Proceedings of Symposia in Applied Mathematics

The Design and Analysis of Approximation Algorithms: Facility Location as a Case Study

David B. Shmoys

Abstract. Approximation algorithms are polynomial-time algorithms for optimization problems that are guaranteed to find near-optimal solutions. The design and analysis of approximation algorithms for *NP*-hard problems has recently been one of the most active areas in optimization. We will present several different approaches to designing approximation algorithms, by focusing on one problem domain in discrete optimization, that of facility location problems.

1. Introduction to approximation algorithms

Most combinatorial optimization problems are *NP*-hard, and hence are unlikely to have algorithms that are guaranteed to find optimal solutions within polynomial time. One of the primary approaches to cope with this intractability is to settle for less than optimality, and to focus on *approximation algorithms* instead. That is, you design a polynomial-time algorithm for your problem that need not always give the optimal solution, but always produces a solution that is nearly optimal, i.e., with objective function value within some performance guarantee. In recent years, one of the most active areas of research within optimization has been in the design and analysis of approximation algorithms.

In this lecture, we will focus on two closely related computational problems, in order to compare and contrast several different algorithmic approaches that have played leading roles in the recent burst of activity in this field. The central concept for this approach is that of a *ρ-approximation algorithm* for an optimization problem: a polynomial-time algorithm that delivers a feasible solution of objective function value within a factor of ρ of the optimal value. The study of approximation algorithms predates the theory of *NP*-completeness. Some early results, such as the proof due to Vizing that a graph always has an edge coloring with at most one more color than the maximum degree, were not even algorithmically stated, but still contain all of the ideas needed to give an interesting approximation algorithm. Graham, in studying a particular scheduling problem, more explicitly stated the goal of analyzing an efficient heuristic from the perspective of its worst-case error

1991 *Mathematics Subject Classification.* Primary 90B80, 90C27; Secondary 68W40, 68W25.
Key words and phrases. combinatorial optimization, design and analysis of algorithms.
The author was supported in part by NSF Grant CCR-9912422.

bound, or more positively stated, its performance guarantee. The seminal work of Johnson, reacting to the recent discovery of the theory of *NP*-completeness, crystallized the perspective that one approach to coping with the intractability of a problem is to design and analyze approximation algorithms. Much of the history of the work in this area over the past forty years is summarized in the textbook of Vazirani [**6**], the volume of surveys edited by Hochbaum [**3**], and the survey of Shmoys [**4**]. Furthermore, there is also recent work that has provided techniques to prove, for some problems, that no such approximation algorithm can exist. This type of result is also discussed in the references above.

Most combinatorial optimization problems can be formulated as integer programming problems; that is, the set of feasible solutions can be described by decision variables restricted to integer values, and constrained to satisfy a collection of linear inequalities and equations, and the objective function to be optimized can also be expressed as a linear function of the decision variables.

For example, consider the following problem, known as the *uncapacitated facility location problem*, which is to be the main focus of this lecture. In this problem, there is a set of locations $\mathcal{F}$ at which we may build a facility (such as a warehouse), where the cost of building at location i is $f_i > 0$, for each $i \in \mathcal{F}$. There is a set $\mathcal{D}$ of client locations (such as stores) that require to be serviced by a facility, and if a client at location j is assigned to a facility at location i, a cost of c_{ij} is incurred. The objective is to determine a set of locations at which to open facilities so as to minimize the total facility opening and client assignment costs. All of the data are assumed to be non-negative, and furthermore, we will assume that all of the locations are located in a common metric space, so that for any $i, j, k \in \mathcal{N} = F \cup D$, $c_{ik} \leq c_{ij} + c_{jk}$; finally, the costs are also symmetric: for any $i, j \in \mathcal{N}$, $c_{ij} = c_{ji}$. Note that we can think of these costs as representing the distances between locations. We have focused on this "metric" special case for a good reason; without this assumption, one can prove that no good approximation algorithms exist.

