A Constant-Factor Approximation for Stochastic Steiner Forest *

Anupam Gupta
Computer Science Dept.
Carnegie Mellon University
Pittsburgh PA 15213
anupamg@cs.cmu.edu

Amit Kumar
Dept. of Computer Science and Engineering
Indian Institute of Technology
New Delhi India 110016
amitk@cse.iitd.ernet.in

ABSTRACT

We consider the *stochastic Steiner forest* problem: suppose we were given a collection of Steiner forest instances, and were guaranteed that a random one of these instances would appear tomorrow; moreover, the cost of edges tomorrow will be λ times the cost of edges today. Which edges should we buy today so that we can extend it to a solution for the instance arriving tomorrow, to minimize the expected total cost? While very general results have been developed for many problems in stochastic discrete optimization over the past years, the approximation status of the stochastic Steiner Forest problem has remained open, with previous works yielding constant-factor approximations only for special cases. We resolve the status of this problem by giving a constant-factor primal-dual based approximation algorithm.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory

Keywords: Approximation Algorithms, Stochastic Algorithms

1. INTRODUCTION

Stochastic combinatorial optimization has received much attention over the past couple of years: in this area, we consider problems where the input is itself uncertain—but is drawn from a probability distribution given to us as input—and the goal is to find strategies that minimize the expected cost incurred. Recent results have shown that for several problems, the stochastic case is "no harder than" the deterministic case, at least from the viewpoint of their approximation guarantees. E.g., for problems like vertex cover, facility location, and set cover, where one can (approximately) solve the stochastic linear program to get fractional solutions—even if costs and demands are both random—and the rounding ideas happen to be robust enough to obtain good integer solutions.

The situation is fairly different for network design problems. In the deterministic case, the Steiner Forest problem [1, 11] is reasonably well-understood, as are many other network design problems whose solutions are built on these ideas. However, prior to this work, we did not understand the approximability of the *Stochastic Steiner Forest* problem. This problem is easily stated: the input is a graph $G = (V, E)$ with edge costs c_e, a probability distribution π over sets of source-sink pairs, and an *inflation* parameter $\lambda \geq 1$. On Monday, we can buy some edges $E_0 \subseteq E$. On Tuesday, the actual demands D arrive, drawn from the distribution π, and we can buy $E_D \subseteq E$, so that $E_0 \cup E_D$ connect up each source-sink pair in D. The objective is to minimize the expected cost

$$\sum_{e \in E_0} c_e + \mathbf{E}_{D \leftarrow \pi}[\, \lambda \cdot \sum_{e \in E_D} c_e \,]. \qquad (1.1)$$

Theorem 1.1 *There is a constant-factor approximation algorithm for Stochastic Steiner Forest.*

For this following discussion, assume that the distribution π is uniform over some m explicitly given sets $\{D_i\}_{i \in [m]}$ of source-sink pairs—we see later that this assumption is without much loss of generality. If we denote $\frac{\lambda}{m}$ by M, the objective function becomes $\sum_{e \in E_0} c_e + M \cdot \sum_{e \in E_{D_i}} c_e$.

Note the special case where each of the sets D_i contains just one source-sink pair: this is the (multi-commodity) rent-or-buy problem, for which constant factor approximation algorithms are known using LP rounding [21] and the boost-and-sample framework [14]. However, these two techniques have not yet succeeded in giving a constant-factor approximation for the stochastic Steiner forest problem, even though they do work for this special case of the problem.

The other promising approach has been to use primal-dual algorithms, but the algorithms are fairly involved even for the case of stochastic Steiner *tree* (where all demands share a common sink), and the natural extensions of these ideas become hopelessly complicated. At a high level, the general approach in these algorithms is to run $M+1$ different moat-growing processes, one for E_0 (called the "core" moat) and

*A. Gupta was partly supported by the NSF awards CCF-0448095 and CCF-0729022, and an Alfred P. Sloan Fellowship. A. Kumar's research partly done at Max-Planck-Institut für Informatik, Saarbrücken, Germany.

STOC'09, May 31–June 2, 2009, Bethesda, Maryland, USA.

one each for the E_{D_i} (called the "scenario" moats). Now if an edge is contributed to by M or more scenario moats, the primal-dual algorithms should instead start loading the edge by the core moat—this corresponds to the intuition that if an edge belongs to more than M sets E_{D_i}, it should instead be added to E_0. In the single-sink case, since all the sources want to be connected to a common sink, this transition from scenario moats to core moat happens only once; however, in the multiple-sink case, there is no such *monotonicity* property, and if some source-sink pairs get satisfied the number of active scenario moats might fall much below M, and so we may have to stop growing the core moat and start growing the scenario moats again. Such an algorithm was given for the rent-or-buy problem [21]—but extending it to the case of Stochastic Steiner forest case seems extremely daunting.

In this paper we use a combination of LP-rounding and primal-dual techniques to simplify and extend the above primal-dual algorithm. In particular, we first solve the LP optimally, but instead of directly rounding this solution, we use the primal solution to filter and decompose the original LP into two different, "simpler" primal-dual LPs. Now our algorithm consists of running dual-growth processes on these two LPs in two "phases": building part of the solution first on one of these simpler LPs, and then on the other. Of course, we have to show that each of these new LPs (suitably scaled down) gives us a lower bound on the optimal value—using these two duals, as well as a third lower bound derived from the optimal LP solution—allows us to bound the cost of the constructed solution.

As is apparent, several technical barriers have to be overcome: for example, when we stop the dual process on the first phase and begin running the second phase, we already have built some partial solution. We do not see how to use a "smooth" uniform dual-growth process employed by almost all primal-dual algorithms, where all dual variables corresponding to active sets are raised at the same rate—instead, we use a hybrid dual-growth process which divides time into intervals of exponentially increasing lengths, and (smoothly) raises a carefully chosen subset of the active dual variables during each of these time intervals. Some of these procedures draw on techniques developed in [21]; however, the new idea of decomposing the dual process into two dual processes (using the optimal LP solution) ends up giving a proof much simpler than that in [21], even though our problem is a strict generalization of the problem in that paper.

1.1 Related Work

The Steiner Forest Problem was one of the problems that showcased the power of the primal-dual schema. A constant-factor approximation was given by Agrawal et al. [1] and was generalized by Goemans and Williamson [11] to a very large class of network design problems.

The study of approximation algorithms for stochastic problems started with [7, 20, 22], and general techniques for solving many such problems were given by [16, 23]. For the stochastic Steiner *tree* problem (with a common sink) in the two-stage model, we know an $O(\log n)$-approximation [20], and $O(1)$-approximations due to [16, 18, 15]; this can be extended to an $2k$-approximation for k stages [17, 19]. All these results hold when the inflation parameter λ is "uniform" (same for all edges): if the inflation can vary over edges, the problem becomes label-cover hard [13]. In many stochastic optimization problems, the support of the probability distribution π (i.e., the number of possible scenarios on Tuesday) can be reduced from exponential to polynomial (in the problem size and inflation factors); see, e.g., [4, 23] for particularly useful forms of this *sample average method*.

The boost-and-sample technique [16] gave $O(1)$-approximation algorithms for many two stage stochastic optimization problems with uniform inflation. The algorithm can be described as follows: In the first stage, sample λ (the inflation parameter) times from the distribution π, and solve the problem on this sampled input (e.g., in the case of Stochastic Steiner Forest, connect these sampled pairs using a good Steiner forest). If there was a cost sharing mechanism for the deterministic version of the problem satsifying two key properties—*fairness* and $O(1)$-*strictness*, then this is an $O(1)$-approximation algorithm for the stochastic problem. Despite progress on some special cases of strictness [9], obtaining these two proerties for the Steiner forest problem remains an open problem.

The *rent-or-buy* problem is a special case of Stochastic Steiner forest when the sets D_i consist of just one source-sink pair: here where we are given a set of source-sink pairs and we need to connect them by single paths by either renting edges (pay c_e times number of paths using e) or buying them (pay a fixed cost of M times c_e). Several $O(1)$-approximation algorithms are known both for the single sink version [12, 10, 14, 24, 8] and also the multi-commodity version [21, 14, 9]. There has been much work on *buy-at-bulk* problems, where the cost of allocating bandwidth on edges is concave and follows natural economies-of-scale: we know an $O(\log n)$-factor approximation [3] in the uniform case, and a polylog approximation in the non-uniform case [5]; it is also known that it is NP-hard to get constant-factor approximations [2].

2. NOTATION AND PRELIMINARIES

In all the problems in this section, we will be given a graph $G = (V, E)$ with edge costs $c_e \in \mathbb{Z}_{\geq 0}$; we will denote this by $G = (V, E, c_e)$ for brevity.

In the *stochastic Steiner forest (SSF)* problem, we are given a graph $G = (V, E, c_e)$ and a probability distribution π over $2^{\binom{V}{2}}$, and an inflation parameter λ. The goal is to buy a set of first-stage edges E_0 and, for each $D \in 2^{\binom{V}{2}}$, a set of second-stage edges E_D such that (i) the edges in $E_0 \cup E_D$ connect each of the pairs in D, and (ii) the expected cost $\sum_{e \in E_0} c_e + \lambda \cdot \mathbf{E}_{D \leftarrow \pi}[\sum_{e \in E_D} c_e]$ is minimized. The sample-average approximation technique (see, e.g., [4]) implies that to obtain an $\alpha(1+\epsilon)$ approximation algorithm for this problem, it is enough to give an α-approximation for the problem where π is the uniform distribution on $m = \text{poly}(n, \lambda, \epsilon^{-1})$ sets $D_1, D_2, \ldots, D_m$. Hence the objective function now is:

$$\sum_{e \in E_0} c_e + \tfrac{\lambda}{m} \cdot \sum_{i=1}^{m} \sum_{e \in E_{D_i}} c_e. \tag{2.2}$$

The SSF problem is equivalent to the following *group multicommodity rent-or-buy problem (GMROB)* problem: we are given a graph $G = (V, E, c_e)$, a collection of *demand groups* $D_1, \ldots, D_m$, with each demand group D_k containing a set of source-sink pairs. Now, for each k, we want to build a Steiner forest T_k connecting all the source-sink pairs in the demand-group D_k; the cost of an edge e in such a solution is $c_e \times \min\{M, f_e\}$, where f_e is the number of forests T_k which contain this edge. To see the equivalence, define $T_k = E_0 \cup E_{D_k}$ and set $M = m/\lambda$.

