Assignment Problem in Content Distribution Networks: Unsplittable Hard-capacitated Facility Location[*]

MohammadHossein Bateni[†‡]   MohammadTaghi Hajiaghayi[§]

Abstract

In a Content Distribution Network (CDN), there are m servers storing the data; each of them has a specific bandwidth. All the requests from a particular client should be assigned to one server, because of the routing protocol used. The goal is to minimize the total cost of these assignments —cost of each is proportional to the distance as well as the request size— while the load on each server is kept below its bandwidth limit. When each server also has a setup cost, this is an *unsplittable hard-capacitated facility location problem*. As much attention as facility location problems have received, there has been no nontrivial approximation algorithm when we have hard capacities (i.e., there can only be one copy of each facility whose capacity cannot be violated) and demands are unsplittable (i.e., all the demand from a client has to be assigned to a single facility). We observe it is NP-hard to approximate the cost to within any bounded factor. Thus, for an arbitrary constant $\epsilon > 0$, we relax the capacities to a $1 + \epsilon$ factor. For the case where capacities are almost uniform, we give a bicriteria $O(\log n, 1+\epsilon)$-approximation algorithm for general metrics and a $(1 + \epsilon, 1 + \epsilon)$-approximation algorithm for tree metrics. A bicriteria (α, β)-approximation algorithm produces a solution of cost at most α times the optimum, while violating the capacities by no more than a β factor. We can get the same guarantee for non-uniform capacities if we allow quasipolynomial running time. In our algorithm, some clients guess the facility they are assigned to, and facilities decide the size of the clients they serve. A straight-forward approach results in exponential running time. When costs do not satisfy metricity, we show that a 1.5 violation of capacities is necessary to obtain any approximation.

It is worth noting that our results generalize bin packing (zero cost matrix and facility costs equal to one), knapsack (single facility with all costs being zero), minimum makespan scheduling for related machines (all costs being zero) and some facility location problems.

Key words: approximation algorithm, PTAS, network, facility location

1   Introduction

Content Distribution Networks (CDNs) have emerged in the last decade in order to distribute media content on behalf of content owners efficiently and cost-effectively. Such networks are especially motivated due to the often bursty nature of demands and the fact that content owners require their contents with good quality to be highly available in a timely manner. A single Autonomous System (AS) with a large footprint in the country or region, e.g., the US Tier-1 ISPs, providing CDN services often has a load-aware CDN architecture; thus load balancing is very desirable. In this context, the following problem formulation for minimizing network cost has been considered recently by Alzoubi, Lee, Rabinovich, Spatscheck, and Van der Merwe [ALR+08].

Assume an AS has m servers, where each server i can serve up to S_i requests per time unit. A request enters the system through one of n ingress Provider Edge (PE) routers. Each ingress PE j contributes d_j amount of request per time unit all of which should be served by one server due to underlying IP Anycast

---

routing architecture. There is a *connection cost* matrix c_{ij} for serving PE j at server i per unit of demand. Since c_{ij} is typically proportional to the distance between server i and PE j within the AS, connection costs often satisfy metricity. We further observe that, in practice, the bandwidth constraints on servers are almost the same; i.e., the ratio of the largest to the smallest is less than a small constant.

Alzoubi et al. notices that this problem of assigning demands to servers unsplittably while minimizing the overall connection cost is a special case of the *Generalized Assignment Problem (GAP)* [RS75]. By rounding a natural LP relaxation of this problem, Shmoys and Tardos [ST93] show that in polynomial time, given a value C, we can either decide no feasible solution of cost C exists, or else find a schedule of cost at most C where the load on each server is at most $S_i + \max_j d_j$. Though the overload amount is bounded by $\max_j d_j$, according to Alzoubi et al. [ALR$^+$08], in practice, the overload volume can be significant since a single PE can contribute a large request load like 20% of a server capacity. Indeed, server capacities are very crucial constraints whose violations can disrupt a large number of connections and should be avoided even though we pay a bit more in total connections cost. This is the problem that we address in this paper by presenting the first algorithm which violates the capacities by at most a $1 + \epsilon$ factor, for an arbitrary constant $\epsilon > 0$, while paying a total connection cost of at most $O(\log n)$ times the optimum. Note that even if all connection costs are zero, just the assignment problem is the NP-hard *bin packing problem*, which needs this $1+\epsilon$ factor capacity inflation to be solved. Thus, in terms of violating capacities, this is the strongest possible type of result for this problem. If connection costs do not satisfy metricity, there is no approximation algorithm in terms of cost which can violate capacities by a factor less than 1.5 (see Theorem 2).

Interestingly, by adding setup costs for servers (a.k.a. *facility costs*) in the problem formulation above, we have the *unsplittable hard-capacitated facility location* problem which is considered first in this paper when we allow violating facility capacities by only a $1 + \epsilon$ factor. So far, all previous approximation algorithms for hard capacities have focused on the splittable demand case to the best of our knowledge.

1.1 Related results

The *generalized assignment problem (GAP)* can be viewed as a scheduling problem on parallel machines, where each machine has a capacity (or maximum load) and each job has a size (or processing time) and a cost, each possibly dependent on the machine to which it is assigned, and the objective is to minimize the total cost incurred. The problem is NP-hard [FJVW86] and has been a central problem in operations research since 1975 [RS75]. Shmoys and Tardos [ST93] showed that in polynomial time, given a value C, we can either decide that no feasible solution of cost C exists, or else find a schedule of cost at most C where the load on each machine is at most twice that of the optimum. Indeed, by scaling the processing times, without loss of generality, we can assume that all machine capacities are equal to T. Then GAP asks for a scheduling of minimum cost with *makespan*, the maximum machine load, at most T. The problem of just minimizing makespan (without any cost involved) is also a very important problem for which Lenstra, Shmoys and Tardos [LST90] gave the currently best approximation algorithm with factor two by rounding a natural LP relaxation of the problem (Shmoys and Tardos [ST93] based their algorithm on this result.) When each job has the same size on all machines, Hochbaum and Shmoys [HS87, HS88] obtained PTASs for the makespan problem. We are generalizing this result in this paper when jobs also have costs. In fact, our result is also a generalization of the well-known bin packing problem, in which given n pieces of size $0 < p_j \leq 1$, the objective is to pack the pieces into the minimum number of bins where the sum of the sizes of the pieces packed in each bin cannot exceed one. In this case, all facility costs are one and all connection costs are zero. There are also maximization versions of GAP, studied by Chekuri and Khanna [CK00] and Fleischer et al. [FGMS06].