There are two types of decision variables x_{ij} and y_i, where each variable y_i, $i \in \mathcal{F}$, indicates whether or not a facility is built at location i, and each variable x_{ij} indicates whether or not the client at location j is assigned to a facility at location i, for each $i \in \mathcal{F}$, $j \in \mathcal{D}$:

$$\text{Minimize } \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{D}} c_{ij}x_{ij} \tag{1.1}$$

subject to

$$\sum_{i\in\mathcal{F}} x_{ij} = 1, \qquad \text{for each } j \in \mathcal{D}, \tag{1.2}$$

$$x_{ij} \leq y_i, \qquad \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}, \tag{1.3}$$

$$x_{ij} \geq 0, \qquad \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}, \tag{1.4}$$

$$x_{ij} \quad \text{integer}, \qquad \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}. \tag{1.5}$$

In effect, we force each variable to be either 0 or 1. We may interpret setting $y_i = 1$ to mean that the facility at location i has been opened, whereas if we set $x_{ij} = 1$, then the client at location j is assigned to a facility at location i. The constraints (1.2) ensure that each client is assigned to some location, whereas the constraints (1.3) ensure that if a client at location j is assigned to a location i, then indeed, a facility has been opened a location i. So, this is an integer programming formulation of the uncapacitated facility location problem. However, the problem is *NP*-hard, and hence no polynomial-time algorithm to solve it will exist unless *P*=*NP*, which is widely viewed as being extremely unlikely.

If we drop the restriction that the variables in (1.1)–(1.4) must take integer values, then we obtain a linear programming (LP) problem. When we drop the integrality restriction in this way, we obtain a *linear programming relaxation* for a problem. Unlike integer programming, there are efficient algorithms known to solve linear programs to optimality. Thus, we can solve the linear programming relaxation of the uncapacitated facility location problem in polynomial time. This leads to one of the most generally applicable methods for designing approximation algorithms: the approach of *LP rounding*. In this approach, we first formulate a problem as an integer program, then consider its linear programming relaxation, solve the relaxation to optimality, and then attempt to round the fractional solution obtained in this way to an integer one (using some polynomial-time procedure). If we can obtain a feasible integer solution that has objective function value within a factor of ρ of the value of the fractional solution that has been rounded, then the resulting algorithm is a ρ-approximation algorithm. The first approximation algorithm for our location problem that we will present will be an LP rounding algorithm.

Unfortunately, LP rounding is not always the method of choice. The main reason for this is that although linear programming relaxations can be solved reasonably efficiently, we are often interested in large inputs, and the LPs that must be solved are quite often huge – too huge to solve within a reasonable amount of time. We can still devise an approximation algorithm based on linear programming, however. For every linear program: minimize $c^T x$ subject to $Ax = b$, $x \geq 0$, there is a dual linear program: maximize $y^T b$ subject to $y^T A \leq c^T$. The strong theorem of linear programming duality states that, provided an optimal solution exists for the original LP, the optimal value of the primal and dual linear programs must be equal. (The reader is referred to the textbook of Chvátal [**2**] for a thorough introduction to linear programming.)

For example, the dual linear program for the linear relaxation (1.1)–(1.4) of the uncapacitated facility location problem is as follows:

$$\text{Maximize} \sum_{j \in \mathcal{D}} v_j \tag{1.6}$$

subject to

$$\sum_{j \in \mathcal{D}} w_{ij} \leq f_i, \qquad \text{for each } i \in \mathcal{F}, \tag{1.7}$$

$$v_j - w_{ij} \leq c_{ij}, \qquad \text{for each } i \in F,\ j \in \mathcal{D}, \tag{1.8}$$

$$w_{ij} \geq 0 \qquad \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}. \tag{1.9}$$

