Approximation Algorithms for Data Placement Problems[*]

Ivan Baev[†] Rajmohan Rajaraman[‡] Chaitanya Swamy[§]

Abstract

We develop approximation algorithms for the problem of placing replicated data in arbitrary networks, where the nodes may both issue requests for data objects and have capacity for storing data objects, so as to minimize the average data-access cost. We introduce the *data placement problem* to model this problem. We have a set of caches $\mathcal{F}$, a set of clients $\mathcal{D}$, and a set of data objects $\mathcal{O}$. Each cache i can store at most u_i data objects. Each client $j \in \mathcal{D}$ has demand d_j for a specific data object $o(j) \in \mathcal{O}$ and has to be assigned to a cache that stores that object. Storing an object o in cache i incurs a storage cost of f_i^o, and assigning client j to cache i incurs an access cost of $d_j c_{ij}$. The goal is to find a placement of the data objects to caches respecting the capacity constraints, and an assignment of clients to caches, so as to minimize the total storage and client access costs.

We present a 10-approximation algorithm for this problem. Our algorithm is based on rounding an optimal solution to a natural LP-relaxation of the problem. One of the main technical challenges encountered during rounding is to preserve the cache capacities while incurring only a constant-factor increase in the solution cost.

We also introduce the *connected data placement problem*, to capture settings where write-requests are also issued for data objects, so that one requires a mechanism to maintain consistency of data. We model this by requiring that all caches containing a given object be connected by a Steiner tree to a root for that object, which issues a multicast-message upon a write to (any copy of) that object. The total cost now includes the cost of these Steiner trees. We devise a 14-approximation algorithm for this problem.

We show that our algorithms can be adapted to handle two variants of the problem: (a) a k-median variant, where there is a specified bound on the number of caches that may contain a given object; (b) a generalization where objects have lengths and the total length of the objects stored in any cache must not exceed its capacity.

[*]This work is a combined version of two papers: an extended abstract by Baev and Rajaraman [3] that appeared in the Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, and an unpublished manuscript by Swamy [38].

[†]`ivan.baev@hp.com`. Java, Compilers, and Tools Laboratory, Hewlett-Packard Company, 11000 Wolfe Road, Cupertino, CA 95014.

[‡]`rraj@ccs.neu.edu`. College of Computer Science, Northeastern University, Boston, MA 02115. Supported by NSF CAREER award NSF CCR-9983901.

[§]`cswamy@math.uwaterloo.ca`. Combinatorics and Optimization, University of Waterloo, Waterloo, ON N2L 3G1. Supported in part by NSERC grant 32760-06. Part of this work was done while the author was a student at Cornell University, NY 14853.

1 Introduction

Consider a distributed network, some of whose nodes need to periodically access certain data objects, and some of whose nodes have storage capacity and may serve as caches to store data objects thereby reducing the cost of accessing data objects. For example, one could have a network of distributed caches and/or processors in a large-scale information system or computing environment. A powerful paradigm to improve performance, which has been explored in several studies [10, 5, 1, 28] is cooperative caching, wherein the caches cooperate in making storage decisions and in serving each others' requests. (Cooperation is of course likely to be the default mode under centralized control, where all the network nodes are under the control of a single entity, e.g, as in an organization's local area network.) Clearly, cooperation has the potential to improve system performance by reducing average access cost and improving storage-space utilization. A basic problem that arises in such a cooperative setup is: given a cost function specifying the cost of accessing an object stored at one location from another, and the access-pattern of each node for each object, determine a placement or mapping of the data objects to caches, so as to minimize the average cost of accessing the data objects.

We abstract this problem via the following mathematical formulation, which we call the *data placement problem*. We are given a set of caches $\mathcal{F}$, a set of data objects $\mathcal{O}$, and a set of clients $\mathcal{D}$. Each cache $i \in \mathcal{F}$ has a capacity u_i that limits the total number of data objects that may be stored in the cache. Each client $j \in \mathcal{D}$ has demand d_j for a specific data object $o(j) \in \mathcal{O}$ and has to be assigned to a cache that stores that object. Storing an object o in cache i incurs a *storage cost* of f_i^o, and assigning client j to cache i incurs an *access cost* of $d_j c_{ij}$ proportional to the "distance" c_{ij} between i and j. The storage costs could be used to model the cost of realizing a placement, e.g., the cost f_i^o might represent the cost of expunging some items from the cache in order to free up storage space. The data placement problem seeks a placement of the data objects to caches that respects the cache capacities, and an assignment of clients to caches, so as to minimize the total storage and client access costs. More precisely, we want to determine a set of objects $\mathcal{O}(i) \subseteq \mathcal{O}$ to place in each cache $i \in \mathcal{F}$ satisfying $|\mathcal{O}(i)| \leq u_i$, and assign each client j to a cache $i(j)$ that stores object $o(j)$, (i.e., $o(j) \in \mathcal{O}(i(j))$) so as to minimize $\sum_{i\in\mathcal{F}} \sum_{o\in\mathcal{O}(i)} f_i^o + \sum_{j\in\mathcal{D}} d_j c_{i(j)j}$. As in several previous studies, especially on facility location [5, 1, 37, 34, 7, 6], we assume that the caches and clients are located in a common metric space, so the distances c_{ij} form a metric.

More generally, each object $o \in \mathcal{O}$ may have a *length* l_o, and the capacity u_i of cache $i \in \mathcal{F}$ now bounds the total length of data objects that may be stored in the cache. The access-cost of an object is weighted by its length, so if client j is assigned to cache i it incurs an access cost of $d_j l_{o(j)} c_{ij}$. Unless otherwise stated, we will use the data placement problem to denote the problem with unit (or equivalently, uniform) object-lengths.

The data placement problem can also be motivated from a facility-location perspective. In a typical facility-location setting, we are given a set of facilities with facility-opening costs and a set of clients with demands, and we want to open facilities and assign clients to open facilities so as to minimize the sum of the facility-opening costs and client-assignment costs. In various such applications, the clients are differentiated according to the kind of service they require, and in order to satisfy a client we need to assign (the demand of) the client to a facility where the service required by it has been "installed" (so that the facility can provide this service). The data placement problem can be used to abstract such settings, wherein the caches represent facilities and the objects correspond to the different services required by the clients; the storage cost models the cost of installing service at a given facility, and the cache-capacity imposes a restriction on the *number of services* that may be installed at a facility. Shmoys et al. [35], and Ravi and Sinha [33] introduced problems closely related to the data placement problem, motivated by such facility-location applications.

The data placement problem is a generalization of the metric *uncapacitated facility location* (UFL) problem, and hence, is *APX*-hard. Moreover, as we show in Section 6 by a reduction from metric UFL, the problem (with uniform object-lengths) remains *APX*-hard even when there are no storage costs.

Our results and techniques. Our main result (Section 3) is a 10-approximation algorithm for the data placement problem. The algorithm we present here is an improvement over the approximation algorithm described in [3]. For the benefit of the reader, we briefly sketch the differences between this algorithm and the one in [3] in Section 1.1.

Our algorithm is based on rounding an optimal solution to a natural linear-programming (LP) relaxation of the problem. Observe that the placement problem for each individual object is a UFL-instance; however these instances are coupled due to the cache-capacity constraints, which is what makes the problem hard. One of the main technical challenges faced in the rounding is to preserve the cache capacities while losing only a constant factor in the approximation ratio. Despite the similarity with UFL, hard capacities make it quite difficult to apply the standard rounding ideas underlying the design of approximation algorithms for UFL. All LP-based algorithms for UFL employ either filtering [26, 37], or use the dual to bound the solution cost [8, 18, 17, 6]. Filtering typically involves blowing up the LP-variables, thereby violating the cache-capacities, and the dual of the LP-relaxation of the data placement problem contains negative variables (corresponding to the primal capacity constraints), which presents a serious obstacle to using the dual to bound the solution cost. Instead, we use the techniques developed by Charikar, Guha, Tardos and Shmoys [7] for the k-median problem.

Our algorithm proceeds in two phases. In the first phase, we build upon a clustering method introduced by Charikar et al. (Step 1 in [7]) and round the LP-solution to a *half-integral* solution. In the second phase of our algorithm, we use the Shmoys-Tardos-Aardal [37] clustering method *without any filtering* to cluster the demand-nodes for each object and obtain a solution with the property that for every object o and cache i, there is at most one demand-node for o that is served by i. The key observation that allows us to do away with the problematic filtering step is that, in a half-integral solution, the distance between a client and any cache serving it fractionally is already bounded relative to its access cost in the half-integral solution. Once we have the aforementioned property, one can view the fractional solution as a feasible flow to a minimum-cost flow problem with integral capacities. By the integrality property of flows one can now extract an integer solution of no greater cost. This algorithm and its analysis are described in Section 3.