The facility location problem is another central problem in operations research. It is also well studied in the field of approximation algorithms, and a number of different approximation algorithms have been proposed for this problem using a variety of techniques (see [Shm00] for a survey). The first constant-factor approximation algorithm for the uncapacitated version of this problem was given by Shmoys, Tardos, and Aardal [STA97] and was based on LP rounding and a filtering technique due to Lin and Vitter [LV92]. Later, the factor was improved by Chudak and Shmoys [Chu98, CS98] to $1 + 2/e$. Jain and Vazirani [JV01] gave a primal-dual algorithm for this problem, achieving a factor of 3. Strategies based on local search

and greedy improvement for facility location problems have also been studied. The work of Korupolu et al. [KPR00] shows that a simple local search heuristic proposed by Kuehn and Hamburger [KH63] yields a $(5 + \epsilon)$-approximation algorithm for any $\epsilon > 0$. This was later improved by Arya et al. [AGK$^+$01] to 3. Charikar and Guha [CG99] improved the factor of the LP-based algorithm [Chu98] slightly to 1.728 by combining the primal-dual algorithm [JV01] with greedy augmentation. Mahdian et al. [MMSV01] used a variant of the primal-dual schema, called the dual-fitting method, to give a combinatorial algorithm with an approximation ratio of 1.861. This was later improved by Jain, Mahdian and Saberi [JMS02] and Mahdian, Ye and Zhang [MYZ02] to 1.61 and 1.52, respectively. The best approximation factor for this problem is currently 1.5 due to Byrka [Byr07] by modifying the algorithm of Chudak and Shmoys [Chu98, CS98] and combining it with that of Mahdian et al. [MYZ02]. Guha and Khuller [GK99] showed that this problem is hard to approximate within a factor better than 1.463, assuming $NP \nsubseteq DTIME[n^{O(\log \log n)}]$.

Capacitated facility location has also received a great deal of attention in recent years. Two main variants of the problem are soft-capacitated facility location and hard-capacitated facility location: in the latter problem, each facility is either opened at some location or not, whereas in the former, one may specify any integer number of facilities to be opened at that location. Soft capacities make the problem easier and by modifying approximation algorithms for the uncapacitated problems, we can also handle this case [STA97, JV01]. All previous approximation algorithms for hard capacities have focused on the uniform demand case or the splittable case in which each unit of demand can be served by a different facility. Korupolu, Plaxton, and Rajaraman [KPR00] gave the first constant approximation algorithm that handles hard capacities, based on a local search procedure, but their approach worked only if all capacities are equal. Chudak and Williamson [CW05] improved this performance guarantee to 5.83 for the same uniform capacity case. Pál, Tardos, and Wexler [PTW01] gave the first constant performance guarantee for the case of non-uniform hard capacities. This was recently improved by Mahdian and Pál [MP03] and Zhang, Chen, and Ye [ZCY04] to yield a 5.83-approximation algorithm. All these approaches are based on local search. The only LP-relaxation based approach for this problem is due to Levi, Shmoys and Swamy [LSS04] who gave a 5-approximation algorithm for the special case in which all facility opening costs are equal (otherwise the LP does not have any constant integrality ratio). As aforementioned, this paper for the first time considers the unsplittable hard-capacitated facility location problem when we allow violating facility capacities by a $1 + \epsilon$ factor (otherwise, it is NP-hard to obtain any approximation factor).

1.2 Contributions

The problem we consider is formally defined as follows.

Definition 1. (UNSPLITTABLE HARD-CAPACITATED METRIC FACILITY LOCATION PROBLEM (UHCMFL))
Given is a weighted graph G. Facilities A and clients B are located (possibly collocated) on the vertices of G. Each client i has a demand d_i which is to be assigned to a single facility. If assigned to facility j, the client needs to pay a connection cost equal to d_i times the length of their shortest path c_{ij}). Besides, the facility needs to be initialized with a one-time pay cost of f_j. Each facility also has a capacity S_j which limits the total sum of demands assigned to it. We seek a set of facilities $A^* \subseteq A$ to open, and a function $\pi : B \mapsto A^*$, such that the cost $\sum_{j \in A^*} f_j + \sum_{i \in B} d_i c_{i,\pi(i)}$ is minimized, while for each facility j, we have $\sum_{\pi(i)=j} d_i \leq S_j$. We let n be the number of nodes (clients and facilities) in the instance.

Theorem 1. *UHCMFL problem does not admit any approximation algorithms unless $P = NP$.*

The proof is via a simple reduction from the knapsack problem: we just need one facility without any connection or facility setup costs (i.e., they are all zero). One the other hand, if we do not have connection costs (all zero) but facility costs are set to one, the case of uniform capacities is simply the bin packing problem. This means a $1 + \epsilon$ relaxation of capacities is necessary.

With connection and facility costs being zero, we obtain the minimum makespan scheduling of related machines (studied by Hochbaum and Shmoys [HS87, HS88]), for which a PTAS is known. Allowing nonmetric costs, on the other hand, makes the problem much harder. In particular, in this case, we cannot hope for any algorithm with bounded cost guarantee violating the capacities by less than 1.5.

3

Theorem 2. *Unless $P = NP$, no approximation factor on cost is possible if we cannot violate by at least a $3/2$ factor capacities of the unsplittable hard-capacitated non-metric facility location problem.*

Proof. Lenstra et al. [LST90] shows that minimizing the makespan is $3/2$-hard. The hardness also holds for a special case, in which the size of each client j is in a set $\{p_j, \infty\}$. This problem can be reduced to unsplittable hard-capacitated non-metric facility location problem. Let each client j have demand p_j corresponding to client j. The cost matrix c_{ij} is zero except for pairs (i, j) such that client j cannot be performed on machine i. In this case, $c_{ij} = \infty$. Any solution for the minimum makespan gives cost zero in the new problem and satisfies all the capacities. Furthermore, any solution having bounded size (zero in our case) transforms to a minimum makespan solution. Hence, it is NP-hard to get a bicriteria approximation algorithm violating capacities by less than $3/2$. □

In this paper, we obtain the first polynomial-time $(1 + \epsilon)$-approximation algorithm on tree metrics for the unsplittable hard-capacitated facility location problem when we allow violating facility capacities by a $1 + \epsilon$ factor and all facilities have almost the same capacity. This is proved in Section 2. If there is a solution satisfying the capacities whose cost is C, an (α, β)-*approximation algorithm* finds a solution for the problem, whose cost is at most αC such that no facility is given load more than β times its capacity.

Theorem 3. *There exists a $(1 + \epsilon', 1 + \epsilon')$-approximation algorithm for* Uniform Tree UHCMFL *running in polynomial time given any constant $\epsilon' > 0$.*

We prove this result for the case of uniform capacities, for the ease of demonstration. Extension to almost uniform capacities (i.e., the maximum and minimum capacities differ by no more than a constant factor) is immediate; the definition of large clients should span the range from $\max_j S_j$ down to $\epsilon \cdot \min_j S_j$.