This dual can be motivated in the following way. Suppose that we wish to obtain a lower bound for our input to the uncapacitated facility location problem. If we reset all fixed costs f_i to 0, and solve this input, then clearly we get a (horrible) lower bound: each client $j \in \mathcal{D}$ gets assigned to its closest facility at a cost of $\min_{i \in \mathcal{F}} c_{ij}$. Now suppose we do something a bit less extreme. Each location $i \in \mathcal{F}$ decides on a given cost-sharing of its fixed cost f_i. Each location $j \in \mathcal{D}$ is allocated a share w_{ij} of the fixed cost; if j is assigned to an open facility at i, then it must pay an additional fee of w_{ij} (for a total of $c_{ij} + w_{ij}$), but the explicit fixed cost of i is once again reduced to 0. Of course, we insist that each $w_{ij} \geq 0$, and $\sum_{j \in \mathcal{D}} w_{ij} \leq f_i$

for each $i \in \mathcal{F}$. But this is still an easy input to solve: each $j \in \mathcal{D}$ incurs a cost $v_j = \min_{i \in \mathcal{F}}(c_{ij} + w_{ij})$, and the lower bound is $\sum_{j \in \mathcal{D}} v_j$. Of course, we want to allocate the shares so as to maximize this lower bound, and this maximization problem is precisely the LP dual.

The duality theory of linear programming leads to the so-called *primal-dual method* for designing approximation algorithms. Suppose that we design an algorithm that simultaneously constructs a feasible integer solution to our integer programming formulation, and a feasible solution to the dual of its linear programming relaxation. If the algorithm runs in polynomial time, and we prove that the objective function value of the integer solution is always within a factor of ρ of the (dual) objective function value of the feasible dual solution constructed, then we have obtained a ρ-approximation algorithm. To see this, note that the objective function value of *any* feasible dual solution is a lower bound on the optimal LP value (dual or primal, they are the same), which is in turn a lower bound on the optimal value to our integer program. Hence, if we find a solution of value at most ρ times the value of a feasible dual solution, then it is also at most ρ times the integer optimal value. We shall describe a primal-dual approximation algorithm for the uncapacitated facility location problem in Section 3.

Unfortunately, this is not typically the type of approach employed when a practitioner wishes to compute good solutions fast. A much more common approach is to base the design of an algorithm on the notion of *local search*. Given a feasible solution for a combinatorial optimization problem, it is often natural to think of ways in which one might slightly modify one's current solution to obtain a new solution. For example, in the uncapacitated facility location problem, one might consider opening one extra facility, or closing one of the facilities already open (and then readjusting the assignment of clients to facilities accordingly).

One can view the set of all feasible solutions as a large *neighborhood graph*: each node in the graph corresponds to a feasible solution, and two nodes (i.e., feasible solutions) are adjacent if one can be obtained from the other by one of the specified perturbations. (For simplicity, we shall assume that the perturbations are reversible, and hence the graph is undirected.) A *local optimum*, with respect to some neighborhood structure, is a feasible solution whose cost is no more than all of its neighbors. One naive approach to finding a good solution is to start with an arbitrary feasible solution, and repeatedly check if there is some neighboring solution with better objective function value, until one finds a local optimum. Such an algorithm is called a *local search procedure*. Remarkably, these algorithms often perform very well in practice, especially when enhanced in various ways with additional randomization within the algorithm; simulated annealing is one approach to adding randomization to such algorithms. (The reader is referred to the volume of Aarts and Lenstra [**1**] for an in-depth treatment of this subject.) In Section 4, we shall also present a simple local search algorithm that can be shown to be a 3-approximation algorithm.

The *k-median problem* is closely related to the uncapacitated facility location problem. In this problem, there is a set $\mathcal{N}$ of locations, and the aim is to select k of them (as "medians"), and assign each point in $\mathcal{N}$ to one of the selected medians so as to minimize the total assignment cost. In other words, if we start with the uncapacitated facility location problem, and consider the case in which $\mathcal{D} = \mathcal{F} = \mathcal{N}$, set each $f_i = 0$, but impose the additional constraint that $\sum_{i \in \mathcal{N}} y_i = k$, then we obtain the k-median problem. For any input to the k-median problem, we can view it as an input to the uncapacitated facility location problem, except that there are no facility costs given. Suppose we set the facility costs equal to some uniform value z; if $z = 0$, then a reasonable solution to the uncapacitated facility location problem would tend to open a facility at each location; on the other hand, if z is set *very* large, then only one facility would be opened. By adjusting the value of this *Lagrangean multiplier*, it seems natural that we can find a solution that opens k facilities, and hence can serve as a good solution for the k-median problem. In fact, this approach does work, and we will describe this in Section 5.