However, we will not deal directly with either of these SSF or GMROB problems. Instead we will work with the problem called *group multicommodity connected facility location problem (GMCFL)*: the input is exactly the same as for the GMROB problem. Let the term "demand" refer to a vertex in one of the demand groups. A solution opens a set of *facilities* and *assigns* each demand j to a facility f_j. It also connects the open facilities by a Steiner forest T, which satisfies the property that for every source-sink pair (s_l, t_l), the facilities to which s_l and t_l are assigned should lie in the same component of this forest. Moreover, for each facility i and demand group D_k it builds a Steiner tree connecting the demands in D_k which are assigned to i and the vertex i – call this tree $T_i(D_k)$. We want to minimize the cost of the solution $\sum_{k,i} \sum_{e \in T_i(D_k)} c_e + M \cdot \sum_{e \in T} c_e$. We shall often call the first term as the *rental cost* and the second term as the *buying cost*. Given an instance $\mathcal{I}$ of GMROB/GMCFL, the following theorem relates the optima of two problems.

Theorem 2.1 *Any solution for GMROB on $\mathcal{I}$ can be transformed to a solution for GMCFL whose cost is at most twice the original cost. Conversely, a solution for GMCFL on $\mathcal{I}$ can be transformed to a solution for GMROB without increasing the cost.*

PROOF. Let S be a solution for an instance $\mathcal{I}$ of GMCFL. Let $\mathcal{I}'$ be the corresponding instance of GMROB. Let T be the Steiner forest used to connect the open facilities, and $T(D_k) = \cup_i T_i(D_k)$ be the Steiner forest corresponding to demand-group D_k. We get a solution S' for $\mathcal{I}'$ as follows – the forest T_k is just the union of $T(D_k)$ and T. We rent each edge in $T(D_k)$ and buy every edge in T.

Conversely suppose we are given an instance $\mathcal{I}$ of GMROB. Let S be a solution to this instance. We transform S to a solution S' for the corresponding instance $\mathcal{I}'$ of GMCFL. Let E_B be the set of edges in S which get bought. It is clear that E_B is a forest, otherwise we can remove an edge from a cycle in E_B, and still get a feasible solution.

Start with a component C of E_B. Consider a demand group D_k – recall that the forest connecting the demands for D_k is T_k. We can first assume that if a component of T_k contains a vertex of C, then it contains all the edges and vertices of C – indeed, we can merge all such components of T_k into a single component without increasing the cost. Further, we can also assume that this component is a tree – otherwise we can delete some rented edges from T_k without increasing the cost of the solution. Note that in this transformation, we only remove rented edges. We do this for all demand groups. We shall call this transformation as *simplifying the solution with respect to C*.

For a demand group D_k, let T'_k be the component of T_k which contains C (if there is such a component). Let E_1 be the edges rented by D_k in T'_k and E_2 be the bought edges in $T'_k - C$. Let D'_k be the demands in D_k present in the component T'_k (note that D'_k must contain pair of every demand in it). Two cases arise:

- Total length of the edges in E_1 is more than that of E_2 : D_k rents all the edges in E_2 – we charge the extra rental cost to that of E_1.

  We open a facility at every vertex of C. Now we show how to assign demands in D'_k to facilities. Contract C to a single vertex in T'_k. Since C forms a connected set vertices in T'_k the resulting graph is still a tree T''. The edges in T'' correspond to $E_1 \cup E_2$, all of which are rented by D_k. Root T'' at the vertex corresponding to C – call this vertex c. Let x be a child of c and consider the subtree E'_x rooted at x. The edge (c, x) corresponds to an edge (y, x) in the uncontracted graph T'_k. Here y lies in C. We assign all the demands of D'_k lying in E'_x to y, and connect them to y by the edges $E'_x \cup \{(x, y)\}$. Now, we do not consider the edges in T'_k in the set T_k, this way D_k does not charge to E_1 again. Since we have already assigned D'_k we remove this set of demands from D_k as well.

- Total length of the edges in E_1 is less than that of E_2 : we buy all edges in E_1 – we charge the cost to that of E_2. Note that we do not charge to C in this process. We update C to this new component.

  Each demand in D'_k itself becomes a facility and is assigned to itself. We remove D'_k from the set D_k.

We repeat this process with the set C as defined above. If the second case above happened, C now contains the edges in E_2, and so we do not charge to E_2 again. Further we simplify the solution with respect to C. We repeat this process as long as there is a component of T_k (for some k) containing C. When this process ends, we set C to be another component of bought edges and continue as above. Thus we get a solution to $\mathcal{I}'$. Further we pay at most twice the cost of the solution corresponding to $\mathcal{I}$. □

3. LP RELAXATION FOR GMCFL

By the reductions from the previous section, it suffices to give a constant factor approximation algorithm for the GMCFL problem. Fix an instance $\mathcal{I}$ of this problem as described above. We now give an LP relaxation for this problem.

For each edge e, demand group D_k and vertex i, we have a variable $f_e^{(k,i)}$ which is 1 if this edge is used to connect a demand from D_k to the facility i (i.e., $e \in T_i(D_k)$), 0 otherwise. For each edge e, we also have a variable z_e which is 1 if we use this edge to connect the facilities, 0 otherwise. Finally, we have variables x_{ij} for each demand j and facility i, which is 1 if we assign j to i, and 0 otherwise. We now write the LP relaxation (we call this the *main LP*):

$$\min \sum_{k=1}^{m} \sum_i \sum_e f_e^{(k,i)} c_e + M \sum_e c_e \qquad \text{(main-LP)}$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad \text{for all demands } j \qquad (3.3)$$

$$\sum_{e \in \delta(S)} f_e^{(k,i)} \geq x_{ij} \qquad \forall D_k, \forall j \in D_k, \forall i, S : j \in S, i \notin S \qquad (3.4)$$

$$\sum_{e \in \delta(U)} z_e \geq \sum_{i \in U} x_{is_l} - \sum_{i \in U} x_{it_l} \ \forall U, (s_l, t_l) \qquad (3.5)$$

$$\sum_{e \in \delta(U)} z_e \geq \sum_{i \in U} x_{it_l} - \sum_{i \in U} x_{is_l} \ \forall U, (s_l, t_l) \qquad (3.6)$$

$$x_{ij}, f_e^{(k,i)}, z_e \geq 0$$

Let OPT denote the optimal value of this LP. We now show how to round this solution. Let Z be a large enough constant.

3.1 High-Level Algorithm

Given the instance $\mathcal{I}$, we first solve the main-LP to get an optimal solution. We then round it to get an integral solution for the corresponding GMROB instance $\mathcal{I}'$. We would

like to run a primal-dual algorithm, but the dual of this LP looks very daunting. So we run our algorithm in two phases, where each phase will run a primal-dual algorithm on a simpler version of the main LP.

In the first phase, we only rent edges. These rental edges would be enough to account for the rental cost of our solution. This should be thought of as the filtering step in the facility location problem – the filtering step decides the connection cost of each demand (upto a constant factor) in the instance. We shall use the following observation in the first phase : suppose there is a demand j and a set S, $j \in S$, such that $\sum_{i \in S} x_{ij} < 0.9$. Then the total rental capacity across the cut S, i.e., $\sum_{e \in \delta(S)} \sum_i f_e^{(k,i)}$ (here D_k is the demand group containing j) is at least 0.1. So we can run a primal-dual algorithm for renting edges for each demand group D_k, where we can grow a moat S as long as it satisfies the condition mentioned above. At the end of phase one, we have a set of connected components of rental edges (for each demand group D_k).

In the second phase, we shall both rent and buy edges. But the cost of rental edges will be at most a constant times that in the first phase. The key observation in the second phase is this: suppose S is a set of vertices and (s_j, t_j) is a demand pair such that $\sum_{i \in S} x_{is_j} - \sum_{i \in S} x_{it_j}$ is at least a constant. Then we know that there is a constant buying capacity across this cut. So while running a primal-dual algorithm for buying edges, we can grow a moat around such a set S.

This idea has several problems though – (i) unlike phase one, we cannot start with a moat as a single demand or a vertex. In fact to start with, a moat will in general be a ball around a demand j. So we need to make sure that the demand j has enough rental edges to reach the boundary of this ball; (ii) For different demands, these balls can be of different radii. So we cannot start growing moats around all of them simultaneously. We get around this by dividing this phase into *stages* : in stage i, we grow moats around balls of radius about Z^i, where Z is a large constant. Achieving these two properties requires several technical details, which we outline in the algorithm description.

3.2 Phase I : Renting Edges Within Groups

For any $v \in V$, define the *ball* $\mathbf{B}(v, r) = \{i \in V | d(j, i) \leq r\}$, where d is the distance metric with respect to edge costs c_e in G. For a set $S \subseteq V$ and demand j, define $A(S, j) := \sum_{i \in S} x_{ij}$ to be the assignment of j inside S. Define the α-*radius* of j as the smallest radius r such that the assignment to facilities within this ball is at least α; i.e., $\min\{r \mid A(\mathbf{B}(j, r), r) \geq \alpha\}$. Define $r_\alpha(j)$ to be the α-radius for j.

Definition 3.1 (Demand Type) *A demand j is of* type l *if its* 0.8*-radius lies in the range $[Z^{l-1}, Z^l]$; if the* 0.8*-radius is less than* 1 *(i.e., it is 0), then j is of* type 0*.*

Consider the following primal-dual pair, which we call *LP1* and (with variables $f_e^k \geq 0$), and (DP1) (with variables $y_S \geq 0$):

$$\min \sum_e f_e^k \cdot c_e \qquad \text{(LP1)}$$
$$\text{s.t.} \sum_{e \in \delta(S)} f_e^k \geq 1, \ \forall S \text{ such that } A(S, j) \leq 0.9 \text{ for some } j \in D_k$$

$$\max \sum_S y_S \qquad \text{(DP1)}$$
$$\text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq c_e, \quad \text{for every edge } e$$

Note that the variables y_S are defined only for sets S such that $A(S, j) \leq 0.9$ for some $j \in D_k$. Let $\mathsf{OPT1}$ denote the optimal value of (LP1).