The formulation of the data placement problem appears most suitable for applications where objects are rarely written. In a setting where write-requests are issued for data objects, one needs to have a separate mechanism to maintain consistency among the replicas of an object. In Section 4, we formulate the *connected data placement problem*, which incorporates this aspect of data management (which is not captured by the data placement problem). As proposed by Krick et al. [22] in the context of another caching problem, we model this by requiring that for every object o, all caches containing o be connected via a Steiner tree T_o to a root r_o. When a write-request is issued for object o, the root initiates an update of all the copies of object o using the tree T_o as a multicast tree. The objective is to minimize the total cost incurred in storing and accessing objects and building the Steiner trees, that is, to minimize $\sum_{i\in\mathcal{F}}\sum_{o\in\mathcal{O}(i)} f_i^o + \sum_{j\in\mathcal{D}} d_j c_{i(j)j} + \sum_{o\in\mathcal{O}} M_o \sum_{e\in T_o} c_e$, where the M_os are input scaling parameters. This generalizes the *connected facility location* problem [14, 39, 15] for which the best known guarantee is 8.55 [39]. We present a 14-approximation algorithm for this problem. One noteworthy feature here is the ease with which one can interface the algorithm developed in Section 3 (which handles the data-placement part of the problem) with the rounding ideas proposed in [32, 14] to handle the connectivity-aspect of the problem.

In Section 5, we consider a couple of extensions. First, we consider the k-median variant, where for every object o, there is a bound of k_o on the number of caches that may store object o. Our rounding algorithm is versatile and extends easily with minimal changes to this variant, yielding the same approximation guarantee. Second, we consider the data placement problem with arbitrary object-lengths. It is easy to show (see Section 6) via a reduction from the PARTITION problem that with arbitrary object-lengths, it is *NP*-complete to even decide if there exists a feasible solution; hence, no approximation ratio is achievable in polynomial time unless *P* =*NP*. We can modify our algorithm to obtain a bicriteria approximation guarantee in this setting: we return a placement of cost at most 10 times the optimal where the total length of objects stored in a cache may exceed its capacity by the maximum object-length. We conclude in Section 6 with a couple of hardness results about the data placement problem with (i) no storage costs, and (ii) non-uniform object lengths.

Related work. The problem of data management in a distributed network has been extensively-studied. Dowdy and Foster [10] initiated the study of cooperative caching in the context of allocating files in a distributed network, and this problem has since received much attention. We limit ourselves to an overview of the work in models that most closely resemble our model; the reader is referred to the surveys [10, 12] for a more detailed discussion.

Various works [5, 2, 1] have considered an *online* version of our problem, both with and without cache capacities, where read and write requests arrive online and have to be taken care of on the fly. The competitive ratios achievable in the online setting are, not surprisingly, weaker than the approximation ratios achievable in the offline setting. Awerbuch, Bartal, and Fiat [2] gave a randomized algorithm with competitive ratio $\text{polylog}(\sum_{i\in\mathcal{F}} u_i)$ for the uniform metric, whereas Awerbuch et al. [1] give a $\text{polylog}(\max_{ij} c_{ij})$-competitive algorithm for arbitrary metrics, but require a $\text{polylog}(\max_{ij} c_{ij})$-factor blow-up in the cache-capacities. Various studies have incorporated routing information into the caching problem, for instance by having intermediate nodes cache copies of an object when the object is being routed [16, 31, 41], or by considering the problem of minimizing network congestion due to routing of requests [27, 28]. In contrast, we abstract away routing concerns by assuming that the c_{ij}-values, which determine the access costs, are given to us as input.

The *offline* data placement problem that we consider was first studied for hierarchical networks, or equivalently when the access costs form an ultrametric (a more restricted class of metrics). Leff, Wolf and Yu [23] considered ultrametrics derived from a star, Korupolu, Plaxton and Rajaraman [21] gave exact and approximation algorithms for general ultrametrics, and Korupolu and Dahlin [19] evaluated the practical performance of several placement algorithms for ultrametrics. Independent of [3], Meyerson, Munagala, and Plotkin [29] considered a generalization of our problem (called the page-placement problem), where a cache also has a client-capacity limiting the number of clients that may be assigned to it. They gave a constant-approximation, but with a logarithmic violation of both the client-capacities and the object-capacities. Subsequently, Guha and Munagala [13] obtained a constant-factor approximation where the capacities are violated only by a constant factor. Fleischer, Goemans, Mirrokni, and Sviridenko [11] considered a maximization version of the data placement problem with similar client-capacity constraints that limit the total demand that may be assigned to a cache. They give a $\left(1-\frac{1}{e}-\epsilon\right)$-approximation algorithm, for any $\epsilon > 0$, and show that no better guarantee is achievable unless *NP* $\subseteq$DTIME$\left[n^{O(\log\log n)}\right]$.

As mentioned earlier, the data placement problem can also be motivated from a facility-location perspective, where caches correspond to facilities and the objects correspond to the different *services* required by clients. Shmoys, Swamy, and Levi [35] formulated a closely related problem in this context called *facility location with service installation costs (FLSIC)*. Using the terminology of the data placement problem, in FLSIC the caches (facilities) are uncapacitated, but one has to pay a location-dependent cache-setup (facility-opening) cost to "build" a cache at a location before storing any data object at that cache. Independently, Ravi and Sinha [33] proposed the multicommodity facility location problem giving a similar motivation. Shmoys et al. [35] give a 6-approximation algorithm for FLSIC under a certain assumption on the service installation costs.

The data placement problem (without storage costs) and FLSIC have also been studied for the special case of the *directed* line-metric under the names of *broadcast scheduling* [4] and the *joint replenishment problem* [24] respectively. In both problems, both the clients and the caches correspond to points on the time-line. In broadcast scheduling, the objects correspond to pages. A client corresponds to a request for a page, a cache corresponds to a page-broadcast, and a request at time t must be assigned to a broadcast of that page at some time $t' > t$. At most c pages may be broadcast at any time; the goal is to minimize the average response time of the requests. The best-known approximation factor for this problem is $O\left(\frac{\log^2 |\mathcal{O}|}{\log\log |\mathcal{O}|}\right)$ due to Bansal et al. [4]. In the joint replenishment problem, the objects are items. Demands for items occur at various points of time, and one has to determine which items to order at which times, so that all demand can be met by orders that are placed at earlier points of time. Placing an order for a subset of items incurs a joint ordering cost to start the order, and an item-dependent cost, and each demand gets charged the cost incurred to hold the inventory for that demand. Levi et al. [24] gave a 2-approximation algorithm for this problem.

The data placement problem is a generalization of UFL, which corresponds to the special case with only one object. There is a large body of literature that deals with designing approximation algorithms for metric UFL; see [34] for a survey of this and earlier work. The first constant approximation guarantee for UFL was obtained by Shmoys, Tardos, and Aardal [37] via an LP-rounding algorithm, and the current state-of-the-art is a 1.5-approximation algorithm due to Byrka [6]. For the closely-related *k-median problem*, the first constant-factor approximation algorithm was given by Charikar, Guha, Tardos and Shmoys [7] using LP rounding. As mentioned earlier, the clustering method developed by them plays a key role in our algorithm.

Finally, we remark that the presence of cache capacities might suggest a similarity to the capacitated facility location (CFL) problem, but this resemblance is only superficial. The capacities in our problem limit the number of objects that may be assigned to a cache, but there is *no bound* on the demand (or number) of clients requesting a given object, or total demand, that may be assigned to a cache. (The data placement problem may be viewed as a collection of *UFL instances*, one for each object, that are coupled due to the cache-capacity constraints.) In particular, as our algorithm shows, the integrality gap of the natural LP relaxation for our problem is at most a constant, whereas there is no known LP relaxation of CFL with constant integrality-gap. Moreover, the local search algorithms in [20, 9, 30, 42] do not directly apply, and it is not clear if they can be adapted to our problem.

1.1 Relationship with the work of [3] and [38]

This work is a merger of two earlier papers: an extended abstract of Baev and Rajaraman [3], and an unpublished manuscript of Swamy [38]. In order to place our work in proper bibliographic context, and for the benefit of the reader, we include a brief comparison of our work with [3] and [38].

The data placement problem that we consider was introduced by Baev and Rajaraman [3]. (In their model, each client j has demand d_{jo} for every object $o \in \mathcal{O}$; this easily reduces to the model considered here since one can create a co-located copy $j^{(o)}$ of client j with demand d_{jo} for every object o.) They gave a 20.5-approximation algorithm for this problem, which is based on rounding an optimal solution to the same LP-relaxation of the problem that we consider. The 10-approximation algorithm described in this paper is from [38], and is based on an improved rounding procedure for the same LP. We briefly describe the main differences between the two algorithms.

As described earlier, our algorithm proceeds in two phases. The first phase of our algorithm, where we round the LP-solution to a *half-integral* solution, is identical to the first half (steps 1–3) of the algorithm in [3]. From here on the two algorithms proceed along different tracks. In both algorithms, the goal is to modify the previously-obtained half-integral solution into one that has the property that for every object o and cache i, there is at most one demand-node for o that is served by i, so that one can then set up a minimum-cost flow problem to round the half-integral solution to an integral one. In the second phase of our algorithm, we use the Shmoys-Tardos-Aardal [37] clustering method (without filtering) to obtain a solution with the above property. In contrast, the Baev-Rajaraman algorithm dovetails the rounding procedure of [7] (creating 1-level trees that are used to cluster the clients) to obtain a solution with the aforementioned property. By adopting a different clustering approach that better exploits half-integrality, we obtain a simpler algorithm that also yields a much better approximation guarantee.