Corollary 4. *There exists a $(1+\epsilon', 1+\epsilon')$-approximation algorithm for* Tree UHCMFL with Almost Uniform Capacities *running in polynomial time given any constant $\epsilon' > 0$.*

In Section 3, we show that, when different facilities have different capacities, we have the above result in quasipolynomial time $n^{O(\log n)}$.

Theorem 5. *There exists a $(1+\epsilon', 1+\epsilon')$-approximation algorithm for* Tree UHCMFL *running in quasipolynomial time given any constant $\epsilon' > 0$.*

Using Bartal's machinery [Bar98] for probabilistically embedding general metrics into tree metrics (or its slight improvement by Fakcharoenphol et al. [FRT04]) this immediately results in an $O(\log n)$ approximation algorithm for general metrics. To the best of our knowledge, this is the first non-trivial approximation algorithm for the unsplittable hard-capacitated metric facility location problem. We emphasize that the capacity violation stays the same.

Lemma 6. *Any (α, β)-approximation algorithm for Tree Facility Location leads to an $(O(\alpha \log n), \beta)$ approximation ratio for the Metric Facility Location problem.*

Proof. Given a weighted graph G, we can in polynomial time find a random tree T (using [FRT04]), such that the distance of vertices in T is roughly the same as those in G. More specifically, $d_G(u, v) \leq d_T(u, v) \leq O(\log n)d_G(u, v)$ for all pairs of vertices u and v. The second inequality is true in an expected sense, among the random choice of the tree.

Take the optimal solution for G. Then, the expected value of using the equivalent paths in T for the edges in this solution, is an $O(\log n)$ factor costlier. By linearity of expectation the solution value is $O(\log n)$ times the original optimal value. We can solve the problem on tree using our algorithms and then transform back to G by using the edges we had instead of the tree edges. The last step does not increase the cost. Therefore, we can, in randomized polynomial time (derandomization is straight-forward), find a solution which has cost $O(\alpha \log n)$ the optimal whose capacity constraints are violated by a β factor. □

Combining Lemma 6 with Corollary 4 and Theorem 5, we get the following Corollary.

Corollary 7. *There exists a polynomial time $(O(\log n), 1 + \epsilon)$-approximation algorithm for UHCMFL with almost uniform capacities, and a quasipolynomial time $(O(\log n), 1+\epsilon)$-approximation algorithm for nonuniform UHCMFL.*

Techniques In our algorithms, we divide the clients into different groups according to their demands. Different demand ranges need different kinds of treatment. There is also some rounding/discretization involved here which brings in the precision error. The grouping by size gets more complicated in the case of non-uniform capacities. We use a dynamic programming approach. For both cases, we need to prove structural theorems that allow us to decrease the number of DP states we need. Each node of the tree stores (for its different DP states) the clients not yet served in the subtree and the facilities having some unused capacity. There are two types of costs involved: the setup costs of facilities which is paid by them, and the connection costs. The latter cost is shared between the facilities and clients in our algorithm. In a rooted tree, the path between a client and a facility has two portions from their lowest common ancestor to each. Each pays for its own portion of the path. This way, we can let the dynamic programming account for this cost while it brings these clients and facilities up in the tree. Another trick is that each facility guesses (differently for different DP states) in what denomination its capacity will be used by clients. This way, it knows how much to pay for the costs coming up the tree and merging of DP states is easier. Last but not least, a nice feature in our first algorithm is that after the dynamic programming is done, we run a post-processing LP rounding step. One of our structural results is that a *semi-fractional* solution (see Corollary 12) is reasonable and can later be rounded to a good integral one. Semi-fractionality on the other hand allows a polynomial time dynamic programming to work out.

The reader might wonder why a set cover technique does not work for this problem. Roughly speaking, the main issue is that for such an approach to work, one needs the guarantee that any mistake should be correctable. That is, in our case, assigning a facility to some clients, should not restrict our options later. Unfortunately, this is not the case with hard capacities. In particular, if a facility is used once, it cannot be used again. This might lead to wrong decisions which make the problem infeasible. The problem is that we do not know if we should fill the facility to the maximum extent possible, or to leave some unused space in it to get a better density.

2 Uniform capacities

Let us for now assume that none of the demands is too small. In particular, if the capacity of facilities are S, then no client has demand smaller than ϵS. This assumption allows us to discretize the demands. Because, it puts an upper bound on the number of clients assigned to each facility, which in turn, controls the accumulated error on the total demand.

We work on rooted trees, the advantage compared to general graphs being the *locality* we have in our assignments. Take a subtree T with root v. Let e be the immediate edge joining v to an outside vertex. Any path that connects a client in T to a facility outside T, has to pass through this edge. Intuitively, a dynamic programming approach can account for such costs when taking into account the vertex v. We only need to know the amount of flow passing through the edge. Another trick is to have clients and facilities share the cost of their connection path. The connection path from a client to a facility has two portions: one that moves up the tree towards the root and the other in which it moves downwards to reach the facility. We charge the client for the first portion of the path and let the facility pay for the second portion. To put it another way, facilities and clients move to their least common ancestor and each pays the cost of getting there.

A DP subproblem, denoted by $(v, \vec{F}, \vec{D})$, seeks the least cost of a solution for the subtree T, rooted at v, in which all the clients in T should be either satisfied internally or brought to the root for *outsourcing* (these clients are somehow specified in $\vec{D}$) and further we have connected some facilities (details in $\vec{F}$) to the root of the tree to help demands from outside T. We should account for the initialization (only for selected facilities) and connection costs of facilities to clients inside the tree plus the cost of connecting the other clients as well as some facilities to the root of the tree.

Vector $\vec{D}$ has size $w = (1 - \epsilon)/\epsilon^2 + 1$ and describes approximately the demands of clients outsourced in this subproblem — brought up to the root of the tree and in the rest of the problem are supposed to be connected to outside facilities. The demand of any client is in the range $[\epsilon S, S]$. We round these values down with precision $\epsilon^2 S$. So, there would be only w distinct demand values, and hence we can represent

the number of unsatisfied clients in each group by an integer not exceeding n. Choice of ϵ comes from the fact that no facility can receive more than $1/\epsilon$ clients, so that the accumulated precision error is bounded by $1/\epsilon \cdot \epsilon^2 S = \epsilon S$.