2. An LP rounding algorithm

We shall first describe a simple procedure to take an optimal solution (x^*, y^*) to the linear programming relaxation for the uncapacitated facility location problem and produce a feasible integer solution $(\bar{x}, \bar{y})$; furthermore, it will be straightforward to prove that the cost of the integer solution is no more than 4 times the cost of the optimal fractional solution. Let (v^*, w^*) denote an optimal solution to the dual linear program.

The algorithm works as follows. First represent the fractional solution (x^*, y^*) as a bipartite graph $G = (\mathcal{F}, \mathcal{D}, E)$ where, for each (i, j) such that $x^*_{ij} > 0$, we include the edge (i, j) in E, i.e., there are two set of nodes $\mathcal{F}$ and $\mathcal{D}$, and each edge in E has one endpoint in each set. We shall rely on an important property of optimal solutions of linear programs, known as complementary slackness: this states that whenever a variable is positive in an optimal solution, then for any optimal solution to the dual, the constraint in the dual that corresponds to this variable must be satisfied with equality. In particular, this means that for each edge $(i, j) \in E$, we have that $v^*_j - w^*_{ij} = c_{ij}$. Since $w^*_{ij} \geq 0$, it follows that $c_{ij} \leq v^*_j$; for each edge in G, we know that the associated assignment cost is not too large.

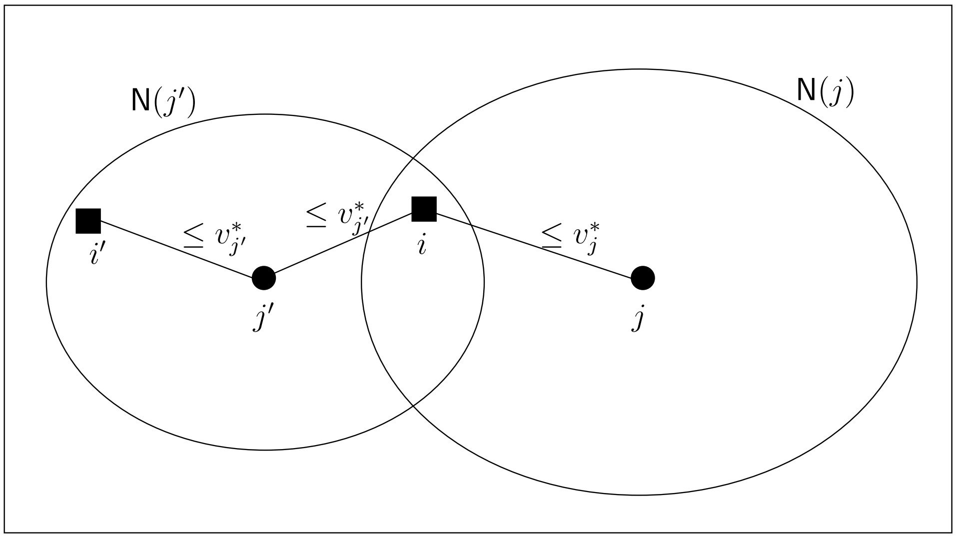

FIGURE 1. Bounding the assignment cost of j; $\mathsf{N}(j)$ and $\mathsf{N}(j')$ denote the set of neighbors of j and j', respectively, in G.

The algorithm partitions the graph G into clusters, and then, for each cluster, opens one facility. The clusters are constructed iteratively as follows. Among all clients that have not already been assigned to a cluster, let j' be the client j for which v_j^* is smallest. This cluster consists of j', all neighbors of j' in G, and all of their neighbors as well (that is, all nodes j such that there exists some i for which (i, j) and (i, j') are both in E). Within this cluster, we open the cheapest facility i'. After all of the clusters have been constructed, each client is then assigned to its nearest opened facility.