The connected data placement problem was introduced by Swamy [38], and the 14-approximation algorithm that we present for this problem was described therein.

2 An LP relaxation

We can express the data placement problem as an integer program and relax the integrality constraints to get a linear program. Throughout we will use i to index the caches in $\mathcal{F}$, j to index the clients in $\mathcal{D}$ and o to index the objects in $\mathcal{O}$.

$$\begin{aligned}
\min \quad & \sum_i \sum_o f_i^o y_i^o + \sum_j \sum_i d_j c_{ij} x_{ij} & & \text{(P)} \\
\text{s.t.} \quad & \sum_i x_{ij} \geq 1 & \forall j & \\
& x_{ij} \leq y_i^{o(j)} & \forall i, j & \\
& \sum_o y_i^o \leq u_i & \forall i & \quad (1) \\
& x_{ij}, y_i^o \geq 0 & \forall i, j, o. &
\end{aligned}$$

Variable y_i^o indicates if object o is stored in cache i and x_{ij} indicates if client j is assigned to cache i. The first and second constraints say that each client must be assigned to a cache and if client j is assigned to cache i then object $o(j)$ must be stored in cache i. The third constraint states that the total length of items stored in any cache i is at most its capacity u_i. An integer solution corresponds exactly to a solution to our problem. We let G_o denote the set of clients that demand object o, i.e., $G_o = \{j : o(j) = o\}$.

3 The rounding procedure

Let (x, y) denote the optimal solution to (P) and OPT be its value. We will round this to an integer solution losing a factor of at most 10. We use the terms access cost and assignment cost interchangeably.

3.1 Overview of the algorithm

We first give a high level description of the algorithm. Suppose for a moment that the optimal solution (x, y) satisfies the following property: for any cache i and object o, there is *at most one* client $j \in G_o$ such that $x_{ij} > 0 \ (*)$. We can then set up the following min-cost flow problem: create a bipartite graph with vertex set $\mathcal{D} \cup \mathcal{F}$ and edges (i, j) for every i, j such that $x_{ij} > 0$ with cost $f_i^{o(j)} + d_j c_{ij}$ and capacity 1; client j has a demand of 1 and cache i has capacity u_i. The LP solution translates to a feasible fractional flow in this graph of cost at most OPT. *Note that property $(*)$ is crucial for this.* Conversely an integer flow yields an integer solution to (P) of cost equal to the flow cost. Therefore by the integrality property of flows (given integer capacities) we can round (x, y) to an integer solution of no greater cost. Of course, the LP solution need not have property $(*)$ so our goal will be (loosely speaking) to transform (x, y) to a solution that has property $(*)$ without increasing the cost by much. One of the major challenges encountered is to do this transformation *without violating the cache capacities*, while increasing the cost only by a constant factor.

Roughly speaking we want to do the following: for each object o, cluster the clients in G_o around certain 'centers' (also clients in G_o) such that (a) every client k is mapped to a "nearby" cluster center j whose LP assignment cost is less than that of k, and (b) the facilities serving the cluster centers in the fractional solution (x, y) are disjoint. So, the modified instance where the demand of a client is moved to the center of its cluster has a fractional solution, namely, the solution induced by (x, y), that satisfies $(*)$ and has cost at most OPT.

Furthermore, given a solution to the modified instance we can obtain a solution to the original instance losing a small additive factor. This clustering-idea lies at the core of most algorithms for facility location, however the necessity of preserving cache-capacities renders many of the known clustering methods [37, 8, 25] unsuitable for our purposes. For example, one option is to use the decomposition method of Shmoys et al. [37] that produces precisely such a clustering. The problem however is that [37] uses filtering which involves blowing up the x_{ij} and y_i^o values and thus violating the cache capacities. Chudak and Shmoys [8], and Levi, Shmoys and Swamy [25] use similar clustering ideas but without filtering, using the dual solution to bound (portions of) the cost. The difficulty here in bounding the cost using the dual solution is that there are terms with negative coefficients in the dual objective function that correspond to the primal capacity constraints (1). Although [40, 25] showed that it is possible to overcome this difficulty in certain cases, the situation here looks more complicated and it is not clear how to use their techniques.

Instead, we use the clustering technique of Charikar et al. [7] developed for the k-median problem. Our algorithm proceeds in two phases. In the first phase (Section 3.2), we extract a modified instance and a fractional solution to this instance from the LP solution, and round this to a *half-integral* solution $(\hat{x}, \hat{y})$, that is, each $\hat{x}_{ij}, \hat{y}_i \in \{0, \frac{1}{2}, 1\}$, losing a factor of 3. Further, any solution here will give a solution to the original instance while increasing the cost by at most $4 \cdot OPT$. We do this by first transferring demands to certain well-separated centers (Step I) exactly as in the demand-consolidation step of [7], so as to ensure that each center has its own private set of caches that serve it to an extent of at least half. This allows us to set up a minimum-cost flow problem (Step II) with half-integral capacities with a one-one correspondence between solutions and flows, and thereby round the fractional solution on the centers to a half-integral solution.

In phase two (Section 3.3), we observe that we can now use the clustering method in [37] on the half-integral solution $(\hat{x}, \hat{y})$ *without any filtering* (Step III) since such a solution is essentially already filtered: if client j is assigned to i and i' in $\hat{x}$, then $c_{ij}, c_{i'j} \leq 2(c_{ij}\hat{x}_{ij} + c_{i'j}\hat{x}_{i'j})$. This clustering satisfies the requirements (a) and (b) mentioned above. Thus, one can obtain an integer solution for the new cluster centers by solving a suitable min-cost flow problem. This is essentially what we do, but we set up the min-cost flow problem more carefully (Step IV) so as to lose only a factor of 2 in converting $(\hat{x}, \hat{y})$ to an integer solution (for the modified instance extracted in phase 1). So overall we get an approximation ratio of $4 + 2 \times 3 = 10$ (Theorem 3.5).

We now describe each of these steps in detail. Let $\bar{C}_j = \sum_i c_{ij}x_{ij}$ denote the cost incurred by the LP solution to assign one unit of demand of client j.

3.2 Obtaining a half-integral solution $(\hat{x}, \hat{y})$

Step I: Consolidating demands around centers. We first consider every object o separately, and consolidate (or cluster) the demand of clients in G_o at certain clients, that we call *cluster centers*. We do not modify the fractional solution (x, y) but only modify the demands so that for some clients j, the demand d_j is "moved" to a "nearby" center k. We assume every client has a non-zero demand.

Set $d'_j \leftarrow 0$ for every j. Consider the clients in G_o in increasing order of $\bar{C}_j$. For each client j, if there exists a client k (including j) such that $d'_k > 0$ and $c_{jk} < 4\max(\bar{C}_j, \bar{C}_k) = 4\bar{C}_j$, set $d'_k \leftarrow d'_k + d_j$, otherwise set $d'_j \leftarrow d_j$. We do this for every object o. Let $D_o = \{j \in G_o : d'_j > 0\}$ and $D = \bigcup_o D_o$. Each client in D is a cluster center. Let $OPT' = \sum_{i,s} f_i^o y_i^o + \sum_{j \in D, i} d'_j c_{ij} x_{ij}$ denote the cost of (x, y) for the modified instance consisting of the cluster centers. Since the demand of each client $k \notin D$ moves a distance of at most $4\bar{C}_k$, it is clear that any solution to the modified instance yields a solution for client-set $\mathcal{D}$ incurring an additive factor of at most $4\sum_{k \notin D} d_k \bar{C}_k \leq 4 \cdot OPT$. We obtain the following lemma.

Lemma 3.1 *The following hold: (i) if $j, k \in D_o$, then $c_{jk} \geq 4\max(\bar{C}_j, \bar{C}_k)$, (ii) $OPT' \leq OPT$, and (iii) any solution (x', y') to the modified instance can be converted to a solution to the original instance incurring an additional cost of at most $4 \cdot OPT$.*

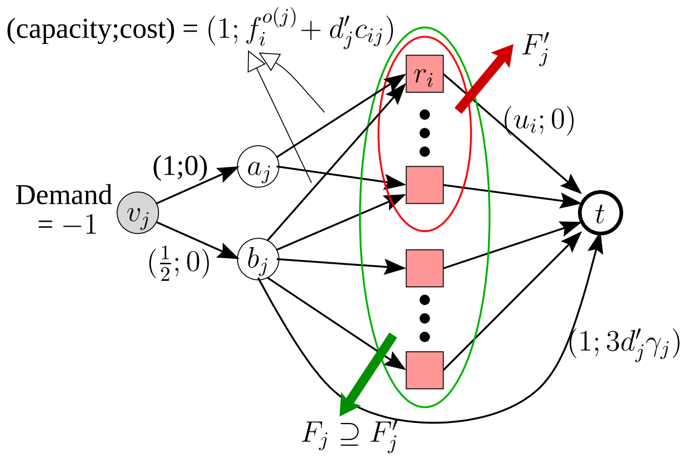


Figure 3.1: The min-cost flow network constructed in Step II. The tuple labeling an edge gives the (capacity;cost) for the edge.