The situation for facilities is a bit different. They have the same capacity, as a leaf they decide (i.e., they allow for all such options) if they are going to be used at all, and if so, by how many clients of what size. This introduces a number of facilities with virtual capacities. Each virtual facility can only be assigned to client of exactly the same demand. We perform the same rounding process and store the information about the number of facilities of different virtual capacities in $\vec{F}$.

Note that, having computed all the cost values for different subproblems (states of DP), the value we seek can be found in $(T, \vec{0}, \vec{0})$, where T is the whole tree of the instance and $\vec{0}$ is a vector of suitable size consisting only of zeros. We can assume without loss of generality that we have a rooted binary tree, where each facility or client is at a leaf. This can be done by adding zero cost edges and new vertices to the tree. The size of the instance can only linearly increase in this process. The base cases of DP are simple to compute: for trees which only have one vertex, we only need to decide how much virtual capacity a possibly existing facility in this tree might provide to the outside clients and in what denomination. There is no other decision to make, and this would lead to some of DP states receiving value zero while others would be set to infinity. For a nonleaf state, we will look at all the possible states of the children consistent with the state in question, and also consider any assignment that can be done at this level; i.e. clients from one subtree might be assigned to facilities offered by the other. Besides, the price of the edges connecting the subtrees to the new root should be computed according to how much flow is passing through them. We do not get into details at this point, as we are also going to accommodate *small clients* in the following. However, observe that, similar to standard DP approaches, if each node stores the states from which it was last updated, the best solution can actually be reconstructed recursively.

Issue of small clients If we want to pull the same argument through, the division value for demands should be about $\epsilon S/n$, since many clients with small demands (close to n such clients) can be assigned to a particular facility without violating the capacity constraint. That would make the number of different distinct demand values $\Theta(n)$ which will in turn make the space and running time of the algorithm exponential. To overcome this issue, we claim that small clients can be somewhat merged. Intuitively, their treatment is quite flexible. In our algorithm, we treat them as if they do not have any integrality constraints. We find a solution which might be fractional with respect to small clients, and then show that an integral assignment of about the same capacity usage exists. This is in contrast to big clients which are part of a knapsack instance, and need to be carefully packed to make best use of facility capacities.

Handling tiny demands In addition, we have another set of clients, which is very small. We call them *tiny demands*. They are so small that capacities are irrelevant for them. We could deal with them as a separate instance of uncapacitated metric facility location, to get an additive 1.52 (or 1 if we use a customized algorithm for trees [Gim83, Gri94, SFC02]) term on cost guarantee. However, we avoid this separate handling of tiny clients to get rid of the additive term. To this end, we simplify the structure of their assignment and incorporate it into the main DP. All tiny clients in a subtree that climb up to a certain point, can be made to go to the same (closest) facility. Thus, each subtree announces at most one facility for export (the one closest to the root). In this part of DP (unlike our main algorithm), facilities do not pay for the connection path. Clients pay their full connection cost upfront. Similarly, each subtree can request for at most one facility to be turned on, meaning it wants to connect some of its tiny clients to that facility. The children of a node should be consistent in asking for a facility; i.e., they cannot request two facilities outside. Furthermore, if the requested facility of a subtree is inside its sibling subtree, that should have been exported.

In the rest of the paper, we assume that $1/\epsilon$ is an integer. There are three types of clients.

- *large clients* are those whose demands are no less than ϵS. They are rounded down to the closest multiple of $\epsilon^2 S$. These demands are large enough to require special care. They should be carefully packed into facilities. The total precision error in each facility due to them is no more than ϵS. In

the DP characteristic, we store the number of large clients (or virtual capacities) for each such value (there are at most $1 + (1 - \epsilon)/\epsilon^2$ such values).

- *small clients* are those whose demands are between ϵS and $\epsilon S/n$. These are rounded down to the closest multiple of $\epsilon^2 S/n$. Since each of them has a negligible size, we have some flexibility in assigning them to facilities. In particular, their assignment is done fractionally, and Corollary 12 shows this treatment is not bad. So, we store only a single number which is the sum of the demands of all such clients.

- *tiny clients* which have demands less than $\epsilon S/n$. These are too small to violate any capacity constraint significantly, even if all of them are assigned to one already full facility. Thus, at each such leaf, they guess the facility they are going to be assigned to, and pay the connection price. In addition, they request that facility to be initialized. That is, they *import* that facility and this request can be satisfied if the desired facility is exported.

Take any solution to the problem. There is a notion of a *partial solution*, which is the solution restricted to a particular subtree. Not all the clients in the subtree are necessarily handled in there and therefore, some might be outsourced. For the very same reason, some facilities might offer their capacity to help the clients outside the subtree. Each subtree can import some facilities for its tiny clients, i.e., ask them to be available, and in the same way, a subtree exports its facilities for tiny clients. In the following, we only consider *restricted* solutions; in a *restricted* solution, no subtree exports or imports more than one facility.

Lemma 8. *There exists a restricted solution with cost at most $1 + \epsilon$ times the optimal, with facilities overloaded by at most $1 + \epsilon$ times their capacities.*

Proof. If a subtree exports more than one facility, then every client using them might as well use one of them, i.e., the one closest to the root and pay no more. Similarly, if two or more facilities are imported by a subtree, then all the tiny clients asking for them can connect to only one of them without increasing the cost. The capacity violation is at most a factor $1 + \epsilon$. □

Equipped with such a structural understanding, we can now define the characteristic of a DP state. We compute the minimum cost value for any such configuration and use them to find the final answer.

Definition 2. *A DP state is specified by $(u, \vec{F}, \vec{D}, F_s, D_s, F_{\exp}, F_{\imp})$, where*

- *u is the root of the subtree associated with this partial solution;*

- *$\vec{F}$ is a vector that keeps the count of the available facilities of large virtual capacities;*

- *$\vec{D}$ is a vector that keeps the count of the outsourced large clients of different demands;*

- *F_s is the amount of facility capacity offered to small clients which is stored as a multiple of $\epsilon^2 S/n$;*

- *D_s is the total demand of outsourced small clients, and is stored as a multiple of $\epsilon^2 S/n$ as well; and*

- *$F_{\exp}$ is the index of a facility being exported from this subtree (or maybe NONE). This facility should have been initialized; and*

- *$F_{\imp}$ is the index of a facility which is imported (or maybe NONE). That is, this facility should be turned on by its subtree for the use of some tiny clients in the subtree under u.*

Two DP cells corresponding to children of a node can be used to update the value of a cell at that node. There are certain conditions which are necessary for this.