Claim 3.2 $\mathsf{OPT1} \leq 10 \cdot \mathsf{OPT}$.

Proof. Consider an optimal solution to (main-LP). Define $f_e^k = \sum_i f_e^{(k,i)}$. Constraint (3.4) implies that $\sum_{e \in \delta(S)} f_e^k \geq \sum_{i \notin S} x_{ij}$, and hence if $A(S, j) \leq 0.9$, (3.3) implies that $\sum_{e \in \delta(S)} f_e^k \geq 1 - 0.9 = 0.1$. Since $10 f_e^k$ satisfies (LP1), the value $\mathsf{OPT1}$ of (LP1) is at most $10 \cdot \mathsf{OPT}$. □

3.2.1 The First-Phase Algorithm

We run the following algorithm for each demand group D_k independently. We initialize dual variables $y_S \leftarrow 0$ for the relevant subsets S defined in (DP1). We run the Goemans and Williamson (GW) moat-growing algorithm with the initial moats being the demands in D_k. A demand j remains *active* as long as the moat $\mathcal{M}$ containing j satisfies the condition that $A(\mathcal{M}, j) \leq 0.9$. A moat is *active* if it contains at least one active demand. We grow the active moats at the same rate. For each moat, we maintain a tree of tight rented edges that connect all nodes in the moat. When two moats meet, they merge into a single moat—note that one of the two moats must have been active—and we get a single component of rented edges inside the moat. Upon merging, the moat might become inactive. While growing a moat $\mathcal{M}$, we also raise variables y_S at the same rate, where S is the set of vertices corresponding to $\mathcal{M}$.

When the process stops, we get a forest F of rented edges—each tree in this forest corresponds to a single moat. Now we keep a subset F' of the edges in F, since we cannot pay for all the edges in F. For an edge $e \in F$, let t_e be the time at which e was added to F. For a demand j, let t_j denote the time at which j became inactive. We add e to F' if one of the following conditions hold:

- There exist two demands j, j' such that e lies on the path between them in F and $t_j, t_{j'} \geq t_e$. *(This is the usual criterion used in primal-dual algorithms.)*
- There exist two demands j, j', each of type at least l, such that e lies on the path between them in F, and $t_e \leq Z^l$.

This completes the description of how we get the forest F' connecting some of the demands in D_k. During our algorithm we also assign *charge* to some of the demands as follows: intially the charge of all demands is 0. For an active moat $\mathcal{M}$ we increment the charge of a demand of highest type in $\mathcal{M}$ at the same rate at which $\mathcal{M}$ is growing. Our algorithm also maintains a set J of demands as follows: we add a demand j to J if at the time t_j when it becomes inactive, j is the only inactive demand of its type in the moat containing it (we are assuming that the times t_j are distinct, which can be forced by breaking ties in some arbitrary but fixed order). Let J_l denote the set of demands of type l in J.

We now bound the cost of the edges in F'. Claim 3.2 shows that the value of (LP1) lower bounds the value of (main-LP),

but this will not be strong enough, and hence we prove a second lower bound on the value of (main-LP). We say that a group of demands K are β-*disjoint* if there exist mutually disjoint sets $\{S_j\}_{j\in K}$ such that $A(S_j, j) \geq \beta$ for all $j \in K$.

Theorem 3.3 *Let J be a set of demands, and $\beta > \alpha > 0$ be constants. Let J_l be the set of demands in J whose α-radius lies in the range $[Z^{l-1}, Z^l]$, $l \geq 1$. Further assume that the demands in J_l are β-disjoint for all l. Then $\sum_l |J_l| \cdot Z^l = O(\mathsf{OPT})$.*

PROOF. For each demand $j \in J$, let S_j be the set of facilities given by the definition of β-disjointness: i.e., $A(S_j, j) \geq \beta$, and moreover the sets S_j are disjoint for demands $j \in J_l$. If $j \in J_l$, let $S'(j) \subseteq S_j$ consist of those facilities $\{i \mid d(i,j) \geq Z^{l-1}\}$. Observe that $\sum_{i\in S'(j)} x_{ij} \geq \beta - \alpha$, since the total assignment of j to facilities at distance less than Z^{l-1} from j is at most α (by the assumption of the theorem).

For a facility i, let $\Delta(i) = \sum_e f_e^{(k,i)} \cdot c_e$. Since the sets $S'(j)$ are disjoint for $j \in J_l$, at most one of these sets can contain i: let this set be $S'(j_l)$ (if such a set exists). We now claim that $\sum_l Z^l \cdot x_{ij_l} = O(\Delta(i))$. Indeed, consider a ball B of radius between Z^{l-1} and Z^l around i. Since j_{l+1} lies outside this ball, at least $x_{ij_{l+1}}$ amount of $f_e^{(k,i)}$ value must be entering this ball. Integrating over the radius of the ball, we get the result.

Adding up over all i, we get $\sum_i \sum_l Z^l \cdot x_{ij_l} = \sum_l Z^l \cdot \sum_i x_{ij_l}$. But $\sum_i x_{ij_l} = \sum_{j\in J_l} \sum_{i\in S'(j)} x_{ij} \geq (\beta - \alpha) \cdot |J_l|$. Thus we get $\sum_i \sum_l Z^l \cdot |J_l| = O\left(\sum_i \Delta(i)\right) = O(\mathsf{OPT})$. This proves the theorem. □

Corollary 3.4 *Consider the set J constructed during the algorithm. Let J_l be the demands of type l in J. Then $\sum_l |J_l| \cdot Z^l = O(\mathsf{OPT})$.*

PROOF. It satisfies to prove that J satisfies the conditions of Theorem 3.3 with $\alpha = 0.8, \beta = 0.9$. Let $j, j' \in J_l$. Suppose $t_j < t_{j'}$ and let $\mathcal{M}$ and $\mathcal{M}'$ be the moats containing j and j' respectively at the time these demands become inactive, i.e., at times t_j and $t_{j'}$ respectively. We claim that $\mathcal{M}$ and $\mathcal{M}'$ are disjoint: indeed, if they were not disjoint, the nested structure of the moats would imply that $\mathcal{M}'$ contains $\mathcal{M}$. But j' should not have been added to the set J in our algorithm, since at the time $t_{j'}$, the inactive demand j would already belong to the moat $\mathcal{M}'$.

Moreover, the very fact that j is inactive at time t_j implies that $A(\mathcal{M}, j) \geq 0.9$, and similarly for j'. Hence these disjoint moats give us witnesses for the demands in J_l being 0.9-disjoint. The proof now immediately follows from Theorem 3.3. □

Theorem 3.5 *The total cost of the edges in F' is $O(\sum_S y_S + \sum_l |J_l| \cdot Z^l)$, where y_S and J are as in the primal-dual algorithm. Moreover, if C be a component in F' such that C contains a demand of type l, then either the total charge assigned to the vertices in C is at least Z^l or C contains a vertex of $J_{\geq l} = \cup_{i\geq l} J_l$.*

PROOF. It is easy to see that the variables y_S constructed are feasible for the dual (DP1). If $e \in F'$, then $c_e = \sum_{S:e\in\delta(S)} y_S$. So it is enough to argue that $\sum_S \delta_{F'}(S) \cdot y_S \leq 2 \cdot \sum_S y_S + \sum_l Z^l \cdot |J_{\geq l}|$, where $\delta_{F'}(S) = |F' \cap \delta(S)|$. As in the analysis of the GW algorithm, we show that the increase in LHS is bounded by the increase in the RHS.

Fix a time t between Z^{l-1} and Z^l. Let $\mathcal{S}$ be the moats at this time. Let $\mathcal{S}'$ be the active moats at this time. Suppose we contract each moat in $\mathcal{S}$ to a single vertex. Let F'' be the subset of F' remaining in this new graph. Let U and U' be the set of vertices corresponding to the moats $\mathcal{S}$ and $\mathcal{S}'$. We essentially need to argue that the average degree of the nodes in U' in F'' is at most 2. In the GW analysis, it is shown by the proving the following facts : (1) If degree of a vertex in F'' is 1, then this vertex lies in U'; (2) The average degree of the nodes in a forest is at most 2.

For us we prove the following modified version of (1). Suppose the degree of a vertex in F'' is 1. Let S be the moat corresponding to this vertex in the original graph. Why is $\delta_{F'}(S) = 1$? Let e be the edge in $F' \cap \delta(S)$. Then e is in F' because of one of the following two reasons (note that e gets added after time t, so $t_e > t$) : (i) e joins two demands which become inactive after time $t_e > t$ – one of these must be in S. But then S has to be active at time t; (ii) e joins two vertices from $J_{\geq l}$: one of these vertices must be in S. It must be inactive at time t (otherwise S will be active). Then one of the vertices in S must belong to $J_{\geq l}$ — the first vertex in S of type at least l which became inactive belongs to J.

Thus, the increase in RHS in a unit time can be bounded by $|J_{\geq l}| + 2|\mathcal{S}'|$. Since the set J_l participates in this sum only till time Z^l, we are done. Now we prove the second part of the theorem. It follows from the following simple observation : if two demands of type at least l belong to the same moat at time Z^l, they lie in the same component of F'. Indeed, all edges connecting such vertices were added before time Z^l and so belong to F' (using rule (ii)). Now consider a component C of F' containing a demand j of type l. Let X be the component containing j at time $t = Z^l$ during the dual raising algorithm. Let X' be the subset of X containing jobs of type at least l. Then all of X' must be in C. Two cases arise : (i) X is inactive at time t. Then X' must have at least one vertex which gets added to J; (ii) X is active at time t – then the algorithm must have been charging at least one vertex in X' during each time $t' < t$. This proves the theorem. □

Thus we can account for the rental cost of F'. We can also associate the following *charge* with each component C in the forest F' : Z^{l^*}, where $l^* = \max\{l : C \text{ has a demand of type } l\}$. This is so because the total charge of the demands is equal to the total dual value raised; and every demand in J_l can get paid Z^l amount (Corollary 3.4).