From now on we will focus on the modified instance with client-set D and modified demands d'_j. At the very end we will use the above lemma to translate an integer solution to the modified instance to an integer solution to the original instance.

Step II: Transforming to a half-integral solution. We define the cluster of a client $j \in D_o$ to consist of all clients $k \in G_o$ whose demand d_k was moved to j, and a set of facilities F_j. F_j consists of all facilities i to which j is fractionally assigned such that j is the center in D_o closest to i, that is, $F_j = \{i : x_{ij} > 0 \text{ and } c_{ij} = \min_{k \in D_o} c_{ik}\}$, with ties broken arbitrarily. Let $F'_j \subseteq F_j = \{i \in F_j : c_{ij} \leq 2\bar{C}_j\}$. Define γ_j to be $\min_{i \notin F_j : x_{ij} > 0} c_{ij}$. Clearly the sets F_j for $j \in D_o$ are disjoint. By property (i) of Lemma 3.1, we have that F_j contains all the facilities i such that $x_{ij} > 0$ and $c_{ij} \leq 2\bar{C}_j$. So $\sum_{i \in F'_j} x_{ij} = \sum_{i : c_{ij} \leq 2\bar{C}_j} x_{ij} \geq \frac{1}{2}$ where the last inequality follows from Markov's inequality.

In the half-integral solution $(\hat{x}, \hat{y})$, we will store object o only at caches that lie in some set F_j for $j \in D_o$. To obtain $(\hat{x}, \hat{y})$, we set up a min-cost flow problem. We create a sink t and a node r_i for every cache i in $\bigcup_{j \in D} F_j$ with an outgoing edge (r_i, t) of capacity u_i and cost 0 (see Fig. 3.1). For each client $j \in D$ we create three nodes v_j, a_j, and b_j. Node v_j has demand -1 (i.e., the net outgoing flow should be 1) to denote the requirement that j must be assigned to a cache. We add edges (v_j, a_j) with capacity 1 and cost 0, and (v_j, b_j) with capacity $\frac{1}{2}$ and cost 0. Node a_j represents the option that j is assigned to a facility in F'_j, so we add edges (a_j, r_i) to every $i \in F'_j$ with capacity 1 and cost $f_i^{o(j)} + d'_j c_{ij}$. Notice that setting the capacity of (v_j, b_j) to $\frac{1}{2}$ forces j to be assigned to an extent of at least $\frac{1}{2}$ to facilities in F'_j. Node b_j signifies that j is assigned either to a facility in F_j or to some other facility. To encode this, we add edges (b_j, r_i) to every $i \in F_j$ with capacity 1 and cost $f_i^{o(j)} + d'_j c_{ij}$, and an edge (b_j, t) with capacity 1 and cost $3d'_j \gamma_j$ (since, as we show later, there is always a facility at distance at most $3\gamma_j$ from j that is at least half-open). Figure 3.1 shows the portion of the min-cost flow instance corresponding to client j.

Since all edge capacities are $\frac{1}{2}$ or 1 the network has a half-integral min-cost flow. Given such a flow we obtain $(\hat{x}, \hat{y})$ as follows. We initialize all $\hat{x}_{ij}$, $\hat{y}_i^o$ to 0. Consider object o. For every $j \in D_o$ and cache $i \in F'_j$, we set $\hat{y}_i^o = \hat{x}_{ij} =$ flow along $(a_j, r_i) +$ flow along (b_j, r_i). For every $i \in F_j \setminus F'_j$, we set $\hat{y}_i^o$ and $\hat{x}_{ij}$ equal to the flow along edge (b_j, r_i). Observe that there is at least one cache $i \in F'_j$ such that $\hat{x}_{ij} > 0$; we call the cache in F'_j closest to j with $\hat{x}_{ij} > 0$ the *primary cache* of j. Note that since the sets F_j (and hence F'_j) for $j \in D_o$ are disjoint, every client in D_o has a unique primary cache i. Let i' be the cache nearest to j, other than its primary cache, with $\hat{y}_{i'}^o > 0$. If edge (b_j, t) carries positive flow (so no edge (b_j, r_i) carries any flow implying that $\hat{y}_i^o = 0$ for every $i \in F_j \setminus F'_j$), we set $\hat{x}_{i'j} =$ flow on $(b_j, t) = \frac{1}{2}$. We do this for every object o. If a client j is assigned to a cache other than its primary cache, we call the other cache the *secondary cache* of j. It is easy to verify that $(\hat{x}, \hat{y})$ is a feasible solution to (P), where the client-set is D. The following lemma shows that the cost of $(\hat{x}, \hat{y})$ is at most $3 \cdot OPT$.

Lemma 3.2 *The cost of* $(\hat{x}, \hat{y})$, *that is,* $\sum_{i,o} f_i^o \hat{y}_i^o + \sum_{j \in D, i} d'_j c_{ij} \hat{x}_{ij}$, *is at most* $3 \cdot OPT' \leq 3 \cdot OPT$.

Proof : First we show that (x, y) induces a flow of cost at most $3 \cdot OPT'$, so the cost of the min-cost flow is no greater. Then we show that the cost of $(\hat{x}, \hat{y})$ is bounded by the cost of the min-cost flow.

Consider the following flow: each edge (v_j, a_j) has flow $\sum_{i \in F'_j} x_{ij}$, (v_j, b_j) has flow $1 - \sum_{i \in F'_j} x_{ij}$, and (b_j, t) has flow $1 - \sum_{i \in F_j} x_{ij}$; every edge (a_j, r_i) or (b_j, r_i) has flow x_{ij}; the flow on (r_i, t) is $\sum_o \sum_{j \in D_o : i \in F_j} x_{ij}$. It is easy to see that this is a feasible flow. The cost of this flow is

$$\sum_{o, j \in D_o} \Big(\sum_{i \in F_j} (d'_j c_{ij} + f_i^o) x_{ij} + 3 d'_j \gamma_j (1 - \sum_{i \in F_j} x_{ij}) \Big) \leq \sum_{i,o} f_i^o y_i^o + \sum_j d'_j \Big(\sum_{i \in F_j} c_{ij} x_{ij} + 3 \gamma_j (1 - \sum_{i \in F_j} x_{ij}) \Big).$$

We have $OPT' = \sum_{i,o} f_i^o y_i^o + \sum_j d'_j \bar{C}_j$. For any $j \in D$, $\bar{C}_j = \sum_{i \in F_j} c_{ij} x_{ij} + \sum_{i \notin F_j} c_{ij} x_{ij} \geq \sum_{i \in F_j} c_{ij} x_{ij} + \gamma_j (1 - \sum_{i \in F_j} x_{ij})$ since γ_j was defined as $\min_{i \notin F_j : x_{ij} > 0} c_{ij}$. This shows that the cost of the constructed flow, and hence of the min-cost flow, is at most $3 \cdot OPT'$.

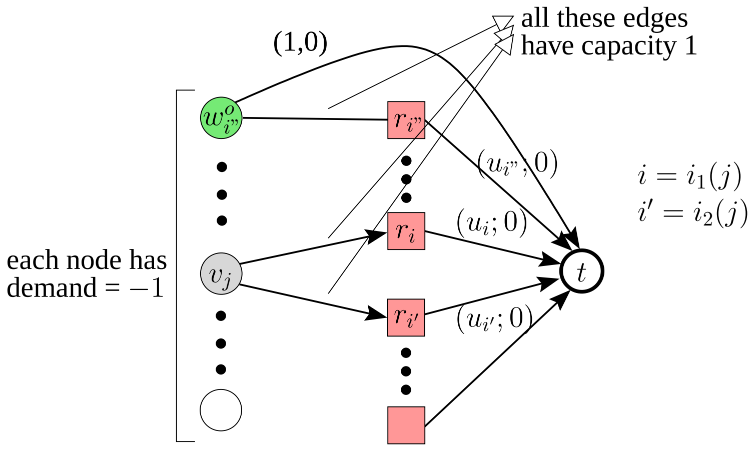
Now consider the solution $(\hat{x}, \hat{y})$ induced by the half-integral min-cost flow. By construction, the quantity $\sum_{i,o} f_i^o \hat{y}_i^o + \sum_{j \in D, i \in F_j} d'_j c_{ij} \hat{x}_{ij}$ is exactly equal to the total cost of the flow on edges (a_j, r_i) and (b_j, r_i). For any $j \in D$ the remaining cost $\sum_{i \notin F_j} d'_j c_{ij} \hat{x}_{ij}$ is equal to $d'_j c_{i'j} \cdot$ (flow on (b_j, t)) where i' is the secondary cache of j. So it suffices to show that $c_{i'j} \leq 3\gamma_j$. Let $\gamma_j = c_{i''j}$ where $i'' \notin F_j$ and $x_{i''j} > 0$. Let k be the center in D_o nearest to i'' and let ℓ be the primary cache of k. Then, $c_{i'j} \leq c_{\ell j}$ and $4 \max(\bar{C}_j, \bar{C}_k) \leq c_{jk} \leq c_{i''j} + c_{i''k} \leq 2\gamma_j$. Also $c_{\ell k} \leq 2\bar{C}_k$ since $\ell \in F'_k$. Combining the inequalities we get that $c_{i'j} \leq 3\gamma_j$ which completes the proof of the lemma. ■