Definition 3. *A state $\chi = (u, \vec{F}, \vec{D}, F_s, D_s, x, y)$ is consistent with $\chi_1 = (u_1, \vec{F_1}, \vec{D_1}, F_s^1, D_s^1, x_1, y_1)$ and $\chi_2 = (u_2, \vec{F_2}, \vec{D_2}, F_s^2, D_s^2, x_2, y_2)$ if and only if*

1. *$\vec{F_1} + \vec{F_2} - \vec{F} = \vec{D_1} + \vec{D_2} - \vec{D} \geq \vec{0}$;*

2. $F_s^1 + F_s^2 - F_s = D_s^1 + D_s^2 - D_s \geq 0$;

3. $y_1 = x_2$ or $y_1 = y$ or $y_1 = NONE$;

4. $y_2 = x_1$ or $y_2 = y$ or $y_2 = NONE$; and

5. $x = x_1$ or $x = x_2$ or $x = NONE$.

The first two conditions ensure that the demand ignored at node u is actually matched to facilities of the same size. The next two make sure that what a subtree asks for, is either provided or is put on the request list for the next node. The last condition prevents us from exporting a facility we do not have.

We also define for a configuration $\chi = (u, \vec{F}, \vec{D}, F_s, D_s, x, y)$, the shorthand $\text{cost}(\chi)$ to be $\text{DP}(\chi)$ plus the cost of moving the outsourced clients in $\vec{D}$ and D_s and the offered facilities in $\vec{F}$ and F_s from u to its parent. More specifically,

$$\text{cost}(\chi) = \text{DP}(\chi) + w\text{Total}(\vec{F} + \vec{D}, F_s + D_s),$$

where w is the cost of the edge immediately above u and Total is a function which adds up a vector of large clients and a small demand value, taking into account their weights. The update rule just takes the value $\text{cost}(\chi_1) + \text{cost}(\chi_2)$ for two states to update their parent.

We first show that the solution given by the dynamic programming has a low cost. The solution is *semi-fractional*, in the sense that the assignment of small clients can be fractional, but the assignment of large and tiny clients is done integrally.

Lemma 9. *No restricted semi-fractional solution is discarded in our DP using update function and the base case definition.*

Proof. Take any integral solution to the original problem. Capacity of facilities remains the same in our DP, whereas demands of clients might only decrease. Therefore, there is a solution for the rounded instance having cost no more than OPT. Since small clients have demands which are a multiple of $\epsilon^2 S/n$, the capacity of a facility for small clients is also a multiple of this unit. In the base case of DP, allocate for each facility the correct capacities for small and large clients respectively. Then, we can do an induction on the height of each node in the tree to show that there is a solution as cheap at each node. The base case is straight-forward and the update rule for the dynamic programming at each level considers all possible ways to assign clients to facilities. The only thing we ignore is that clients having the same demand do not differ from one another. The same thing holds for virtual capacity of facilities. The consistency rule for the tiny clients (imports and exports) is by definition sound and complete for the tiny clients. $\square$

We now show that the running time is polynomial.

Lemma 10. *The number of states and the running time of the DP is polynomial*

Proof. The total number of DP characteristics is bounded by $\Phi = n^3 n^{O(1/\epsilon^2)} (n^2/\epsilon)^2$: There are roughly n choices for u, $F_{\exp}$ and $F_{\imp}$. Each vector has $n^{O(1/\epsilon^2)}$ choices. We have n^2/ϵ options for F_s or D_s, since each small demand is at most n/ϵ times the unit they are measured against, and there are at most n such demands. This argument is only true for demands, and for facilities, we need to be careful to never allocate more than this for small demands, as they cannot be used.

At each node, we consider all the configurations of its two children and do some polynomial work. Thus, the running time is $O(\Phi^3 \text{poly}(n))$ which is a polynomial. $\square$

The complete algorithm for uniform capacities is given in Figure 1. The significant addition from previous description of the algorithm is the LP Rounding step. We have a semi-fractional assignment of clients to open facilities with a good cost C. We invoke the rounding algorithm of Shmoys and Tardos [ST93] on the fractional part to get a good integral solution. They introduce the following LP:

Algorithm Uniform-UHCMFL Solver
Input: *Weighted graph G, facilities A, clients B, demands d_i, capacity S and setup costs f_j*
Output: *Subset A^* of activated facilities, and assignment π*

1. Change the instance into a rooted binary tree with facilities and clients at leaves; remove any other leaves.

2. Round down large ($\geq \epsilon S$) client demands with precision $\epsilon^2 S$.

3. Round down small client demands (between $\epsilon S/n$ and ϵS) with precision $\epsilon^2 S/n$.

4. Initialize DP for leaves:

   - If there is a client with demand d at a leaf v,
     - for a non-tiny demand, let $(v, \vec{0}, \vec{D}, 0, d_s, \text{NONE}, \text{NONE})$ get value 0 if demand d is represented in $\vec{D}$ or d_s depending on its size;
     - for a tiny demand, let for any facility x, DP value associated with $(v, \vec{0}, \vec{0}, 0, 0, \text{NONE}, x)$ be d times the distance between x and v; and
     - all other values corresponding to this leaf are set to infinity.
   - If facility i is located at this leaf,
     - let $(v, \vec{0}, \vec{0}, 0, 0, \text{NONE}, \text{NONE})$ get value 0;
     - for any vector $\vec{F}$ and f_s such that $0 < \text{Total}(\vec{F}, f_s) \leq S$, let $\text{DP}(v, \vec{F}, \vec{0}, f_s, x, \text{NONE})$ have value f_i (setup cost of this facility), where x can be either v or NONE; and
     - put value infinity for any other DP cell corresponding to this subtree.

5. Update the DP values bottom-up.
   If the node u has two children u_1 and u_2, a cell $\chi = (u, \vec{F}, \vec{D}, F_s, D_s, x, y)$ is updated (if this gives a better value) from $\chi_1 = (u_1, \vec{F_1}, \vec{D_1}, F_s^1, D_s^1, x_1, y_1)$ and $\chi_2 = (u_2, \vec{F_2}, \vec{D_2}, F_s^2, D_s^2, x_2, y_2)$ to $\text{cost}(\chi_1) + \text{cost}(\chi_2)$, if C is *consistent* with χ_1 and χ_2.

6. Build the answer recursively from $\text{DP}(r, \vec{0}, \vec{0}, 0, 0, \text{NONE}, \text{NONE})$, with r being the root of the tree.