We first show that each client has a nearby facility that has been opened. Consider a client j, let i' denote the facility opened in the cluster that contains j, and let j' be the node with minimum v^*-value used in forming the cluster (see Figure 1). In other words, we know that there exists a path in G from j to i' consisting of an edge connecting i' and j' (of cost at most $v_{j'}^*$), an edge connecting j' and some node i (of cost at most $v_{j'}^*$), and an edge connecting i and j (of cost at most v_j^*). Hence, by the triangle inequality, the cost of assigning j to i' is at most $2v_{j'}^* + v_j^*$. Since j' was chosen as the remaining client with minimum v^*-value, it follows that $v_{j'}^* \le v_j^*$, and so the cost of assigning j to i' is at most $3v_j^*$. Hence, the total assignment cost of the solution found is at most $3\sum_{j\in\mathcal{D}} v_j^*$, which is 3 times the optimal objective value of the (dual) linear program.

Next we bound the total cost of the facilities that are opened by the algorithm. Consider the first cluster formed, and let j' be the node with minimum v^*-value used in forming it. We know that $\sum_{i:(i,j')\in E} x_{ij'}^* = 1$. Since the minimum of a set of values is never more than a weighted average of them, the cost of the facility selected is

$$f_{i'} \le \sum_{i:(i,j')\in E} x_{ij'}^* f_i \le \sum_{i:(i,j')\in E} y_i^* f_i,$$

where the last inequality follows from constraint (1.3). That is, for the first cluster, the cost of the facility opened in that cluster is no more than the fractional cost incurred by all of the facilities (fractionally) opened in the cluster in the LP optimum.

Observe that, throughout the execution of the algorithm, each location $j \in \mathcal{D}$ that has not yet been assigned to some cluster has the property that each of its neighbors i must also remain unassigned. Hence, for *each* cluster, the cost of its open facility is at most the cost that the fractional solution assigned to nodes in $\mathcal{F}$ within that cluster. Hence, in total,

$$\sum_{i\in\mathcal{F}} f_i \bar{y}_i \le \sum_{i\in\mathcal{F}} f_i y_i^*;$$

that is, the total facility cost of the rounded solution is at most the facility cost incurred by the optimal fractional solution (which is, of course, at most the total cost of the optimal LP solution). Thus, the total cost of the solution found by the rounding algorithm is at most 4 times the cost of the optimal solution to the LP.

There is a simple way to improve the algorithm, which relies on a technique known as *randomized rounding*. Consider a cluster formed by first selecting the client j'. In the previous algorithm, we opened the cheapest neighbor of j'. Instead, choose a neighbor i of j' with probability $x_{ij'}^*$, and open that facility. Furthermore, for each facility i in the cluster, open an additional facility at i with probability $y_i^* - x_{ij'}^*$. Finally, for each facility i that is not in any cluster, open that facility with probability y_i^*. Once again, assign each client to its nearest open facility. This is a randomized algorithm; the algorithm "tosses coins" and the output of the algorithm is a random variable. It is easy to see that the expected facility cost incurred by this solution is at most the facility cost incurred by the optimal LP solution. In this case, one can prove a tighter bound on the expected assignment cost incurred (since, for each client j, with probability at least $1 - 1/e$, one of the neighbors of j in G is an open facility). Overall, one can prove that the expected cost of the solution found by this randomized algorithm is at most $1+3/e$ times the cost of the optimal LP solution. Furthermore, one can apply a simple "greedy rule" to obtain, without randomization, a solution of cost no greater than this expected value, and hence obtain a $(1+3/e)$-approximation algorithm. (This derandomization method is generally called the *method of conditional expectations*.)

3. A primal-dual algorithm

Primal-dual algorithms have the advantage that one need not solve a linear program, while still exploiting the structure that a linear programming relaxation brings to many problems. As described above, we will construct a feasible integer solution for the facility location problem, and at the same time construct a feasible solution to the dual of the linear programming relaxation. To prove the performance guarantee of the algorithm, we will bound the cost of the (primal) solution computed by a constant factor of the objective function of the dual solution computed.

We shall describe an algorithm that works in two phases. In the first phase, we compute a feasible dual solution. This dual solution will be used to construct a graph, much as the optimal LP solution was used in the previous section. In the second phase, the graph will be decomposed into clusters (though in a manner different from the method used above), where again we open one facility in each cluster. Finally, each client j will be assigned to the nearest opened facility.