3.3 Converting $(\hat{x}, \hat{y})$ to an integer solution $(\tilde{x}, \tilde{y})$

Define $\hat{C}_j = \sum_i c_{ij} \hat{x}_{ij}$ for $j \in D$. Let $i_1(j)$ denote the primary cache of j. For convenience, we will say that every client $j \in D$ has both a primary cache $i_1(j)$ and a secondary cache i' with $\hat{x}_{i_1(j)j} = \hat{x}_{i'j} = \frac{1}{2}$, with the understanding that if j does not have a secondary cache then i' is a copy of $i_1(j)$, so effectively $\hat{x}_{i_1(j)j} = 1$. We denote the secondary cache by $i_2(j)$. Then we have, $\hat{C}_j = \frac{1}{2}(c_{i_1(j)j} + c_{i_2(j)j})$, $c_{i_1(j)j} \leq \hat{C}_j$ and $c_{i_1(j)j} \leq c_{i_2(j)j} \leq 2\hat{C}_j$. Notice that $i_1(j)$ and $i_2(j)$ are the (one or) two caches with $\hat{y}_i^{o(j)} > 0$ that are nearest to j. Let $L_o = \{i : \hat{y}_i^o > 0\}$ and $L = \bigcup_o L_o$.

Step III: Clustering. First for every object o we cluster the clients in D_o as follows: pick $j \in D_o$ with smallest $\hat{C}_j$. Remove every client $k \in D_o$ such that both j and k are (fractionally) assigned to a cache $i \in L_o$, and recurse on the remaining set of clients until no client in D_o is left. Let D'_o be the set of clients picked for object o and let $D' = \bigcup_o D'_o$; D' denotes the *new cluster centers*. It is clear that for any cache in L_o at most one client in D'_o is assigned to it. Observe that for every client $k \in D_o \setminus D'_o$ there is some $j \in D'_o$ such that $\hat{C}_j \leq \hat{C}_k$ and $\hat{x}_{ij}, \hat{x}_{ik} > 0$ for some $i \in L_o$, implying that $c_{jk} \leq 4\hat{C}_k$. We call j the *center* of k and denote it by $\mathsf{ctr}(k)$.

Now for every client $k \in D \setminus D'$ we can move its demand d'_k to $j = \mathsf{ctr}(k)$. The resulting instance with client-set D' (and the new demands) satisfies the property $(*)$ mentioned in Section 3.1. Hence, one can set up a min-cost flow problem as mentioned in Section 3.1 to get an integer solution to the instance with client-set D', which translates to a solution with client-set D (and the original demands d'_j). Doing



cost of (v_j, r_i) is $f_i^{o(j)} + d'_j c_{ij} + \sum_{k \in A_j} d'_k c_{ik}$
cost of $(v_j, r_{i'})$ is $f_{i'}^{o(j)} + d'_j c_{i'j} + \sum_{k \in A_j} d'_k c_{i'k} + \sum_{k \in B_j} d'_k (c_{i'k} - c_{i_2(k)k})$
cost of $(w^o_{i"}, r_{i"})$ is $f^o_{i"} + \sum_{k \in B^o_{i"}} d'_k (c_{i"k} - c_{i_2(k)k})$

Figure 3.2: The min-cost flow network constructed in Step IV. The tuple labeling an edge gives the (capacity;cost) for the edge.

this naively, we lose an additive factor of (at most) $4\sum_{k\in D} d'_k \hat{C}_k$ in translating the demands back from D' to D. We will set up the min-cost flow network more carefully so that we only lose a multiplicative factor of 2 in rounding $(\hat{x}, \hat{y})$ to an integer solution for the client-set D. We want to capture the following observation: suppose the demand of a client $k \in D_o$ is moved to $j = \mathsf{ctr}(k)$. Let $\hat{x}_{ik} = \hat{x}_{i'k} = \frac{1}{2}$ and $\hat{x}_{ij} = \hat{x}_{i''j} = \frac{1}{2}$. The per-unit-demand assignment cost of k is at most $\frac{1}{2}(c_{ik} + c_{i''k}) \le c_{ik} + \hat{C}_j \le 3\hat{C}_k$ (since $c_{i''k} \le c_{i''j} + c_{ij} + c_{ik}$), which is much less than the naive bound of $4\hat{C}_k + \hat{C}_j$.

Step IV: The min-cost flow network. Fix $o \in \mathcal{O}$ and consider a client $j \in D'_o$. We will maintain two sets A_j and B_j for j. Let $i = i_1(j)$ and $i' = i_2(j)$ be the primary and secondary caches of j. We define $A_j = \{k \in D_o : \mathsf{ctr}(k) = j\}$, and $B_j = \{k \in D_o : \mathsf{ctr}(k) \neq j \text{ and } i' = i_1(k)\}$. Also, for every cache $i \in L_o$ such that $\hat{x}_{ij} = 0$ *for every* $j \in D'_o$, we define $B^o_i = \{k \in D_o : i = i_1(k)\}$ (which is either empty or a singleton). Note that all the sets A_j, B_j and B^o_i are subsets of $D_o \setminus D'_o$.

We create a sink t, and a node r_i for every $i \in L$ for which $\hat{x}_{ij} > 0$ for some $j \in D'$, or $B^o_i \neq \phi$ for some o (see Fig. 3.2). We have an edge (r_i, t) of capacity u_i and cost 0. For every client $j \in D'$ we create a node v_j. Further, for every $i \in L, o \in \mathcal{O}$ with $B^o_i \neq \phi$ we create a node w^o_i. The nodes v_j and w^o_i all have demand -1. For every node v_j we have edges (v_j, r_i) to every i with $\hat{x}_{ij} > 0$, and we have edges (w^o_i, r_i), (w^o_i, t) for every node w^o_i. All these edges have capacity 1. The cost of these edges is set as follows. Consider a node v_j and let $i = i_1(j), i' = i_2(j)$. We set the cost of (v_j, r_i) to $f_i^{o(j)} + d'_j c_{ij} + \sum_{k\in A_j} d'_k c_{ik}$ and the cost of $(v_j, r_{i'})$ to $f_{i'}^{o(j)} + d'_j c_{i'j} + \sum_{k\in A_j} d'_k c_{i'k} + \sum_{k\in B_j} d'_k (c_{i'k} - c_{i_2(k)k})$. We set the cost of (w^o_i, r_i) to $f^o_i + \sum_{k\in B^o_i} d'_k (c_{ik} - c_{i_2(k)k})$ and the cost of (w^o_i, t) to 0; see Figure 3.2.

Since all capacities are integer, there is an integer min-cost flow. We map this to an integer solution $(\tilde{x}, \tilde{y})$ to the instance with client-set D. Set $\tilde{x}_{ij}, \tilde{y}_i \leftarrow 0$ for all i, j. Consider object o. First, for every $j \in D'_o$ and $i \in \{i_1(j), i_2(j)\}$, we set $\tilde{x}_{ij}$ = flow on edge (v_j, r_i). For every client $k \in B_j$ we set $\tilde{x}_{i_2(j)k} = \tilde{x}_{i_2(j)j}$, and for every $k \in B^o_i$ we set $\tilde{x}_{ik}$ = flow on (w^o_i, r_i). Next, for every $j \in D'_o$ and every $k \in A_j$ that has not yet been assigned (i.e., $\sum_i \tilde{x}_{ik} = 0$), we set $\tilde{x}_{ik} = \tilde{x}_{ij}$ for $i \in \{i_1(j), i_2(j)\}$. Finally, set $\tilde{y}^o_i = \max_{j\in D_o} \tilde{x}_{ij}$. We do this for every o. Observe that $\tilde{y}^o_i = 1$ for at most one facility from F'_j for every client $j \in D_o$. This will be useful in Section 4. It is easy to see that $(\tilde{x},\tilde{y})$ is a feasible integer solution. We now bound its cost.

Lemma 3.3 *The cost of the min-cost flow in the network is at most twice the cost of* $(\hat{x},\hat{y})$.