7. Round the fractional assignments using procedure of Shmoys and Tardos [ST93].

8. Use the same values of π and A^* for the original instance.

Figure 1: Algorithm for the uniform capacity case.

$$\text{minimize} \qquad \sum_{ij} c_{ij}x_{ij} \qquad\qquad\qquad\qquad\qquad (1)$$

$$\text{subject to} \qquad \sum_{i} x_{ij} = 1 \qquad\qquad \forall \text{ job } j,$$

$$\sum_{j} x_{ij}d_{ij} \leq S_i \qquad\qquad \forall \text{ machine } i,$$

$$x_{ij} \geq 0 \qquad\qquad\qquad \forall i,j.$$

Theorem 11 (MinMax LP Rounding [ST93])**.** *Having a fractional solution to LP* (1)*, we can round it without increasing the cost, but violating the capacity constraints by at most the maximum running time of a job.*

Noting that demands of small clients are bounded by ϵS, we use the procedure they propose to obtain the following direct corollary.

Corollary 12. *Converting a semi-fractional solution to an integral solution does not increase the cost, and can increase the loads by at most ϵS.*

Now comes the proof of the main result.

Theorem 3. The algorithm in Figure 1 achieves this. By Lemmas 8 and 9, and by completeness of the DP, its cost is at most $1 + \epsilon$ times the optimal. Restoring the original demand of big clients would increase the load of each facility by at most $1/\epsilon \cdot \epsilon^2 S = \epsilon S$. By the same token, restoring the demand of small clients would have the same effect. Then, we can get a fractional solution to LP (1) for small clients which can be rounded by increasing the capacities by at most ϵS. To summarize, the total increase in demands would be $(1 + \epsilon)^3 \approx 1 + 3\epsilon$.

Increase in cost is only done in restoring to original sizes, which is $(1 + \epsilon)$. Hence, we get a bicriteria guarantee of $1 + \epsilon$ for cost and $(1 + \epsilon)^3$ for capacities. Then, we can pick $\epsilon \approx \epsilon'/3$ small enough to get the PTAS guarantee with parameter ϵ'. Polynomial running time is established in Lemma 10. $\qquad\square$

3   Non-uniform Capacities

The big issue with non-uniform capacities is that there is no fixed definition of being large, small or tiny for clients. This was crucial to the design of the previous algorithm, as their treatments were completely different. Here, we can only have such definitions relative to a particular facility. Therefore, it is not a priori obvious as to which role a specific client is going to play.

We need some discretization for a dynamic programming approach. The subdivision size should be some constant related to ϵ so that the error resulting from rounding is not too large. Shooting for a PTAS, we should keep at each point only a constant sized vector, i.e., the number of different sizes (demands or capacities) at each dynamic programming state should be a constant. However, even if we simplistically neglect the effect of overlapping intervals, and build intervals of constant length around each facility capacity (and client demand), the DP approach does not work smoothly. Since that can lead to an exponential number of states. This is true for a special case in which the sizes are partitioned into well-separated ranges. Clients in each range only need to decide if they are going to be assigned to servers in that range (and obey capacity constraints) or not. This decision does not seem to be easy to make.

One way to tackle the problem is to work in iterations by doing a dynamic programming similar to the previous section. That is, divide the demand/capacity values into separate intervals and at each iteration, add a new group of values to the instance, update the DP table using the table from previous levels. The idea is that the details of the decisions made in previous levels does not matter for the new level, and the smaller facility sizes cannot be used for larger clients. Nonetheless, our attempts failed to find a way of using the decisions at previous levels as a black-box to solve the augmented instance. The problem is basically that at a certain node of the tree, there might be demand/client pair of really different sizes (about a factor

n apart) that should be matched. Besides, if we want to carry around the condition of unsatisfied demands (and available facilities) from the previous levels, their capacities/demands alone does not suffice. We need to know their positions and their detailed demand vector, so that future levels of DP can handle them. The vector should then have size at least logarithmic.

We will follow a similar DP approach, i.e., storing for each subtree the best assignment cost given the status of demands outsourced and facilities offered to the client outside the subtree. To take the number of states into a tractable range, we have to play a number of tricks. The standard trick is the well-known discretization found in many PTAS algorithms, similar to what we did in the previous section. Because of the non-uniformity, we need to have different large/small intervals. These are defined relative to one particular facility or demand, at each point. Unfortunately, we do not know how to accomplish the merging trick for small clients, so we just ignore the concept of small clients altogether (and treat them as large clients). Each client would be either large or tiny, with respect to a facility. The caveat is that this makes the interval size polynomial, rather than constant as in previous algorithm, which in turn makes the number of DP states quasipolynomial.

We also need another trick. We treat small clients as big clients. But, we cannot get rid of tiny clients in the same way unless all the input demands and capacities are polynomially bounded. If they are, then the algorithm is much simpler. The new trick is to ignore, in a sense, the *tiny clients* during the computations, meaning we do not bother to store their information in the DP state. This is done carefully so that their cost can be charged to other clients without increasing the total cost considerably. Doing this is a bit tricky because of the fluid definition of being tiny.

To do this, we first round each capacity or demand to its closest $(1 + \epsilon)^k$. We actually round demands down and facilities up. The loss in the cost and capacity usage using the new values is roughly a factor 2ϵ more. Since we have not lost any solution of the original problem due to capacity constrains, and restoring the original demands/capacities would only increase the cost or load on a facility by $(1 + \epsilon)^2$. Furthermore, there are only polynomially many different values now. Because, the input values are supposed to have polynomial size.[1] Then, for each important size value (rounded demand or capacity), we build an interval of multiplicative length n/ϵ, the upper value being the size value considered and the lower limit being a factor n/ϵ smaller. The interval is divided into $(1 + \epsilon)$-sized subdivisions (multiplicatively). So, there are $O(\log n)$ such values in any interval. Each interval is represented by the index of its largest value, called its *base*, and similarly to $\vec{F}$ and $\vec{D}$ in the previous algorithm, is also given a vector of logarithmic length.

Now let us see what the characteristic of a DP state is. It should describe a partial solution.

Definition 4. *A DP state is specified by* $(u, base, \vec{F}, \vec{D})$, *where*

- u *is the root of the subtree associated with this partial solution;*

- *base is the largest capacity or demand (for storage purposes, its index) in vectors $\vec{D}$ or $\vec{F}$;*

- $\vec{F}$ *is a vector of logarithmic length that keeps the count of the available facilities of different sizes in a range;*

- $\vec{D}$ *is a vector of logarithmic length that keeps the count of the outsourced clients of different demands in a range; and*

The span of $\vec{F}$ and $\vec{D}$ is a multiplicative range of ϵ/n. Therefore, they have length $\log_{1+\epsilon} \frac{n}{\epsilon}$. In addition, we have the property that unless $\vec{F} = \vec{D} = \vec{0}$, one of them has a nonzero value in its most significant entry. This means, we store these vectors in a normalized fashion.