3.3 Phase II: Connecting Demands

We now give the main algorithm in this section which shows how to connect the demands. As mentioned earlier, we use the following key observation : if S is a set of vertices such that $A(S, s_j) - A(S, t_j) \geq 0.1$(say), then $\sum_{e\in\delta(S)} z_e \geq 0.1$. We identify such a subset $S(j)$ for each demand j. These subsets should be such that j can rent to any vertex in this set. We would like to contract these sets and run the primal-dual algorithm for Steiner forest. But these sets are of different radii and may intersect with many other sets. So we solve this problem by dividing the primal-dual algorithm in stages – in stage i, we only look at those demands for which

the radius of $S(j)$ is at most Z^i. Further, we pick a subset of such demands so that the sets $S(j)$ are far apart from each other so that the effect of contracting these sets will not make much difference, i.e., the distance between any two points will get distorted by a constant factor and an additive factor of about Z^i. We will also make sure that the moats will grow for at least Z^{i+1} units of time in stage i. The dual value accrued during this moat growing process will allow us to buy edges.

3.3.1 Some More Definitions

Recall Definition 3.1 defining the type of demands: assume for all source-sink pairs (s_k, t_k) that the type of s_k is at most that of t_k. We now define a slightly related quantity, the *class* of a demand j.

Definition 3.6 (Demand Class) *Given (s_k, t_k), define $\mathsf{class}(s_k)$ to be equal to its type, and $\mathsf{class}(t_k) = \max\{\mathsf{class}(s_k), l\}$, where l is such that the 0.4-radius of t_k lies between Z^{l-1} and Z^l.*

By definition, $\mathsf{class}(s_k) \leq \mathsf{class}(t_k)$. Note that the definition is asymmetric: the reason for this asymmetry will become clear in the analysis of our algorithm.

For each demand j, we define a set $S(j)$ of facilities as follows. Define $S(s_k) = \mathbf{B}(s_k, r_{0.8}(s_k))$, and $S(t_k) = \mathbf{B}(t_k, r_{0.4}(t_k))$— hence $A(S(s_k), s_k) \geq 0.8$ and $A(S(t_k), t_k) \geq 0.4$. Note that the asymmetry in this definition matches the asymmetry in the Definition 3.6 for class above.

3.3.2 The "Reduced" LPs

Consider the main LP. We say that a set S *is* valid for s_k if S contains $S(s_k)$, but $A(S, t_k) \leq 0.7$. We say that S *is valid for* t_k if S contains $S(t_k)$ and $A(S, s_k) \leq 0.3$. If S is valid for some demand j, constraints (3.5, 3.6) imply that $\sum_{e \in \delta(S)} z_e \geq 0.1$. It turns out that this is the only property of the z_e variables required by our algorithm. So we can write down an alternate linear program:

$$\min \quad M \cdot \textstyle\sum_e c_e z'_e \tag{LP2}$$
$$\text{s.t.} \textstyle\sum_{e \in \delta(S)} z'_e \geq 1, \quad \text{for all } S \text{ such that } S \text{ is valid for some } j$$

We argued above that $10 \cdot z$ is a feasible solution to (LP2), and so the optimal value of (LP2) is a lower bound (upto a factor of 10) on the optimal value of (main-LP). To write the dual, define variables y_S, where S is valid for *some* demand.

$$\max \quad \textstyle\sum_S y_S \tag{DP2}$$
$$\text{s.t.} \textstyle\sum_{S: e \in \delta(S)} y_S \leq M \cdot c_e \tag{3.7}$$

3.3.3 The Various Graphs: G, G_B, G_R

Our algorithm will maintain three different graphs, each obtained by contracting some vertices in G :

Graph G_B: At any point of time, we would have *bought* some edges in G. The graph G_B is obtained by contracting all the bought edges. Hence, each vertex of G_B corresponds to a connected subgraph of G.

Graph G_R: As we raise the dual variables in (DP2), we will declare some edges *tight*. This can happen for edge e if *(i)* the inequality (3.7) for e gets satisfied with equality, or *(ii)* we buy the edge e. The graph G_R is obtained by contracting all the tight edges in G. Since all bought edges are declared tight, each vertex in G_B can be thought of as contained inside a unique vertex in G_R.

The relationship between these graphs is conceptually simple: the graph G_B is obtained by contracting some edges in G, and G_R is obtained from G_B by contracting some more edges.

- For a vertex $v \in G_B, G_R$, define $G[v]$ as the (connected) subgraph of G corresponding to v. Similarly define $G_B[v]$ for a vertex $v \in G_R$.
- For a vertex $v \in G$, define $v(G_B)$ as the vertex in G_B which *contains* v. Similarly, define $v(G_R)$ for v in G or G_B.
- For a vertex $v \in G_R$, there is a special vertex $\mathsf{core}(v) \in G_B[v]$. The idea behind the core is this: to build a path through v, we will need to go from the boundary of $G_B[v]$ to $\mathsf{core}(v)$ in G_B.
- Each vertex v in G_R will have a *weight* associated with it. The weight will roughly denote the sum of the dual variables y_S for subsets S such that $S \subseteq G[v]$ which have not been used yet for buying edges; this weight will be a power of some large constant Z.

3.3.4 Buying or Renting Edges in G_R

At various points during our algorithm, we will *buy* an edge $e = (u, v)$ in G_R. This corresponds to buying edges in G_B (and hence in G) to connect the cores of u and v as follows: if the end-points of e in G_B are u' and v' respectively (so $u' \in G_B[u], v' \in G_B[v]$), then we buy a shortest-path from $\mathsf{core}(u)$ to u' in $G_B[u]$, the edge (u', v'), and a shortest-path from v' to $\mathsf{core}(v)$ in $G_B[v]$. To buy a path P from u to v in G_R, we buy paths for each edge in P as above, thus buying a path from $\mathsf{core}(u)$ to $\mathsf{core}(v)$ in G_B: denote this path by $P[G_B]$. Similarly, renting a path in G_R corresponds to renting a path in G_B.

3.4 The Primal-Dual Algorithm

Initially the graphs are identical $G_R = G_B = G$, and for all $v \in G_R$, $\mathsf{core}(v) = v$ and $\mathsf{weight}(v) = 0$. Our algorithm starts with the feasible dual solution $y_S = 0$ for all S. We shall say that a demand ω is *active* if we have not routed it to its mate yet; initially all demands are active.

The primal-dual algorithm will run in several *stages*: we ensure that each dual variable increases by a (roughly) exponential amount in each stage. We assume that any two demands in G are either co-located at the same vertex or the distance between them is at least Z—this can be achieved by initially scaling the distances in G.

For the demand group D_k, we will maintain a collection $\mathcal{C}(k)$ of connected components; we start off with the connected components formed in Phase I. For each connected component $C \in \mathcal{C}(k)$, we associate a *charge* of Z^l, where l is the highest type of any demand in C. (Note that this is different from the *weights* of vertices in G_R.) As the algorithm proceeds, we may connect these components together (using rented or bought edges), and hence the set $\mathcal{C}(k)$ will change. Whenever we merge two connected components C_1, C_2 in $\mathcal{C}(k)$, the charge of the new component will be the maximum of the charges of C_1 and C_2. Define $\mathcal{C}(k, i)$ as the

set $\mathcal{C}(k)$ at the beginning of stage i in the algorithm below (so $\mathcal{C}(k,0)$ is the initial set of connected components from Phase I). Recall that for a demand j and set S, $A(S,j)$ is the assignment of j inside S. Here S is a subset of vertices in G, but we shall extend this definition to the case when S is a subset of vertices in G_R in natural manner.

3.4.1 Stage i of the Algorithm

Let us now describe a generic stage i of the primal-dual algorithm. The algorithm starts with $i = 0$.

Step 1: Merging Components. If there are two components C_1, C_2 in $\mathcal{C}(k)$ for some k with charges Z^{l_1}, Z^{l_2} respectively such that (i) $i \le l_1 \le l_2$ and (ii) the distance between them in G_R is at most Z^{l_1+2}, we *rent* edges on the shortest path between them in G_B to connect the two components. The charge associated with the resulting new component is Z^{l_2}. We repeat this step as long as possible.

Step 2 : Deactiviating Demands. An active demand pair $(s_p, t_p) \in D_k$ is made inactive if one of the following conditions hold (with preference for case (i) over case (ii)):

(i) if s_p and t_p belong to the same component of $\mathcal{C}(k)$.

(ii) if $\mathsf{class}(s_p) \le i \le \mathsf{class}(t_p)$, let B denote the ball of radius $10 \cdot Z^{i+1}$ around $s_p(G_R)$. If $A(B, t_p) \ge 0.7$, then let B_G denote the set of vertices in G corresponding to B, i.e., $B_G = \cup_{v \in B} G[v]$. We say that s_p *gets associated with* the set B_G. While we make the pair (s_p, t_p) inactive, we do not connect s_p and t_p right now: we shall call such pairs *special* and connect them after the last stage, as explained in Section 3.4.2.

Step 3: Choosing Representatives. In this step we select some vertices in G_R around which we shall grow moats. Let $J_1 \subseteq G_R$ be the set of vertices v of weight Z^i such that $G[v]$ contains $S(j)$ for some active demand j of class *less than* i. Let $\mathbf{A}_i$ be the set of active demands of class i. We greedily pick a maximal subset $\mathbf{A}'_i$ of these active demands $\mathbf{A}_i$ which has the following property—for any $\omega \in \mathbf{A}'_i$, the distance of $\omega(G_R)$ from $\omega'(G_R)$ for any other $\omega' \in \mathbf{A}'_i$, or from any vertex in J_1 is at least Z^{i+1}. Define $J_2 = \{\omega(G_R) \mid \omega \in \mathbf{A}'_i\} \subseteq G_R$. Let $J = J_1 \cup J_2$.

