $S_j(t)$ be the set of locations *reachable* by j. A location p is reachable by j, if it lies inside j's ghost (i.e. $c(j,p) \le t$), or it lies in a component that j is connected to. An edge e is *crossed* by the set S_j, if it has one endpoint inside s and one outside. We start with all $d_{j,e} = 0$. At every time t, we increase the length $d_{j,e}$ of each edge e that is crossed by S_j at rate $f_j(t)$.

Since at every time point we increase the length of all edges in some j-s cut at rate $f_j(t)$, the length of the shortest j-s path in the d_j metric will be equal to $\int_0^\infty f_j(t)dt = \alpha_j$. The boundary of each S_j moves at speed greater or equal to 1 (it may jump when components merge), hence no edge can accumulate $d_{j,e}$ that is greater than its original length c_e. Finally, consider property (2). Each edge e can be crossed by two types of boundaries, corresponding to ghosts of users and ghosts of components. Each location on e can be crossed by at most M user ghosts, contributing at most one unit of length per unit of time each, and at most one component ghost, contributing M units of length per unit of time. Hence, $\sum_j d_{j,e} \le Mc_e + Mc_e = 2Mc_e$. ∎