We note that, using this definition for a partial solution, a subtree cannot offer two facilities whose capacities differ by more than a factor n/ϵ. Nor will it outsource clients of considerably different demands. But, we will deal with this seemingly problematic issue shortly. Furthermore, if the subtree cannot outsource a demand d if it is also offering facilities of capacity larger than $n/\epsilon d$. Nor will it offer a facility of capacity c

---

[1]Note that we are not assuming the numbers are polynomially bounded, in which case the number of distinct values would only be logarithmic.

while it is also asking for one of capacity $n/\epsilon c$. Let us emphasize that associated with a state $(u, \text{base}, \vec{F}, \vec{D})$, there might be a hidden demand of up to $\epsilon \cdot \text{base}$ not accounted for explicitly as we move up the tree.

The following lemmas show these issues are not very serious. A solution is said to be *facility-relaxed* if none of whose partial solutions offers a facility that is smaller than another offered facility or outsourced client of the same subtree by more than a n/ϵ factor.

Lemma 13. *For the instance rounded as above, there is an approximately optimal solution, in particular one with bicriteria approximation ratio $(1 + \epsilon, 1 + \epsilon)$ which is facility-relaxed.*

Proof. Take any solution for a rounded instance. No facility is overloaded. For two reasons, we might need to get rid small facilities: there is another larger facility offered from this subtree; or there is a large demand being outsourced. If the former happens, redirect any client using the small facility to use the bigger one. For the latter, make any client using the small facility use the facility the large demand is assigned to. In both cases, the small client is reassigned to a facility whose capacity is at least n/ϵ larger than the demand. Even if n clients are reassigned to facility i, the overall increase in load is no more than ϵS_i. As for cost, one should note that, along each edge, we are charging to the current flow, and the increase is at most an ϵ factor. □

In our algorithm, before doing the dynamic programming, we inflate the capacities one notch to the next level. More specifically, each is multiplied by $1 + \epsilon$ and thus we allow for the violation needed in Lemma 13. A solution is *client-relaxed* if at each node of the tree, it might discard demands that are smaller than another (or an offered facility) by more than a factor ϵ/n.

Lemma 14. *Any* client-relaxed *solution can be turned into a complete one, increasing the cost and loads by at most a $(1 + \epsilon)$ factor.*

Proof. In a client-relaxed solution obtained, some demands might have been ignored. This can have either of two reasons: there was a large demand being outsourced from the subtree, or a large facility offered. In the latter case, assign the client to the facility, and in the former, reassign to the facility the large demand is using. The capacity of the facility used is larger than any new demand by at least a factor n/ϵ. There are at most n such clients and thus the total violation is bounded as desired. A similar argument shows that the connection cost can be charged to the current costs. □

Given $\phi_1 = (m_1, \vec{V_1})$ and $\phi_2 = (m_2, \vec{V_2})$, their sum $\phi = \phi_1 + \phi_2 = (m, \vec{V})$ is defined as follows. First we should normalize ϕ_i if $\vec{V_i} = \vec{0}$. This is done by changing m_i to a small enough value. Then m is simply $\max\{m_1, m_2\}$. Next, we shift the vector $\vec{V_i}$ corresponding to the smaller m_i, possibly losing some nonzero elements, so that its m_i equals the larger one. Finally, we can just add up the vectors to get $\vec{V}$.

We again define for a configuration $\chi = (u, \phi_1, \phi_2)$, the shorthand $\text{cost}(\chi)$ to be $\text{DP}(\chi)$ plus the cost of moving the outsourced clients in ϕ_1 and the offered facilities in ϕ_2 from u to its parent.

Theorem 5. By induction on the depth of a tree node, we prove that assuming discrete values for capacities and demands, we compute the best facility-relaxed client-relaxed solution. Lemma 13 states the value of this solution is at most $1 + \epsilon$ times the optimal solution, and the increase in capacities is $(1 + \epsilon)^2$ due to rounding and capacity inflation. Then, using Lemma 14, we can find a solution with cost at most $(1 + \epsilon)^2$ times the optimal that violates capacities by $(1 + \epsilon)^3$. Rounding the demands back to the original, we get an $((1 + \epsilon)^3, (1 + \epsilon)^3)$-approximation algorithm. Then, we just pick $\epsilon = \epsilon'/3$.

The base case of induction is simple. Leaves for demand are straight-forward and as for facilities, we consider every possible way (including not turning the facility on) to allocate the capacity of this facility for the clients.

The induction step is clearly sound. We take any two states for the children nodes, add them up, and assign any possible configuration of clients to facilities in the result. It is also complete. Since, we can consider any node and any way to connect clients of one side to facilities of the other. The rest should be handled in the children themselves.

Algorithm UHCMFL Solver
Input: *Weighted graph G, facilities A, clients B, demands d_i, capacities S_j and setup costs f_j*
Output: *Subset A^* of activated facilities, and assignment π*

1. Change the instance into a rooted binary tree, with facilities and clients at leaves; remove any other leaves.

2. Round down demands to nearest $(1 + \epsilon)^k$.

3. Round capacities up to nearest $(1 + \epsilon)^k$. Then, multiply them by $(1 + \epsilon)$.

4. Initialize DP for leaves:

   - If there is a client with demand d at leaf v,
     - let $(v, d, \vec{0}, \vec{D})$ get value 0 if there is a 1 at the most significant entry of $\vec{D}$; and
     - all other values corresponding to this subtree are set to infinity.
   - If facility i with capacity c is located at this leaf,
     - let $(v, 0, \vec{0}, \vec{0})$ get value 0;
     - for any value m and $\vec{F}$ such that $0 < m \sum_{i \geq 0}(1+\epsilon)^{-i}F_i \leq c$, let $\mathrm{DP}(v, m, \vec{F}, \vec{0})$ have value f_i (setup cost of this facility); and
     - put value infinity for any other DP cell corresponding to this subtree.

5. Update the DP values bottom-up.
   For a node u whose children are v and w, take any two configurations $\chi_1 = (v, \phi_{11}, \phi_{12})$ and $\chi_2 = (w, \phi_{21}, \phi_{22})$.
   Let $\chi = (u, \phi_1, \phi_2)$, where $\phi_1 = \phi_{11} + \phi_{21}$ and $\phi_2 = \phi_{12} + \phi_{22}$.
   Take any vector $\vec{X}$ such that $\vec{X} \leq \phi_1$ and $\vec{X} \leq \phi_2$ after normalization, and let χ' be the result of subtracting $\vec{X}$ out of χ.
   Update $\mathrm{DP}(\chi')$ with $\mathrm{cost}(\chi_1) + \mathrm{cost}(\chi_2)$ if this is better than the current value.