Step 4: In this step, we connect each demand in $\mathbf{A}_i$ to one of the vertices in J. Consider the group D_k and some component C of the forest $\mathcal{C}(k,0)$ for D_k at the beginning of Phase II. Define $\mathbf{A}_i^k = \mathbf{A}_i \cap D_k$ and $\mathbf{A}_i^k(C)$ to be the demands in $\mathbf{A}_i^k$ which belong to C. We find a maximal subset of demands $\mathbf{A}_i^k(C)'$ of $\mathbf{A}_i^k$ such that $d_{G_R}(j, j') \ge Z^{i+1}$ between any two demands j, j' in this subset. For every vertex $j \in \mathbf{A}_i^k(C)'$, we find a vertex $v_j \in J$ closest to it and *rent* the following path: first we *rent* a path in G_R from $j(G_R)$ to v_j. This gives a path in G_B from $\mathsf{core}(u)$—where $u = j(G_R)$—to $\mathsf{core}(v_j)$. Then we *rent* a path in G_B from $j(G_B)$ to $\mathsf{core}(u)$ in $G_B[u]$. Thus we connect $j(G_B)$ to $\mathsf{core}(v_j)$ in G_B. We do this for every component $C \in \mathcal{C}(k,0)$ and every demand group D_k (see Figure 3.4.1).

Step 5: Growing Moats and Duals. We grow moats in G_R at the same rate around all vertices in J for $4 \cdot Z^{i+1}$ units of time. As usual, when two moats meet, we *buy* edges between the centers of the moats.

Since this moat-growing process happens on G_R, we need to specify how to raise the dual variables of (DP2), which correspond to sets in G. We will show (in Lemma 4.9) that at the beginning of this step, the distance in G_R between any two vertices in J is at least Z^{i+1}. Hence, for the first $Z^{i+1}/4$ units of time, no two moats shall meet in the moat-growing process on G_R. We raise the duals during the time interval $[Z^i, Z^{i+1}/4]$ thus: For a moat $\mathcal{M}$ around a vertex $v \in J$, we raise y_S at M times the rate at which the moat $\mathcal{M}$ was grown (during this time period), where $S = \cup_{x \in \mathcal{M}} G[x]$. We prove (in Theorem 4.10) that S is valid for some demand ω.

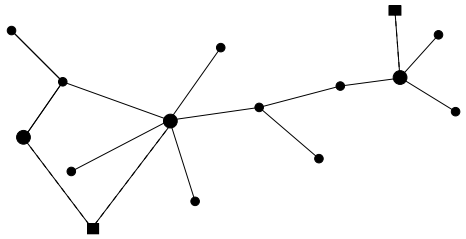

Figure 1. C is a component in $\mathcal{C}(k,0)$ in Step 4. The vertices with large circle represent vertices in $\mathbf{A}_i^k(C)'$. The square vertices represent vertices in J and and the dotted lines represent rental paths from the vertices in $\mathbf{A}_i^k(C)'$ to the nearest vertex in J

When the process stops, we contract each moat into a single vertex in G_R and give this vertex a weight of Z^{i+1}. The core of this vertex is defined to be the new vertex in G_B obtained by contracting the union of the cores of each of the vertices in J which belong to this moat, along with the edges bought to connect these cores.

Step 6: Cleaning Up. Suppose we find a path P between two vertices $u, v \in G_R$, each having weight Z^{i+1}, such that (a) P does not contain an internal vertex of weight Z^{i+1}, and (b) $\mathsf{length}_{G_R}(P) \le \gamma \cdot \sum_{x \in P} \mathsf{weight}(x)$, where γ is a constant to be specified later. Then we *buy* the path P (i.e., the edges in $P[G_B]$), and contract P to a single vertex x. We set $\mathsf{weight}(x)$ to Z^{i+1}, and the core of x is the vertex in G_B obtained by contracting the path $P[G_B]$.[1] We repeat this process until no such path exists.

This completes the description of stage i of the algorithm; let the last stage be denoted $f - 1$. It remains to show how to connect the special pairs.

3.4.2 The Final Step: Connecting Special Pairs

We show how to connect special pairs belonging to a demand group D_k; this procedure is done for each value of k. Let $\mathcal{C}(k,f)$ be the set of components $\mathcal{C}(k)$ at the end of the last stage. Observe that if (s_p, t_p) and $(s_{p'}, t_{p'})$ are two special pairs such that $s_p, s_{p'}$ and $t_p, t_{p'}$ lie in the same component of $\mathcal{C}(k,f)$ respectively, then it is enough to connect s_p to t_p. Define an equivalence relation on the special pairs where (s_p, t_p) and $(s_{p'}, t_{p'})$ are related if they satisfy the property above: pick a representative pair from each

[1] An (important) aside: Finding such a path P is computationally infeasible in general. However, if such a path P exists, then we can efficiently find a path Q in satisfies property (a) and has length at most twice that promised by (b). For this, we guess $\mathsf{length}_{G_R}(P)$ and use a 2-approximation algorithm for the orienteering problem [6], with the profit of each vertex x set to $\gamma \cdot \mathsf{weight}(x)$. Using this path Q suffices for our algorithm.

equivalence class, and call this set of special pairs $\mathcal{S}$—note that it is enough to connect these pairs.

For each l, define $\mathcal{S}^{(l)}$ be those pairs $(s_p, t_p) \in \mathcal{S}$ such that $\mathsf{class}(t_p) = l$. We build a digraph $G^l = (V^l, E^l)$ with vertex set $V^l = \{s_p \mid (s_p, t_p) \in \mathcal{S}^{(l)}\}$ as follows. Let $\mathcal{S}^{(l,i)}$ be those vertices $s_p \in V^l$ such that s_p becomes inactive in stage i. By the requirements of Step 2(ii), it follows that any such s_p satisfies $\mathsf{class}(s_p) \le i \le l$. Let $B_G(s_p)$ be the set of vertices in G associated with it in Step 2(ii). We further classify each demand $s_p \in \mathcal{S}^{(l,i)}$ as *old* if $\mathsf{class}(s_p) < i$ and as *new* if $\mathsf{class}(s_p) = i$. For the arc set E^l, consider $s_p \in \mathcal{S}^{(l,i)}, s_{p'} \in \mathcal{S}^{(l,i')}$ with $i < i'$. We create an arc $(s_{p'}, s_p) \in E^l$ if the associated balls intersect (that is, $B_G(s_{p'}) \cap B_G(s_p) \neq \emptyset$) and $s_p, s_{p'}$ are either both new or both old. The proof of the following claim appears in Section 4.3.

Claim 3.7 *Given the digraph $G^l = (V^l, E^l)$ above, the set V^l can be partitioned into a set of disjoint in-arborescences $\mathcal{T}^l$ such that:*

(i) For each tree $T \in \mathcal{T}^l$, the demands $\{t_p : s_p \in T\}$ belong to the same component of $\mathcal{C}(k, f)$;

(ii) The sets in $\{B_G(s_T) : s_T$ is the root of a tree $T \in \mathcal{T}^l$ and is a new demand $\}$ are mutually disjoint. Similarly, the sets $B_G(s_T)$ for those trees T such that s_T is old are mutually disjoint.

By Claim 3.7(i), it is sufficient to connect all vertices in such a tree $T \in \mathcal{T}^l$ to t_T, the mate of s_T. To do this, for any tree $T \in \mathcal{T}^l$, and for each arc $e = (s_{p'}, s_p) \in E(T)$, rent edges from $s_{p'}(G_B)$ to $s_p(G_B)$ in G_B. Furthermore, rent edges from $s_T(G_B)$ to $t_T(G_B)$, thus connecting all demands in T to t_T, and hence to their respective mates. Doing this for all arborescences $T \in \mathcal{T}^l$, and for all values of l, would connect up all the special pairs.

In the next section, we will analyse the cost of all these connections, as well as the cost incurred over all stages of Phase II of the algorithm.

4. ANALYSIS OF THE ALGORITHM

To analyse the algorithm we first show that in stage i, the metrics in G_R and G_B are close to each other. More formally, the distance between two points in G_B is at most a constant times the distance between the corresponding points in G_R plus an additive factor of about Z^i. Thus the length of a path bought or rented in Steps 1, 4, 5, 6 in G_B is only within a constant factor of the length of the corresponding path in G_R. Next we prove that we can account for the rental cost in Steps 1 and 4 to the charge of the components. Accounting for the rental cost for connecting special pairs turns out to be more tricky. Finally, we prove that the dual solution obtained in Step 5 is feasible – this essentially follows from the fact that the only problematic cases were the special pairs, which we declare inactive before this step. Accounting for the cost of the bought edges turns out to be much easier – the dual solution obtained in Step 5 accounts for the weight of the vertices. This weight is used for buying edges. We prove these claims over the next few sections: in Section 4.1 we prove some important facts about distances in the various graphs, after which we prove the above claims in the subsequent three sections.

4.1 Relating Distances in G_R and G_B

Let $G_B^{(i)}, G_R^{(i)}$ be the graphs G_B and G_R at the beginning of stage i respectively.

Theorem 4.1 *The following invariants hold at the beginning of stage i :*

(i) For every vertex v of weight Z^i in $G_R^{(i)}$, the distance of any vertex in $G_B^{(i)}[v]$ from $\mathsf{core}(v)$ is at most $10 \cdot Z^i$.

(ii) If P is a path in $G_R^{(i)}$, then the length of $P[G_B]$ between the core of the end-points of P is at most $2\,\mathsf{length}_{G_R}(P) + 40 \cdot Z^i$.