Proof : We exhibit a fractional flow of cost at most the claimed cost. The fractional flow is obtained by setting the flow on every edge (v_j,r_i) to $\hat{x}_{ij}$, and the flow on (w_i^o,t) and (w_i^o,r_i) to $\max_{k\in B_i^o}\hat{x}_{ik}=\frac{1}{2}$, where the equality follows since every $k\in B_i^o$ is assigned to an extent of $\frac{1}{2}$ to $i_2(k)\neq i$. The flow on the edges (r_i,t) is set accordingly to $\sum_o\bigl(\sum_{j\in D'_o}\hat{x}_{ij}+\max_{k\in B_i^o}\hat{x}_{ik}\bigr)$. This is a feasible flow since for every i,o, either $B_i^o=\phi$ and there is exactly one $j\in D'_o$ such that $\hat{x}_{ij}>0$, or $B_i^o\neq\phi$ and $\hat{x}_{ij}=0$ for every $j\in D'_o$. So $\sum_{j\in D'_o}\hat{x}_{ij}+\max_{k\in B_i^o}\hat{x}_{ik}$ is at most y_i^o.

The cost of an edge (v_j,r_i) or (w_i^o,r_i) consists of a storage component ($f_i^{o(j)}$ or f_i^o) and an assignment component that can be attributed to various clients. We call the contribution of the storage components to the flow cost the *flow storage cost*, and the contribution of the assignment components the *flow assignment cost*. The flow storage cost is $\sum_{i,o}f_i^o\bigl(\sum_{j\in D'_o}\hat{x}_{ij}+\max_{k\in B_i^o}\hat{x}_{ik}\bigr)\leq\sum_{i,o}f_i^oy_i^o$ by the above reasoning. To evaluate the flow assignment cost we consider the contribution of each client to the assignment components separately. Fix an object o. First consider $j\in D'_o$ with $i=i_1(j),\ i'=i_2(j)$. Client j only figures in the assignment component of (v_j,r_i) and $(v_j,r_{i'})$ and its contribution is $d'_j(c_{ij}\hat{x}_{ij}+c_{i'j}\hat{x}_{i'j})=d'_j\hat{C}_j$. A client $k\in D_o\setminus D'_o$ is in exactly one set A_j where $j=\mathsf{ctr}(k)$ and may possibly also lie in one of the sets $B_{j'}$ or $B_{i''}^o$. Let $i=i_1(j)$ and $i'=i_2(j)$.

1. If k does not lie in any set $B_{j'}$ or $B_{i''}^o$, then it must be that $\hat{x}_{i_1(k)j}>0$. Client k contributes only to the assignment component of edges (v_j,r_i) and $(v_j,r_{i'})$ and this contribution is $d'_k(c_{ik}\hat{x}_{ij}+c_{i'k}\hat{x}_{i'j})\leq d'_k(c_{i_1(k)k}+\hat{C}_j)\leq 2d'_k\hat{C}_k$ since $\hat{x}_{ij}=\hat{x}_{i'j}=\frac{1}{2}$ and $c_{ik}+c_{i'k}\leq 2c_{i_1(k)k}+c_{ij}+c_{i'j}$.
2. Now suppose k is also in one of the sets $B_{j'}$ or $B_{i''}^o$, so it also contributes to the assignment component of an edge $(v_{j'},r_{i_1(k)})$ or an edge $(w_{i''}^o,r_{i''})$. The contribution in both cases is $\frac{d'_k}{2}(c_{i_1(k)k}-c_{i_2(k)k})$ since we must have $x_{i_1(k)j'}=\frac{1}{2}=x_{i''k}$. Adding the contributions to edges (v_j,r_i) and $(v_j,r_{i'})$, the total contribution is $\frac{d'_k}{2}(c_{ik}+c_{i'k}+c_{i_1(k)k}-c_{i_2(k)k})\leq d'_k(\hat{C}_k+\hat{C}_j)\leq 2d'_k\hat{C}_k$ since $c_{ik}+c_{i'k}\leq 2c_{i_2(k)k}+c_{ij}+c_{i'j}$.

So the flow assignment cost is at most $2\sum_{j\in D}d'_j\hat{C}_j$. Thus the total flow cost is at most $\sum_{i,o}f_i^o\hat{y}_i^o+2\sum_{j\in D}d'_j\hat{C}_j$ which is at most twice the cost of $(\hat{x},\hat{y})$. ■

Lemma 3.4 *The cost of the integer solution* $(\tilde{x},\tilde{y})$ *is at most the cost of the min-cost integer flow.*

Proof : Observe that for any o, $\sum_i f_i^o\tilde{y}_i^o=\sum_{i,j\in D'_o}f_i^o\tilde{x}_{ij}+\sum_{\text{nodes }w_i^o}f_i^o(\text{flow on }(w_i^o,r_i))$. So the total storage cost is $\sum_{e=(v_j,r_i)}f_i^{o(j)}(\text{flow on }e)+\sum_{e=(w_i^o,r_i)}f_i^o(\text{flow on }e)$ which is just the flow storage cost.

We will bound the assignment cost of a client by the contribution it makes to the flow assignment cost. Fix object o. Consider $j\in D'_o$. Let $i=i_1(j)$ and $i'=i_2(j)$. At most one of the edges (v_j,r_i), $(v_j,r_{i'})$ carries non-zero flow and we set $\tilde{x}_{ij},\tilde{x}_{i'j}$ equal to the flow on the corresponding edge. So the assignment cost of j is $d'_j\bigl(c_{ij}(\text{flow on }(v_j,r_i))+c_{i'j}(\text{flow on }(v_j,r_{i'}))\bigr)$, which is also the contribution of j to the assignment flow cost. The same argument holds for $k\in A_j$ if k is assigned to one of i or i'. The remaining case is when $k\in A_j$, and k is not assigned to i or i', but it is assigned to $i''=i_1(k)$ either because $k\in B_{j'}$ where $i''=i_2(j')$ and $(v_{j'},r_{i''})$ has non-zero flow, or because $k\in B_{i''}^o$ and $(w_{i''}^o,r_{i''})$ carries non-zero flow. The assignment cost of k is $d'_kc_{i''k}$. The contribution of k to the assignment flow cost is at least $d'_k(c_{i''k}-c_{i_2(k)k})+d'_k\min(c_{ik},c_{i'k})$ since $k\in A_j$. This is at least $d'_kc_{i''k}$ since both $c_{ik},c_{i'k}$ are at least $c_{i_2(k)}$. So the assignment cost of $(\tilde{x},\tilde{y})$ is bounded by the flow assignment cost. This completes the proof. ■

Combining Lemmas 3.1–3.4, we obtain that $(\tilde{x}, \tilde{y})$ yields an integer solution to the original instance of cost at most $10 \cdot OPT$. Thus, we obtain the following theorem.

Theorem 3.5 *There is a 10-approximation algorithm for the data placement problem.*

4 The connected data placement problem

The formulation of the data placement problem seems most suitable for applications where objects are rarely written. In the presence of write-requests, one needs to have a mechanism that ensures that all the copies of a data-object replicated in the various caches are consistent, and this requires that a write-request updates all the replicas of the data object. One way of modeling this, as proposed by Krick et al. [22], is to insist that all caches containing the same data object be interconnected via a Steiner tree, which would serve as a multicast tree that is used to update all copies of an object when a write-request is issued for it.

This gives rise to the *connected data placement problem*. We assume that there is a root $r_o \in \mathcal{D} \cup \mathcal{F}$ for each object o that issues the multicast message when a write-request is issued for o, and require that all caches containing object o be connected to r_o. Thus, our goal is to find a placement $\{\mathcal{O}(i)\}_{i \in \mathcal{F}}$ of objects to caches respecting the cache-capacity constraints, assign each client j to a cache $i(j)$ containing the object $o(j)$, and for each object o, connect the caches storing object o to r_o via a Steiner tree T_o, so as to minimize

$$\sum_{i \in \mathcal{F}} \sum_{o \in \mathcal{O}(i)} f_i^o + \sum_{j \in \mathcal{D}} d_j c_{i(j)j} + \sum_{o \in \mathcal{O}} M_o \sum_{e \in T_o} c_e.$$

Here $M_o \geq 1$ is an input scaling parameter, e.g., it might denote the total number of write-requests for object o.

The LP relaxation (P) is modified as follows. We introduce variables $z_e^o \geq 0$ for each object o, and each edge e (of the complete graph on $\mathcal{D} \cup \mathcal{F}$) that indicates (in the integer program) if edge e is part of the tree T_o. The objective function includes the additional term $\sum_o M_o \sum_e c_e z_e^o$. For each object o, set $S \subseteq \mathcal{D} \cup \mathcal{F}$ such that $r_o \notin S$, and client $j \in G_o$, we add the constraint $\sum_{e \in \delta(S)} z_e^o \geq \sum_{i \in S} x_{ij}$, where $\delta(S) = \{e = (u, v) : |S \cap \{u, v\}| = 1\}$. Although this LP has an exponential number of constraints, it can be solved efficiently via the ellipsoid method.

Observe that the connected data placement is a generalization of the *connected facility location* problem [14, 39, 15] (which is the special case with only one object) for which the best-known approximation guarantee is 8.55 [39]. However, due to the presence of cache capacities, it is not clear how to apply the primal-dual technique in [39], or the random-sampling idea in [15]. We show that the LP-rounding technique proposed in [32, 14] to handle such connectivity requirements can be overlaid almost directly on top of our rounding procedure from Section 3, to round an optimal solution to the above LP losing a factor of at most 14.