The procedure to compute a feasible dual solution is what is often called a *dual ascent method*. We start with a feasible solution, where all of the dual variables are set to 0, and then increase the value assigned to some of the variables while maintaining a feasible solution. In particular, there is a continuous notion of time that evolves throughout the execution of the algorithm. At time $t = 0$, all of the dual variables are set to 0. As time progresses, each variable v_j will continue to be equal to the time t, until some point in the algorithm, when that variable is frozen, and that variable remains fixed for the remainder of the algorithm. For each client j, the final value of its dual variable, which we shall denote $\tilde{v}_j$, denotes the time at which that variable is frozen. Initially, we shall keep all of the variables w_{ij} set to 0.

As time progresses, there will be three types of events that determine the course of the algorithm. Suppose that we have reached a time t for which, for some client, $v_j = t$ is equal to c_{ij}, for some facility location $i \in \mathcal{F}$. We can view this as a physical process, where the sphere of influence for client j has reached the facility i. Note that if we allow the algorithm to proceed unchanged, then at subsequent times, we shall set v_j to a value for which constraint (1.8) is violated, and we will no longer maintain the invariant that the current dual solution is feasible. Instead, we also increase w_{ij} at a rate of 1 unit per unit of time elapsed; that is, we maintain that v_j is equal to the current time t, and also maintain that $c_{ij} + w_{ij} = v_j$. If one views v_j as being the composite cost of serving the client at location j, once the current budget for j is sufficient to pay to connect to a facility at location i, then the excess can go to the share of the fixed cost for i that location j is willing to pay.

The second type of event occurs when, for some facility location i, the total of the shares being offered to it by the clients is sufficient to pay the fixed cost f_i. Stated more algebraically, we have increased the values w_{ij} so that the constraint (1.7) now holds with equality. Clearly, when this happens, we can no longer increase each w_{ij} for a client j for which $v_j \geq c_{ij}$. For each such client j, we freeze all dual variables v_j and $w_{i'j}$ (for all i' including i) associated with the client j (at their current values). We shall say that facility i is now *paid for*, and we let t_i denote the time at which facility i is paid for.

In the final type of event, the dual variable v_j for a client j increases to be equal to c_{ij}, where the facility i is already paid for (by the shares of *other* clients). Unlike in the first type of event, we cannot increase w_{ij} anymore, and hence we freeze all dual variables v_j and $w_{i'j}$ at their current values. Although we have described this procedure in terms of a continuous notion of time, it is straightforward to view it in terms of the events that occur, by calculating the next (discrete) event (and advancing the time to that moment).

We continue this dual ascent process until all dual variables v_j have been frozen. Let $(\tilde{v}, \tilde{w})$ denote the final solution computed by this procedure. Construct the graph $G = (\mathcal{F}, \mathcal{D}, E)$ where, for each (i,j) such that $\tilde{w}_{ij} > 0$, we include the edge (i,j) in E. Once again, we have a similar property as in the LP algorithm: for each edge $(i,j) \in E$, we have that $c_{ij} + \tilde{w}_{ij} = \tilde{v}_j$. Since $\tilde{w}_{ij} > 0$, it follows that $c_{ij} < \tilde{v}_j$; for each edge in G, we know that the associated assignment cost is not too large.

Once again, the algorithm partitions the graph G into clusters, and then, for each cluster, opens one facility. The clusters are constructed iteratively as follows. Among all facilities that are paid for and that have not yet been assigned to a cluster, let i' be the facility that is paid for earliest; that is, t_i is smallest. The new cluster consists of i', all neighbors of i' in G, and all of their neighbors as well (that is, all facility locations i such that there exists some j for which (i,j) and (i',j) are both in E (or in other words, are such that both $\tilde{w}_{ij} > 0$ and $\tilde{w}_{i'j} > 0$). We open the facility i'. This continues until all paid-for facilities have been assigned to a cluster. After all of the clusters have been constructed, each client is then assigned to its nearest opened facility.