Proof. Clearly the invariant holds for $i = 0$. Now assume these conditions hold at the beginning of stage i. We now consider stage i. In Step 5, consider a moat $\mathcal{M}$. Suppose it gets contracted to a vertex v. Let $u \in \mathcal{M}$ and $w \in J \cap \mathcal{M}$. Then $d_{G_R}(u, w) \le 4 \cdot Z^{i+1}$ and in fact such a path lies inside the moat $\mathcal{M}$. So induction hypothesis implies that $d_{G_B}(\mathsf{core}(u), \mathsf{core}(v)) \le 9 \cdot Z^{i+1}$ and such a path lies inside $G_B[v]$. So if $u' \in G_B[u]$, then we get that (using induction hypothesis), $d_{G_B}(u', \mathsf{core}(v)) \le 10 \cdot Z^i + 9 \cdot Z^{i+1} \le 10 \cdot Z^{i+1}$. Since any vertex $x \in G_B[v]$ must be belong to $G_B[u]$ for some $u \in \mathcal{M}$, we have shown that invariant (i) holds at the end of step 5. Therefore invariant (i) continues to hold during Step 6 as well because when we create a new node v by merging several nodes, then the core of v contains the core of all the merged vertices.

Now we show that invariant (ii) also holds at the end of stage i. Consider the point of time at the end of Step 6. Let P be a path in G_R – if it does not contain a vertex of weight Z^{i+1}, we are done by induction hypothesis. So assume it contains vertices of weight Z^{i+1} – $x_1, \ldots, x_k$ ordered from left to right. Consider the path P_i between x_i and x_{i+1}. The path $G_B[P_i]$ between the core of x_i and the core of x_{i+1} has length at most $\mathsf{length}(P_i) + \cdot 20 \cdot \sum_{v \in P_i} \mathsf{weight}(v)$ (using invariant (i)). But $\mathsf{length}(P_i) > \delta \cdot \sum_{v \in P_i} \mathsf{weight}(v)$ (otherwise we would have contracted this path in Step 6). So length of $G_B[P_i]$ is at most twice the length of P_i. Let P' be the part of P from x_1 to x_k. We get $\mathsf{length}(G_B[P']) \le 2 \cdot \mathsf{length}(P')$. Let y be the vertex preceding x_1 and u be the left end-point of P. Let P' be the part of P from u to y. Since the highest weight of a vertex in P' is at most Z^i, we can use induction hypothesis on P'. The length of $G_B[P']$ is at most $2 \cdot \mathsf{length}(P') + 40 \cdot Z^i$. Further the distance between the core of y and the core of x_1 (using (i)) is at most $10 \cdot Z^i + 10 \cdot Z^{i+1} + l_e =\le 11 \cdot Z^{i+1} + c_e$, where e is the egde joining y and x_1. Arguing similarly on the part of P to the right of x_k, we see that the length of $G_B[P]$ is at most $2 \cdot \mathsf{length}(P) + 40 \cdot Z^{i+1}$. This proves the theorem. □

4.2 Paying for the Rental Cost

Lemma 4.2 *Suppose v is vertex of weight Z^i in G_R at the beginning of stage i such that $G[v]$ contains $S(j)$ for some active demand j of class $l < i$. Then the end of stage i, there is a vertex x of weight Z^{i+1} containing v, such that the core of x contains that of v. More formally, $G[v] \subseteq G[x]$ and $G[\mathsf{core}(v)] \subseteq G[\mathsf{core}(x)]$.*

Proof. The vertex v belongs to the set J_1 defined in Step 3, and hence we grow a moat around v. At the end of Step 5, we contract this moat to a vertex x of weight Z^{i+1}

such that the core of w contains that of v. In Step 6, we can only merge x into a new vertex of weight Z^{i+1}, whose core again contains that of x. This proves the lemma. □

Lemma 4.3 *Consider demand $j \in D_k$ of class $l < i$ which is active at the beginning of stage i. Then there exists a vertex $v \in G_R^{(i)}$ of weight Z^i such that $G[v]$ contains $S(j)$ and $j(G_B)$ is connected to $\mathsf{core}(v)$ in G_B by rented edges.*

PROOF. For the demand j of class l, let us first consider stage l. If j is active at the end of Step 2 of this stage, then j belongs to the set $\mathbf{A}_l$ as defined in Step 3. Suppose $j \in \mathbf{A}_l^k(C)$, for some $C \in \mathcal{C}(k,0)$. Then Step 4 implies that we can find a demand $j' \in \mathbf{A}_l^k(C)'$ such that $d_{G_R}(j(G_R), j'(G_R)) \le Z^{l+1}$, such that Step 4 connects j' to a vertex $v_{j'} \in J$. Hence, in the graph G_B, $j(G_B)$ and $\mathsf{core}(v_{j'})$ belong to the same component of $\mathcal{C}(k)$.

In Step 5 of the algorithm, we grow a moat around $v_{j'}$ for $4 \cdot Z^{l+1}$ units of time. We claim that this moat will contain all the vertices $u(G_R)$ for $u \in S(j)$. Indeed, $d_{G_R}(v_{j'}, u(G_R)) \le d_{G_R}(v_{j'}, j'(G_R)) + d_{G_R}(j(G_R), j'(G_R)) + d_{G_R}(j(G_R), u(G_R)) \le Z^{l+1} + Z^{l+1} + Z^l$, the first two bounds of Z^{l+1} from the definition of J and of $\mathbf{A}_k^l(C)'$, and the last bound of Z^l follows from the definition of $S(j)$ and the definition of *class*. Hence the distance $d_{G_R}(v_{j'}, u(G_R)) < 3Z^{l+1}$, and we grow the moat for $4Z^{l+1}$ units of time; hence the moat contain all these vertices. At the end of Step 5, this moat contracts to a vertex v whose core contains $\mathsf{core}(v_{j'})$, and hence the lemma holds at the beginning of stage $l+1$. This proves the lemma for stage $i = l+1$; Lemma 4.2 implies that the invariant is maintained for subsequent steps as long as j is active, which proves the lemma. □

Lemma 4.3 also shows that eventually all demands will become inactive. Indeed, as long as a demand is active, the set J_1 (and hence J) constructed in Step 3 will be non-empty, and we will make progress.

Claim 4.4 *Consider a $C \in \mathcal{C}(k)$ such that C contains a demand of type l, then the charge of C is at least Z^l.*

PROOF. The claim follows by induction on the number of steps in the algorithm. It is true at the beginning of the algorithm (by definition of the charge of a component). Suppose it is true at the beginning of some step t. If we merge two components at this time, the charge of the new component is equal to the maximum of the two components. So this invariant holds at the end of this step as well. □

We now show how we can pay for the rental cost in stage i.

- **Step 1 connections:** If we connect C_1 and C_2, Theorem 4.1 implies the rental cost is $O(Z^{l_1})$. hence the charge for C_1 can pay for this connection, leaving the C_2's charge for the resulting component.
- **Step 4 connections:** Fix a demand group D_k and component $C \in \mathcal{C}(k,0)$. Consider the demands in $\mathbf{A}_i^k(C)'$: the cost paid for connecting each such demand in the graph G_B to the nearest vertex in J is $O(Z^{i+1})$ (again using Theorem 4.1).

  How can we account for this cost? We can take Z^i from the charge of C (since, by the definition of $\mathbf{A}_i$, the component C contains a demand of class i); since different stages would take exponentially increasing amounts from C, this would be fine. However, $\mathbf{A}_i^k(C)'$ may contain more than a constant number of demands, each costing $O(Z^{i+1})$. Let $E_C(i)$ be the edges of C which are "uncontracted" at the beginning of stage i in the graph G_R. For each $j \in \mathbf{A}_i^k(C)'$ consider a ball $B(j)$ of radius $Z^{l+1}/2$ around $j(G_R)$. By the fact that all $j \in \mathbf{A}_i^k(C)$ are well-separated, these balls are disjoint. Now, the total cost of the edges in each ball $B(j) \cap E_C(i)$ is at least $Z^{l+1}/2$, since $E_C(i)$ has an edge incident to $j(G_R)$, it has an edge incident to a vertex $j' \in \mathbf{A}_i^k(C)'$ outside the ball $B(j)$ and it is a connected set of edges. So we charge the rental cost paid by j to the edges of $E_C(i)$ in $B(j)$. Note that we never charge to these edges again in later stages, because as in the proof of Lemma 4.3, the edges in any such set $E_C(i)$ will be covered by a single moat around some vertex $v \in J$, contracted to a single vertex, and never be charged again.

This accounts for all rental costs except those incurred to connect the special demand pairs (put aside in Step 2 of each stage and connected at the end of the process): these we consider in the following section.

4.3 Rental Cost II: Special Pairs Unit

Lemma 4.5 *If (s_p, t_p) is a special pair, then $\mathsf{class}(s_p) < \mathsf{class}(t_p)$ (i.e., they are not equal).*

PROOF. For a contradiction, let $\mathsf{class}(s_p) = \mathsf{class}(t_p) = l$. Since this is a special pair, it must have been declared inactive during Step 2(ii) of stage l. Hence the ball B around $s_p(G_R)$ of radius $10 \cdot Z^{l+1}$ (as in Step 2(ii)) must satisfy $A(B, t_p) \ge 0.7$. Since $A(S(t_p), t_p) \ge 0.4$, the set $B_G = \cup_{v \in B} G[v]$ must contain a vertex of $S(t_p)$. But, since $\mathsf{class}(t_p) = l$, the 0.4-radius of t_p is at most Z^l. By the triangle inequality, $d_{G_R}(s_p(G_R), t_p(G_R)) \le 11 \cdot Z^{l+1}$.

Now since the type of a demand is at least its class, the types of both s_p, t_p are at least l. So if s_p and t_p lie in components C_1 and C_2 respectively, Claim 4.4 implies that the charge of both these components is at least Z^l, and hence these components would have been merged in Step 1 of stage l, if not earlier. But then we would have used Step 2(i) for this pair, contradicting the fact that (s_p, t_p) is a special pair. □

Recall that $\mathsf{class}(t_p) = \max\{\mathsf{class}(s_p), l\}$ where l is such that $r_{0.4}(t_p) \in [Z^{l-1}, Z^l]$. Lemma 4.5 implies that if (s_p, t_p) is special, then $\mathsf{class}(t_p) = l$.

Fix a demand group D_k for the rest of the section. Consider the set $\mathcal{S}^{(l)}$ as defined in Section 3.4.2. Recall that for two pairs (s_p, t_p) and $(s_{p'}, t_{p'}) \in \mathcal{S}^{(l)}$, if $t_p, t_{p'}$ share a component of $\mathcal{C}(k, f)$, then $s_p, s_{p'}$ do not. Also, observe that by Lemma 4.5, if $(s_p, t_p) \in \mathcal{S}^{(l)}$, then $\mathsf{class}(t_p) = l$, and $\mathsf{class}(s_p) < l$.