We briefly sketch the main steps. Let (x, y, z) be an optimal fractional solution, and $\bar{C}_j = \sum_i c_{ij} x_{ij}$. We slightly modify the demand-consolidation step (Step I) of our rounding procedure: we now move the demand of client k to client j (where $d'_j > 0,\ \bar{C}_j \leq \bar{C}_k$) if $c_{jk} < 8 \max(\bar{C}_j, \bar{C}_k)$. Recall that $F'_j = \{i : x_{ij} > 0,\ c_{ij} \leq 2\bar{C}_j\}$ and that the sets F'_j are disjoint for clients in D_o. Due to the above change, we lose an additive factor of $8 \sum_j d_j \bar{C}_j$ in translating a solution for client-set $D = \bigcup_o D_o$ to a solution for $\mathcal{D}$. More importantly, for any two facilities $i \in F'_j$ and $i' \in F'_k$ where $j, k \in D_o,\ j \neq k$, we now have that $c_{ii'} \geq 4 \max(\bar{C}_j, \bar{C}_k)$. The rest of the rounding process in Section 3 is unchanged. Thus, the sum of the storage costs and access costs is at most $14\big(\sum_{i,o} f_i^o y_i^o + \sum_{j,i} d_j c_{ij} x_{ij}\big)$.

For each object o, we build the tree T_o as follows. We contract the sets F'_j for $j \in D_o$ into supernodes and build a minimum spanning tree (MST) T'_o connecting these to r_o, and then connect the caches storing object o to T'_o. To bound the cost of T'_o, notice that $2z^o$ yields a fractional Steiner tree on the supernodes and r_o, since for any set S containing a supernode F'_j and not containing r_o, we have $\sum_{e\in\delta(S)} z^o_e \geq \sum_{i\in F'_j} x_{ij} \geq \frac{1}{2}$. Thus, we get $c(T'_o) \leq 4\sum_e z^o_e$ since it is well known the cost of the MST is at most twice the cost of a fractional solution for the Steiner tree LP. Observe that an edge e of T'_o joining F'_j and F'_k has $c_e \geq 4\max(\bar{C}_j, \bar{C}_k)$. Let i be a facility on which object o is stored. Notice there is a *unique* client $j \in D_o$ such that $i \in F'_j$. To connect i to T'_o, we add the edge (i,j), and add edges joining j to every cache in F'_j that has an edge incident to it in T'_o. We do this for every cache on which o is stored. Let δ_j denote the degree of the supernode F'_j in the tree T'_o. The cost of adding these extra edges is at most $\sum_{j\in D_o}(1+\delta_j)2\bar{C}_j \leq 2\sum_{j\in D_o}\delta_j \cdot 2\bar{C}_j \leq 2c(T'_o)$. Thus, $c(T_o) \leq 3c(T'_o) \leq 12\sum_e z^o_e$, and the total cost incurred is at most $14\sum_{i,o} f^o_i y^o_i + 14\sum_{j,i} d_j c_{ij} x_{ij} + 12\sum_o M_o \sum_e z^o_e$, yielding a 14-approximation algorithm.

Theorem 4.1 *There is a 14-approximation algorithm for the connected data placement problem.*

5 Extensions

The k-median variant. We can easily adapt our techniques to handle an extension of the data placement problem where additionally, for every object o, there is a bound of k_o on the number of caches that can store object o. This adds the constraints $\sum_i y^o_i \leq k_o \ \ \forall o$ to (P). We need to modify the min-cost flow network construction slightly in Steps II and IV of Section 3. In Step II, we remove the edges (b_j, t). Instead for every object o, we add a node p_o with demand $|D_o| - k_o$ and edges (b_j, p_o) for $j \in D_o$ of capacity $\frac{1}{2}$ and cost $3\gamma_j$. We also add an edge (p_o, t) with capacity k_o and cost 0. The effect of these changes is to limit the total flow on edges (a_j, r_i) and (b_j, r_i), where $j \in D_o$, to at most k_o so that at most k_o caches store object o (half-integrally). The half-integral solution $(\hat{x}, \hat{y})$ is obtained as before with p_o playing the role of t now. It is easy to see that $(\hat{x}, \hat{y})$ is feasible and Lemma 3.2 still holds. Similarly, in Step IV, we remove the edges (w^o_i, t). For every o, we add a node p_o with demand $|\{i : B^o_i \neq \phi\}| - (k_o - |D'_o|)$, add edges (w^o_i, p_o) with capacity 1 and cost 0, and add edge (p_o, t) with capacity $k_o - |D'_o|$ and cost 0. This limits the total flow on edges (v_j, r_i), where $j \in D'_o$ and (w^o_i, r_i) to at most k_o. The integer solution $(\tilde{x}, \tilde{y})$ is obtained as before and Lemmas 3.3 and 3.4 still hold. So we get the following theorem.

Theorem 5.1 *There is a 10-approximation algorithm for the data placement problem with a priori bounds on the number of caches that may store an object.*

Non-uniform object-lengths. We can obtain a bicriteria approximation algorithm for the setting where each object o has a non-uniform length l_o and the total length of the objects stored in any cache must not exceed its capacity. Constraint (1) of (P) now reads $\sum_o l_o y^i_o \leq u_i$. As mentioned in the Introduction, no approximation ratio is achievable in polynomial time in this case, unless *P =NP* (see Theorem 6.2). We show the following.

Theorem 5.2 *For the data placement problem with arbitrary object-lengths, one can compute in polynomial time a placement of cost at most* $10 \cdot OPT$ *where the cache capacities are violated by an additive amount of at most* $\max_o l_o$.

Proof : We only need to modify Steps II and IV above. Instead of formulating a min-cost flow problem to take care of cache-capacities, we will now construct an instance of the *generalized assignment problem* [36] (GAP). In Step II, each node a_j, b_j of the min-cost flow network represents a job, and each node r_i and the sink t represents a machine. Each machine r_i has processing-time capacity $2u_i$, and the sink t has 0 capacity. An edge (a_j, r_i) or (b_j, r_i) denotes that job a_j or b_j has processing time $l_{o(j)}$ on machine r_i. Its assignment cost for machine r_i is $f^{o(j)}_i + d'_j l_{o(j)} c_{ij}$, which is simply a modification of the cost of the corresponding edge in the flow network that takes into account the length $l_{o(j)}$. Job b_j also has processing time 0 and assignment cost $3d'_j l_{o(j)}\gamma_j$ on machine t, corresponding to the edge (b_j, t). All other processing times (corresponding to non-edges) are infinity. It is not hard to see that $(2x, 2y)$ induces a feasible solution to this GAP-instance of cost at most $6 \cdot OPT'$. Hence, by [36], there exists an integer solution $(2\hat{x}, 2\hat{y})$ of no greater cost where $\sum_o l_o \cdot 2\hat{y}_i^o \leq 2u_i + \max_o l_o$ for every $i \in \mathcal{F}$. Thus, $(\hat{x}, \hat{y})$ yields a half-integral solution of cost at most $3 \cdot OPT'$ where the cache capacities are violated by at most $\frac{1}{2}\max_o l_o$.

Similarly, in Step IV, we have a job for each node v_j and each node w_i^o, and a machine for each node r_i and the sink t. Each machine r_i has capacity $u_i + \frac{1}{2}\max_o l_o$ and machine t has 0 capacity. As before, an edge (v_j, r_i) or (w_i^o, r_i) represents that the corresponding job has processing time $l_{o(j)}$ or l_o respectively on machine r_i. The assignment cost is the cost of the corresponding edge in the flow network modified (as above) to incorporate object-lengths by multiplying the terms not involving the storage-cost by the object-length ($l_{o(j)}$ in case of job v_j, and l_o in case of job w_i^o). For example, corresponding to the edge $(v_j, r_{i'})$, where $i' = i_2(j)$, we set the assignment cost of job v_j on a machine $r_{i'}$ to be $f_{i'}^{o(j)} + l_{o(j)}\big(d'_j c_{i'j} + \sum_{k\in A_j} d'_k c_{i'k} + \sum_{k \in B_j} d'_k(c_{i'k} - c_{i_2(k)k})\big)$ (note that $o(k) = o(j)$ for all $k \in A_j \cup B_j$). Edge (w_i^o, t) denotes that job w_i^o has 0 processing time and 0 assignment cost on machine t. All job-machine processing times corresponding to non-edges are infinity. As in Lemma 3.3, $(\hat{x}, \hat{y})$ induces a half-integral feasible solution of cost at most twice the cost of $(\hat{x}, \hat{y})$. Using the algorithm in [36] directly, one can obtain an integer solution of no greater cost where the load of every machine r_i is at most $u_i + \frac{3}{2}\max_o l_o$. A more careful analysis that exploits the half-integrality of the solution shows that the violation in the capacity of r_i is in fact at most $\frac{1}{2}\max_o l_o$, and the load of r_i is at most $u_i + \max_o l_o$. As in Lemma 3.4 and Theorem 3.5, this yields an integer solution of cost at most $10 \cdot OPT$. ■

We observe that for the connected versions of the above extensions, one obtains the same guarantees as for the connected data placement problem. We simply use the algorithms described above (with the modification to Step I specified in Section 4) to handle the data-placement part of the problem; then we apply the rounding method of Section 4 to build the Steiner trees. The analysis from Section 4 still applies, since it is still true that for any cache i on which an object o is stored, there is a unique client $j \in D_o$ such that $i \in F'_j$.