6. Build the answer recursively from $\mathrm{DP}(r, 0, \vec{0}, 0, \vec{0})$, where r is the root.

7. Handle the forgone demands by assigning them to their replacement.

8. Use the same values of π and A^* for the original instance.

Figure 2: QPTAS for non-uniform capacities.

The number of DP characteristics we have is $\Phi = n^3 \cdot n^{O(\log n)}$. Since there are n^2 ways to choose the root node and the base, and each element in the vector can be at most n. The runtime is at most $\Phi^3 \text{poly}(n)$. Hence, we have a QPTAS. $\qquad\square$

4   Open Problems

The main open problem is to solve the problem directly for general graph metrics, avoiding the tree embedding technique, to get rid of the $O(\log n)$ factor in the approximation guarantee.

It is also instructive to improve the running time of the second algorithm, to obtain a PTAS for non-uniform capacities. When we have widely differing capacities, we do not know how to generalize the concept of small clients, so that we can have a small sized vector for large clients. One class of problematic instances has distinct capacity values that are roughly a factor n apart. Clients in each range have to decide whether they want to connect to facilities of their own size or the larger ones.

A variant of the problem is when the cost of serving a client does not depend on its demand. This case is particularly useful in an application where the incremental serving cost is negligible compared to the initialization cost of transportation. We do not know how to solve this variant, since handling of small jobs should be somewhat different.

It is quite natural to consider a generalization of this model to multi-dimensional demands. For example, each client i can have a *bandwidth demand d_i* and a *CPU demand c_i*. The connection cost would be proportional to d_i times distance, while each facility can only handle demands whose CPU requirements c_i add up to at most S, and whose bandwidth demands d_i do not exceed D. We have made some progress on this front, and partial results in a different networking context has been obtained [BGHSar].

Acknowledgment

References

[AGK+01] Vijay Arya, Naveen Garg, Rohit Khandekar, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 21–29, 2001.

[ALR+08] Hussein A. Alzoubi, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van der Merwe. Anycast CDNs revisited. In *Proceeding of the 17th International Conference on World Wide Web (WWW)*, pages 277–286, 2008.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 161–168, 1998.

[BGHSar] MohammadHossein Bateni, Alexandre Gerber, MohammadTaghi Hajiaghayi, and Shubhabrata Sen. Multi-VPN optimization for scalable routing via relaying. In *Proceedings of the 28th Annual Conference of the IEEE Communications Society (INFOCOM)*, to appear.

[BH09] MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Assignment problem in content distribution networks: Unsplittable hard-capacitated facility location. In *Proceedings of the twentieth annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 805–814, 2009.

[Byr07] Jaroslaw Byrka. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In *Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 29–43, 2007.

[CG99]   Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location and
           k-median problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer
           Science (FOCS)*, pages 378–388, October 1999.

[Chu98]   Fabián A. Chudak. Improved approximation algorithms for uncapacited facility location. In
           Proceedings of Integer Programming and Combinatorial Optimization (IPCO), pages 180–194,
           1998.

[CK00]   Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *Pro-
           ceedings of the eleventh annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages
           213–222, 2000.

[CS98]   Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapac-
           itated facility location problem. Unpublished manuscript, 1998.

[CW05]   Fabián A. Chudak and David P. Williamson. Improved approximation algorithms for capacitated
           facility location problems. *Mathematical Programming*, 102(2, Ser. A):207–222, 2005.

[FGMS06]   Lisa Fleischer, Michel X. Goemans, Vahab S. Mirrokni, and Maxim Sviridenko. Tight approxi-
           mation algorithms for maximum general assignment problems. In *Proceedings of the seventeenth
           annual ACM-SIAM Symposium on Discrete algorithm (SODA)*, pages 611–620, 2006.

[FJVW86]   Marshall L. Fisher, R. Jaikumar, and Luk N. Van Wassenhove. A multiplier adjustment method
           for the generalized assignment problem. *Management Science*, 32(9):1095–1103, 1986.

[FRT04]   Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary
           metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[Gim83]   E. K. Gimadi. An effective algorithm for the solution of the location problem with service
           domains connected with respect to an acyclic network. *Upravlyaemye Sistemy*, 23:12–23, 1983.

[GK99]   Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms.
           Journal of Algorithms, 31:228–248, 1999.

[Gri94]   V. P. Grishukhin. On polynomial solvability conditions for the simplest plant location problem.
           Transactions of the American Mathematical Society, 2(158):37–46, 1994.

[HS87]   Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling
           problems: theoretical and practical results. *Journal of the Association for Computing Machinery*,
           34(1):144–162, 1987.

[HS88]   Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling
           on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*,
           17(3):539–551, 1988.

[JMS02]   Kamal Jain, Mohamma Mahdian, and Amin Saberi. A new greedy approach for facility location
           problem. In *Proceedings of the 34rd ACM Symposium on Theory of Computing (STOC)*, pages
           731–740, 2002.

[JV01]   Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and
           k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the
           ACM*, 48(2):274–296, 2001.

[KH63]   A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management
           Science*, 9:643–666, 1963.

[KPR00]   Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search
           heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000.

[LSS04]   Retsef Levi, David B. Shmoys, and Chaitanya Swamy. LP-based approximation algorithms for capacitated facility location. In *Proceedings of Integer Programming and Combinatorial Optimization (IPCO)*, pages 206–218, 2004.

[LST90]   Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(3, (Ser. A)):259–271, 1990.

[LV92]   Jyh-Han Lin and Jeffrey S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC)*, pages 771–782, 1992.

[MMSV01] Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proceedings of the fourth International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 127–137, 2001.

[MP03]   Mohammad Mahdian and Martin Pál. Universal facility location. In *Proceeding of 11th Annual European Symposium on Algorithms (ESA)*, pages 409–421, 2003.

[MYZ02]   Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 229–242, 2002.

[PTW01]   Martin Pál, Éva Tardos, and Tom Wexler. Facility location with nonuniform hard capacities. In *Proceedings of 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 329–338, 2001.

[RS75]   G. Terry Ross and Richard M. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8:91–103, 1975.

[SFC02]   Rahul Shah and Martin Farach-Colton. Undiscretized dynamic programming: faster algorithms for facility location and related problems on trees. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 108–115, 2002.

[Shm00]   David B. Shmoys. Approximation algorithms for facility location problems. In *Proceedings of Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 27–33, 2000.

[ST93]   David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3, Ser. A):461–474, 1993.

[STA97]   David B. Shmoys, Éva Tardos, and Karen I. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 265–274, 1997.

[ZCY04]   Jiawei Zhang, Bo Chen, and Yinyu Ye. A multi-exchange local search algorithm for the capacitated facility location problem. In *Proceedings of Integer Programming and Combinatorial Optimization (IPCO)*, pages 219–233, 2004.