Claim 4.6 *Suppose $s_p \in \mathcal{S}^{(l,i)}, s_{p'} \in \mathcal{S}^{(l,i')}$ for $i \le i' \le l$, and $B_G(s_p) \cap B_G(s_{p'}) \neq \emptyset$. At the end of stage l, t_p and $t_{p'}$ are in the same component of $\mathcal{C}(k)$. In particular, if $(s_{p'}, s_p) \in E^l$, then t_p and $t_{p'}$ are in the same component of $\mathcal{C}(k)$.*

PROOF. Let $v \in B_G(s_p) \cap B_G(s_{p'})$. By the definition of $B_G(s_p)$, the distance between $v(G_R)$ and $s_p(G_R)$ in stage i is at most $10 \cdot Z^{i+1}$, and $A(B_G(s_p), t_p) \geq 0.7$. Hence $B_G(s_p) \cap S(t_p) \neq \emptyset$; let w lie in this intersection. Now since $w \in S(t_p)$, the distance $d_{G_R}(w(G_R), t_p(G_R)) \leq Z^l$. By the triangle inequality, $d_{G_R}(t_p(G_R), v(G_R)) \leq d_{G_R}(t_p(G_R), w(G_R)) + d_{G_R}(w(G_R), s_p(G_R)) + d_{G_R}(s_p(G_R), v(G_R)) \leq Z^l + 10Z^{l+1} + 10Z^{l+1} \leq 21Z^{l+1}$. Subsequently, the distance between $t_p(G_R)$ and $v(G_R)$ can only decrease.

A similar argument shows that $d_{G_R}(t_{p'}(G_R), v(G_R)) \leq 21 \cdot Z^{l+1}$ at the beginning of stage i', and $d_{G_R}(t_p(G_R), t_{p'}(G_R) \leq 42 \cdot Z^{l+1} < Z^{l+2}$. Now since each of $t_p, t_{p'}$ have type at least l, the components containing them have charge at least Z^l (by Claim 4.4), Step 1 of stage i' would connect their components. Hence, for at the end of any stage $l \geq i'$, $t_p, t_{p'}$ share the same component. □

Claim 4.7 *If $s_p, s_{p'} \in \mathcal{S}^{(l,i)}$ both have directed paths in G^l to a common vertex s_r, then s_p and $s_{p'}$ must lie in the same component of $\mathcal{C}(k, f)$.*

PROOF. Take such a path $P = \{s_p = s_1, s_2, \ldots, s_h = s_r\}$, such that $s_g \in \mathcal{S}^{(l,i_g)}$ for all $g \in [1, h]$. By the construction of the arc set E^l, it follows that $i_1 > i_2 > \cdots > i_h$, and that these vertices are either all new, or all old. As in the proof of Claim 4.6, the distance in G_R between $s_g(G_R)$ and $s_{g+1}(G_R)$ at the beginning of stage i is $O(Z^{i_g})$. The resulting geometric sum implies the distance between $s_p(G_R)$ and $s_r(G_R)$ is $O(Z^i)$. An identical argument holds for $s_{p'}$, and so the distance $d_{G_R}(s_p(G_R), s_{p'}(G_R)) = O(Z^i)$ by the triangle inequality.

Now two cases arise—s_p and $s_{p'}$ are either both new, or both old. If they are both new, then both have class $= i$, and hence their components have charge at least Z^i; thus Step 1 of stage i would ensured they lie in the same component. In the other case, if both are old, then Lemma 4.3 implies that at the beginning of stage i, vertices $s_p(G_R), s_{p'}(G_R)$ have weight Z^i; since the previous paragraph bounded their distance by $O(Z^i)$, they would have been connected by edges bought in Step 6 of stage $i-1$. Finally, Lemma 4.3 also implies that $\mathsf{core}(s_p(G_R))$ and s_p are connected in G_B by rented edges, and the same holds true for $\mathsf{core}(s_{p'}(G_R))$ and $s_{p'}$. Putting this all together, s_p and $s_{p'}$ lie in the same component of $\mathcal{C}(k, i)$. □

We can now give the proof of Claim 3.7, which relies on the machinery developed in this section.

Proof of Claim 3.7. To construct the set $\mathcal{T}^l$, let J^l denote the set of sink vertices in the digraph G^l. We find node-disjoint in-arborescences rooted at the vertices of J^l such that each vertex in V^l appears in exactly one such in-arborescence. This is the set $\mathcal{T}^l$. For any $T \in \mathcal{T}^l$, Claim 4.6 implies that the set $\{t_p \mid s_p \in T\}$ is contained in a single component of $\mathcal{C}(k, f)$.

To prove part (ii) of the claim, let J^l_o be the old demands in J^l, and J^l_n be the new ones. We claim the sets $B_G(s_p)$ for $s_p \in J^l_o$ are disjoint; an identical argument holds for sets corresponding to $s_p \in J^l_n$. For $s_p, s_{p'} \in J^l_o$, let $s_p \in \mathcal{S}^{(l,i)}, s_{p'} \in \mathcal{S}^{(l,i')}$. If $i' \neq i$, the sets $B_G(s_p), B_G(s_{p'})$ are disjoint else we would have added an arc between them, contradicting the fact that they are both sinks. The other possibility is that $i' = i$: suppose the two sets intersect. Let $v \in B_G(s_p) \cap B_G(s_{p'})$. Then at the beginning of stage i, $d_{G_R}(v(G_R), s_p(G_R)), d_{G_R}(v(G_R), s_{p'}(G_R)) \leq 10 \cdot Z^{i+1}$. So $d_{G_R}(s_p(G_R), s_{p'}(G_R)) \leq 20 \cdot Z^{i+1}$. Therefore the demands s_p and $s_{p'}$ must lie in the same component of $\mathcal{C}(k)$ at the end of step 1 of this stage. This is true for $t_p, t_{p'}$ also (by Claim 4.6); but this gives us a contradiction to the construction of $\mathcal{S}$. □

Finally, we bound the cost of connecting all the special vertices.

Lemma 4.8 *For any tree $T \in \mathcal{T}^l$, the total rental cost of connecting the demands in T and then s_T to r_T is $O(Z^l)$. Also, the total cost of connecting the special pairs is $O(\mathsf{OPT})$.*

PROOF. First observe that for any $i \leq l$, the tree T contains at most one demand from $\mathcal{S}^{(l,i)}$; if there were two such demands $s_p, s_{p'}$, then Claim 4.7 would imply that they are in the same component of $\mathcal{C}(k, f)$. Moreover, Claim 4.6 implies that $t_p, t_{p'}$ also share a component of $\mathcal{C}(k, f)$—but then these pairs would be equivalent and could not both belong to $\mathcal{S}$.

Consider a single arc $(s_{p'}, s_p)$ of T, with $s_{p'} \in \mathcal{S}^{(l,i')}, s_p \in \mathcal{S}^{(l,i)}$ and $i' > i$. As in Claim 4.6, the distance in G_R at the end of stage i' between $s_p(G_R)$ and $s_{p'}(G_R)$ is $O(Z^{i'})$, and hence the rental cost in G_B corresponding to this arc is $O(Z^{i'})$, by yet another invocation of Theorem 4.1. Since at most one demand in T can belong to any particular set $\mathcal{S}^{(l,i)}$, and since $i \leq l$ for all the demands, adding the cost over all the arcs in T gives us a geometric sum which is at most $O(Z^l)$. Finally, the distance between the root s_T and its mate t_T in G_R is at most $O(Z^l)$ using arguments similar to those in the proof of Claim 4.6, and hence the cost of connecting them in G_B is again at most $O(Z^l)$ by an application of Theorem 4.1.

Now summing over all trees in $\mathcal{T}^l$, the total cost incurred in connecting all the special pairs in $\mathcal{S}^l$ is $O(|J^l| \cdot Z^l)$, and the total cost is $\sum_l O(|J^l| \cdot Z^l)$. We would like to use Theorem 3.3 to bound this quantity by OPT. To do that, note that Claim 3.7(ii) implies that for $s_p, s_{p'} \in J^l_o$, we have $B_G(s_p) \cap B_G(s_{p'}) = \emptyset$. But $A(B_G(s_p), t_p) \geq 0.7$ by the construction of the special pairs, and hence the demands in $\overline{J}^l_n = \{t_p \mid s_p \in J^l_n\}$ are 0.7-disjoint. Similarly, we can argue that the demands in $\overline{J}^l_o = \{t_p \mid s_p \in J^l_o\}$ are 0.7-disjoint. Now applying Theorem 3.3 (with $\alpha = 0.4, \beta = 0.7$) to $\overline{J}^l_o$ and $\overline{J}^l_n$ now implies that $\sum_l |\overline{J}^l_o| \cdot Z^l = \sum_l |J^l_o| \cdot Z^l = O(\mathsf{OPT})$ and $\sum_l |\overline{J}^l_n| \cdot Z^l = \sum_l |J^l_n| \cdot Z^l = O(\mathsf{OPT})$, completing the proof. □

4.4 Dual Feasibility

We now turn to the moat-growing process being executed on G_R, and the corresponding dual-raising process that is carried out in Steps 4-5 of the algorithm; we have to show we generate a feasible dual (which is a valid lower bound on the optimal cost). When we raise y_S in Step 5, we just have to ensure that S is valid for some demand. The moat growing procedure will make sure that we always satisfy the constraints (3.7).