Theorem 5.3 *There is a 14-approximation algorithm for the connected version of the following data placement problems:*

(i) the placement problem with a priori bounds on the number of caches that may store a data object;

(ii) the placement problem with arbitrary object lengths; here we obtain a bicriteria guarantee where the cache capacities may be violated by an additive amount of at most $\max_o l_o$.

6 Hardness results

In this section, we establish two hardness results. It is clear that the data placement problem with storage costs is *APX*-hard, since it is a generalization of metric UFL. We show that the data placement problem is *APX*-hard even when there are no storage costs. Our second result is that for the data placement problem with arbitrary object lengths, it is *NP*-complete to even decide if there exists a feasible solution; hence, one cannot achieve any approximation ratio in polynomial time unless *P* =*NP*.

Theorem 6.1 *The data placement problem is* APX-*hard even when there are no storage costs.*

Proof : We give a reduction from metric UFL. In the unit-demand version of metric UFL (which is still *APX*-hard), we are given a set of n facilities F with facility-opening costs $\{f_i\}_{i\in\mathcal{F}}$, a client-set D and distances/assignment-costs $\{C_{ij}\}$ that form a metric. The goal is to open a subset of the facilities and assign each client to an open facility, so as to minimize the sum of the facility-opening and client-assignment costs.

Given such a UFL instance, we construct the following instance of the data placement problem. We let $\mathcal{F} = F \cup \{\Gamma\}$ be the set of caches, and $\mathcal{D} = D \cup F'$ be the set of clients, where F' is a copy of F, i.e., for every $i \in F$, we create a unique client $\sigma(i) \in F'$. There are $|F| + 1$ data objects $o_0, o_1, \ldots, o_n$. Each client $j \in D$ has unit demand for object o_0. Each client $\sigma(i) \in F'$ has demand f_i/M for object o_i, where M is some large number such that $M \gg \max_{i,j} C_{ij}$. Each cache $i \in F$ has unit capacity, and cache Γ has capacity $n + 1 = |F| + 1$. We define the distances c_{ij} for $i \in \mathcal{F}$ and $j \in \mathcal{D}$; all other distances are equal to the shortest-path distances in the bipartite graph $\left(\mathcal{F} \cup \mathcal{D}, \big\{(i,j) : i \in \mathcal{F},\ j \in D \cup \{\sigma(i)\}\big\}\right)$ with these c_{ij}s as the edge-weights. For every $i \in F$, we set $c_{ij} = C_{ij}$ if $j \in D \subseteq \mathcal{D}'$, and 0 if $j = \sigma(i)$; for $i = \Gamma$ and every $j \in \mathcal{D}$, we set $c_{ij} = M$. It is easy to see that the c_{ij}'s form a metric.

We show that this is an approximation-preserving reduction by arguing that any UFL-solution translates to a data-placement solution of no greater cost and vice-versa. Consider a UFL-solution that opens the facilities in $S \subseteq F$ (and assigns each client to the nearest facility in S). We map this to the data-placement solution, where each cache in S stores object o_0, each cache $i \in F \setminus S$ stores object o_i, and cache Γ stores the objects o_i for $i \in S$. Clearly, the total access cost incurred for object o_0 is equal to the client-assignment cost of the UFL-solution, the total access cost incurred for the objects o_i, where $i \in S$ is $\sum_i f_i$, and the access cost for all other objects is 0. So the cost of this data-placement solution is exactly the cost of the UFL-solution.

Conversely, suppose we have a data-placement solution. We may assume that object o_0 is stored in some cache in F, otherwise we can improve the solution-cost by storing o_0 in some cache $i \in F$ (and moving the object stored in i to Γ if necessary). Let $S \subseteq F$ be the set of caches that store o_0, We open the facilities corresponding to S (and assign each client to the nearest facility in S). Since $M \gg \max_{i,j} C_{ij}$, the client-assignment cost in the UFL solution is at most the total access cost for o_0. For each cache $i \in S$, the access cost for object o_i is at least $f_i/M \cdot M = f_i$ (since the distance from i to any other cache is at least M), so the facility-opening cost of the UFL solution is at most the access cost for the objects o_i where $i \in S$. Thus, the cost of the UFL-solution is at most that of the data-placement solution. ■

Theorem 6.2 *It is* NP*-complete to decide if there exists a feasible solution to an instance of the data placement problem with arbitrary object lengths. Consequently, there is no polynomial time approximation algorithm for this problem unless* P =NP.

Proof : Membership in *NP* is immediate. The *NP*-hardness proof follows from an easy reduction from the PARTITION problem. Let $a_1, \ldots, a_m$ be an instance of the PARTITION problem with $A = \sum_i a_i/2$. In the data-placement instance, we have two caches with capacity A, m objects with lengths $a_1, \ldots, a_m$, and m clients, each of which has unit demand for a unique object. (The distances and the locations of the clients and the caches are not important.) Clearly, any feasible solution to the data placement problem yields a solution to the PARTITION problem, and vice-versa. The *NP*-completeness result follows. ■

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed paging for general networks. *Journal of Algorithms*, 28(1):67–104, 1998.

[2] B. Awerbuch, Y. Bartal, and A. Fiat. Heat & Dump: competitive distributed paging. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, 1993, pages 22–31, 1993.

[3] I. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 661–670, 2001.

[4] N. Bansal, D. Coppersmith, and M. Sviridenko. Improved approximation algorithms for broadcast scheduling. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 344–353, 2006.

[5] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341-358, 1995.

[6] J. Byrka An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In *Proceedings of the 10th APPROX*, pages 29–43, 2007.

[7] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

[8] F. Chudak and D. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

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

[10] L. Dowdy and D. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys*, 14(2):287–313, 1982.

[11] L. Fleischer, M. Goemans, V. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 611–620, 2006.

[12] B. Gavish and O. Sheng. Dynamic file migration in distributed computer systems. *Communications of the ACM*, 33(2):177–189, 1990.

[13] S. Guha and K. Munagala. Improved algorithms for the data placement problem. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–107, 2002.

[14] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 389–398, 2001.

[15] A. Gupta, A. Kumar, and T. Roughgarden. Simple and better approximation algorithms for network design. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 365–372, 2003.

[16] A. Heddaya and S. Mirdad. WebWave: Globally load balanced fully distributed caching of hot published documents. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 160–168, May 1997.

[17] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual-fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

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

[19] M. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. In *Proceedings of the IEEE Workshop on Internet Applications*, pages 62–71, July 1999.

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

[21] M. Korupolu, C. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical cooperative caching. *Journal of Algorithms*, 38(1):260–302, 2001.

[22] C. Krick, H. Räcke, and M. Westermann. Approximation algorithms for data management in networks. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 237–246, 2001.

[23] A. Leff, J. Wolf, and P. Yu. Replication algorithms in a remote caching architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1185–1204, 1993.

[24] R. Levi, R. Roundy, and D. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31:267–284, 2006.

[25] R. Levi, D. Shmoys, and C. Swamy. LP-based approximation algorithms for capacitated facility location. In *Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization*, 2004.

[26] J. H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 771–782, 1992.

[27] B. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 284–293, 1997.

[28] F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Caching in networks. In *Proceedings of 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–439, 2000.

[29] A. Meyerson, K. Munagala, and S. Plotkin. Web caching using access statistics. In *Proceedings of 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 354–363, 2001.

[30] M. Pál, É. Tardos, and T. Wexler. Facility location with nonuniform hard capacities. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 329–338, 2001.

[31] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an Internet hosting service. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 101–113, May 1999.

[32] R. Ravi and F. S. Selman. Approximation algorithms for the traveling purchaser problem and its variants in network design. In *Proceedings of the 7th Annual European Symposium on Algorithms*, pages 29–40, 1999.

[33] R. Ravi and A. Sinha. Multicommodity facility location. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 335–342, 2004.

[34] D. B. Shmoys. Approximation algorithms for facility location problems. In *Proceedings of the 3rd APPROX*, pages 27–33, 2000.

[35] D. B. Shmoys, C. Swamy, and R. Levi. Facility location with service installation costs. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1081–1090, 2004.

[36] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461–474, 1993.

[37] D. B. Shmoys, É. Tardos, and K. I. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[38] C. Swamy. Algorithms for Data Placement Problems. *Unpublished manuscript*, 2004.

[39] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245–269, 2004.

[40] C. Swamy and D. B. Shmoys. Fault-tolerant facility location. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 735–736, 2003.

[41] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2):255–314, 1997.

[42] J. Zhang, B. Chen, and Y. Ye. A multi-exchange local search algorithm for the capacitated facility location problem. In *Proceedings of the 10th IPCO*, pages 219–233, 2004.