Lemma 4.9 *Consider the set J constructed in Step 3 of stage i. For any $v, w \in J$, $d_{G_R}(v, w) \geq Z^{i+1}$.*

PROOF. Let $v, w \in J_1$ in the graph $G_R^{(i)}$. For contradiction, suppose the distance between them in $G_R^{(i)}$ is less than Z^{i+1}. Let P be a shortest path between these two vertices, and v', w' be two consecutive vertices of weight Z^i on this path—there must exist two such vertices, since both the endpoints v, w have weight Z^i. The length of the subpath P' between u', v' is less than $Z^{i+1} \leq 2\gamma Z^i \leq \gamma \sum_{x \in \widehat{P}} \mathsf{weight}(x)$, and hence we would have contracted this subpath P' in Step 6 of the previous stage (we are assuming that γ is at least $2 \cdot Z$). This ensures the desired lower bound on the distance between nodes in J_1; the vertices in $J_2 = J \setminus J_1$ have at least this distance by construction. □

Consider a vertex $v \in J$ at the beginning of Step 5 in stage i. We define the *active demand j associated with v* as follows: If $v \in J_1$, then by definition of J_1, there exists an active demand j with $\mathsf{class}(j) < i$ such that $S(j) \subseteq G[v]$ (and j can reach $\mathsf{core}(v)$ through rented edges in G_B)—we associate this demand j with v. If $v \in J_2$, then $v = j(G_R)$ for some $j \in \mathbf{A}'_i$—we associate this demand j with v.

Recall that we grow a most $\mathcal{M}$ around v in Step 5, and also raise y_S during the period $[Z^i, Z^{i+1}/4]$, where $S = \cup_{w \in M} G[w]$. To prove that S is a valid cut for *some* demand, we show this for the demand j associated with v.

Theorem 4.10 *The cut S remains valid for j during the time interval $[Z^i, Z^{i+1}/4]$.*

PROOF. If $v \in J_1$, then the very criterion for inclusion into J_1 ensures that $S(j) \subseteq G[v]$; hence $S(j) \subseteq S$ during this process. If $v \in J_2$, then $v = j(G_R)$ for some class-i demand $j \in \mathbf{A}'_i$. Note that $\mathsf{class}(j) = i$ implies that $S(j)$ has radius at most Z^i in G, and hence after time Z^i the set S will contain all the vertices in $S(j)$. So in either case, the set S contains $S(j)$ during this time interval. We now need to show that S does not get too much assignment from j's mate. Since some of our definitions are asymmetric, we consider two cases, depending on whether $j = s_p$ or t_p.

Case I: Suppose $j = s_p$. In this case, recall that the set S remains valid if $A(S, t_p) \leq 0.7$ over the relevant time period. We know that $\mathsf{class}(s_p) \leq i$; the first subcase is when $\mathsf{class}(t_p) < i$. Lemma 4.3 argues that there is a vertex w of weight Z^i such that $G[w]$ contains $S(t_p)$, and so $w \in J$ as well. Note that $w \neq v$, else both s_p, t_p would lie in the same component of $\mathcal{C}(k, i)$, and cannot be active. By Lemma 4.9 the G_R-distance between v and w is at least Z^{i+1}, and so the moat $\mathcal{M}$ containing s_p cannot capture w in the time interval $[Z^i, Z^{i+1}/4]$. Since $G[w]$ contains $S(t_p)$, the assignment $A(\mathcal{M}, t_p)$ is at least $1 - A(S(t_p), t_p) \leq 0.6$, and hence S remains valid for s_p during the entire time period.

Now consider the other subcase when $\mathsf{class}(t_p) \geq i$. Let B' be the ball of radius Z^{i+1} around $s_p(G_R)$. We know that $A(B', t_p) < 0.7$, otherwise we would have declared this demand inactive (and special) in Step 2(ii). So the set S remains valid for s_p during this time period.

Case II: Now for the case $j = t_p$; recall now that the set S remains valid if $A(S, s_p) \leq 0.3$ over the relevant time period. If $\mathsf{class}(t_p) < i$, the argument is identical to the one above for the case when $\mathsf{class}(s_p) \leq i, \mathsf{class}(t_p) < i$. The other case is when $class(t_p) = i$; note that $\mathsf{class}(s_p) \leq \mathsf{class}(t_p) = i$. In this case $t_p(G_R) \in J_2$, and the distance in G_R between $s_p(G_R)$ and $t_p(G_R)$ is at least Z^{i+1}, otherwise we would not have added t_p to $\mathbf{A}'_i$ (if $\mathsf{class}(s_p) < i$), or we would have merged the components of s_p, t_p together in Step 1 (if $\mathsf{class}(s_p) = i$). Now consider some $x \in S(s_p)$, the distance in G_R between $s_p(G_R)$ and $x(G_R)$ is at most Z^i, since $\mathsf{class}(s_p) \leq i$. We claim that $x(G_R) \notin S$ during the time interval $[Z^i, Z^{i+1}]$; indeed, that would make the G_R-distance between $s_p(G_R)$ and $t_p(G_R)$ would be $Z^{i+1}/4 + Z^i < Z^{i+1}$. Hence S and $S(s_p)$ are disjoint for this time interval, and hence $A(S, s_p) \leq 1 - 0.8 = 0.2 < 0.3$, and hence S remains valid for t_p. □

The above theorem implies that the dual solution for (DP2) constructed during Step 2 (over all the stages) is a feasible solution, and hence gives us a lower bound for OPT.

4.5 Paying for the Buying Cost

Since Sections 4.2 and 4.3 have already accounted for the edges *rented* by the algorithm, it now suffices to show that the edges *bought* in Steps 5 and 6 of the algorithm can be paid for.

The weight of a vertex is accounted for by the dual values raised during Step 5. Consider a moat $\mathcal{M}$ formed at the end of Step 5 in stage i, and suppose it contains m vertices from J—call this set J'. These vertices in J' raised a total dual value of $\Omega(m \cdot Z^{i+1})$. Moreover, the total edges bought inside the moat $\mathcal{M}$ is proportional to $(m-1) \cdot Z^{i+1}$, since when two moats meet, the distance between their centers is at most $2 \cdot 4 \cdot Z^{i+1}$, and Theorem 4.1 ensures that length of all the edges bought in G_B is also proportional to Z^{i+1}. Hence, out of the dual value raised, $O((m-1) \cdot Z^{i+1})$ goes towards accounting for the edges bought and $\Omega(Z^{i+1})$ goes towards accounting for the weight of the new vertex obtained by contracting $\mathcal{M}$.

Finally, to account for the edges bought in Step 6, we again use Theorem 4.1 to infer that the total cost of edges bought in $P[G_B]$ can be paid by the weights of all the vertices of P *except one of the end-points*. The weight of this remaining vertex gets transfered to the new vertex formed by collapsing this bought path, and hence we can pay for the bought edges in Step 6.

References

[1] A. Agrawal, P. Klein, 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] M. Andrews. Hardness of buy-at-bulk network design. In *45th FOCS*, 115–124. 2004.

[3] B. Awerbuch, Y. Azar. Buy-at-bulk network design. In *38th FOCS*, 542–547. 1997.

[4] M. Charikar, C. Chekuri, M. Pál. Sampling bounds for stochastic optimization. In *8th APPROX*, *LNCS*, vol. 3624, 257–269. 2005.

[5] C. Chekuri, M. Hajiaghayi, G. Kortsarz, M. R. Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design problems. In *47th FOCS*, 677–686. 2006.

[6] C. Chekuri, N. Korula, M. Päl. Improved algorithms for orienteering and related problems. In *19th SODA*, 661–670. 2008.

[7] S. Dye, L. Stougie, A. Tomasgard. The stochastic single resource service-provision problem. *Nav. Res. Logist.*, 50(8):869–887, 2003.

[8] F. Eisenbrand, F. Grandoni, T. Rothvoß, G. Schäfer. Approximating connected facility location problems via random facility sampling and core detouring. In *19th SODA*, 1174–1183. 2008.

[9] L. Fleischer, J. Könemann, S. Leonardi, G. Schäfer. Simple cost sharing schemes for multicommodity rent-or-buy and stochastic steiner tree. In *38th STOC*, 663–670. ACM Press, New York, NY, USA, 2006.

[10] N. Garg, R. Khandekar, G. Konjevod, R. Ravi, F. S. Salman, A. Sinha. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design formulation. In *8th IPCO*, *LNCS*, vol. 2081, 170–184. 2001.

[11] M. X. Goemans, 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).

[12] S. Guha, A. Meyerson, K. Mungala. A constant factor approximation for the single sink edge installation problems. In *33rd STOC*, 383–388. 2001.

[13] A. Gupta, M. Hajiaghayi, A. Kumar. Stochastic Steiner tree with non-uniform inflation. In *10th APPROX*, 134–148. 2007.

[14] A. Gupta, A. Kumar, M. Pál, T. Roughgarden. Approximation via cost sharing: simpler and better approximation algorithms for network design. *J. ACM*, 54(3):Art. 11, 38 pp., 2007.

[15] A. Gupta, M. Pál. Stochastic Steiner trees without a root. In *32nd ICALP*, *LNCS*, vol. 3580, 1051–1063. 2005.

[16] A. Gupta, M. Pál, R. Ravi, A. Sinha. Boosted sampling: Approximation algorithms for stochastic optimization problems. In *36th STOC*, 417–426. 2004.

[17] —. What about Wednesday? approximation algorithms for multistage stochastic optimization. In *8th APPROX*, *LNCS*, vol. 3624, 86–98. 2005.

[18] A. Gupta, R. Ravi, A. Sinha. LP Rounding Approximation Algorithms for Stochastic Network Design. *Mathematics of Operations Research*, 32(2):345–364, 2007.

[19] A. Hayrapetyan, C. Swamy, E. Tardos. Network design for information networks. In *16th SODA*, 933–942. 2005.

[20] N. Immorlica, D. Karger, M. Minkoff, V. Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *15th SODA*, 684–693. 2004.

[21] A. Kumar, A. Gupta, T. Roughgarden. A constant-factor approximation algorithm for the multicommodity rent-or-buy problem. In *43rd FOCS*, 333–342. 2002.

[22] R. Ravi, A. Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. In *10th IPCO*, 101–115. 2004.

[23] D. B. Shmoys, C. Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM*, 53(6):978–1012, 2006.

[24] D. P. Williamson, A. van Zuylen. A simpler and better derandomization of an approximation algorithm for single-source rent-or-buy. *Operations Research Letters*, 35(6):707–712, 2007.