We now analyze this algorithm. We will prove that the total cost of the solution found is at most $3\sum_{j\in\mathcal{D}} \tilde{v}_j$, and hence is within a factor of three of optimal. In fact, we will prove a stronger property: we will show that not only is the total cost at most this much, if we consider 3 times the facility cost incurred plus the total assignment cost, then even this quantity is no more than $3\sum_{j\in\mathcal{D}} \tilde{v}_j$. (We will explain in Section 5 why this is useful.)

Consider any client j that is adjacent in G to a facility i that has actually been opened. Note that our algorithm has ensured that, for any client j, there is at most one such facility. The assignment cost for this client is therefore at most c_{ij}. But by dual feasibility, we see that even $3(c_{ij} + \tilde{w}_{ij}) = 3\tilde{v}_j$. We shall charge each such client only $c_{ij} + 3\tilde{w}_{ij}$, which is at most $3\tilde{v}_j$. This charge pays for its assignment cost, and 3 times its share of the fixed cost f_i.

On the other hand, consider the open facility i; each of its neighbors j' in G will pay for three times $\tilde{w}_{ij'}$, its share of the fixed cost f_i. But these neighbors are the only clients that contribute *any* positive share to the fixed cost of location i in the dual solution $(\tilde{v}, \tilde{w})$, and since i was opened, the facility at location i must have been paid for. Hence, the total charge incurred by its neighbors must pay for all of their assignment costs, as well as $3f_i$.

Consider any client that is not adjacent to an open facility. For each such client j, we will argue that its assignment cost is most $3\tilde{v}_j$, and this is sufficient to complete the proof of the performance guarantee. Since the dual variable v_j was frozen at some point by the algorithm, there must be some paid-for facility i for which $\tilde{v}_j \geq c_{ij}$. If i has been opened, then the assignment cost incurred for j is clearly at most $c_{ij} \leq \tilde{v}_j \leq 3\tilde{v}_j$.

Suppose that i has not been opened. Then it must belong to the cluster of some other open facility i'; furthermore, there is a client j' that is contributing a positive share to the facility costs of both i and i'. Consider the moments in time that facilities i and i' have been paid for, denoted t_i and $t_{i'}$, respectively. Since facility i is in the cluster started by facility i', we have that $t_{i'} \leq t_i$. Furthermore, since client j' contributes a positive share to the fixed cost of both i and i', we have that $c_{ij'} < t_i$ and $c_{i'j'} < t_{i'}$. Furthermore, facility i must be paid for no later than the time that client j is frozen; hence, $t_i \leq \tilde{v}_j$. Hence, the distance from client j to the open facility i' is at most

$$c_{ij} + c_{ij'} + c_{i'j'} < \tilde{v}_j + t_i + t_{i'} \leq \tilde{v}_j + 2t_i \leq 3\tilde{v}_j.$$

Hence, we have shown that this primal-dual algorithm is a 3-approximation algorithm.

4. A local search algorithm

Local search is one of the most effective techniques for designing heuristic procedures to compute near-optimal solutions. Unfortunately, in most cases we cannot prove a strong guarantee on the quality of the solution found. LP rounding algorithms and primal-dual algorithms compute both a feasible integer solution and a lower bound; hence, even without an *a priori* guarantee, these approaches generate, on an input by input basis, a means to provide an *a postiori* bound on the performance of the algorithm. Unfortunately, no comparable information is generated by a local search procedure. Thus, it is even more important to be able to prove a performance guarantee for a local search algorithm that holds for all inputs.

In describing a feasible solution to the uncapacitated facility location problem, it is natural to focus on the set of facilities that have been opened, since we can assume that each client is assigned to its nearest open facility. Thus, our aim is to find a set S^* for which the total cost is minimum. However, in describing the local search procedure, we will maintain both a set of open facilities, and a particular assignment of clients to open facilities.

The algorithm starts by choosing one facility location $i_0 \in \mathcal{F}$ arbitrarily, and then setting $S = \{i_0\}$ (and each client is assigned to that facility). In each iteration, one can change S in one of two possible ways: (1) an *add move*, in which some facility location $i \in \mathcal{F}$ is added to S; and (2) a *swap move*, in which we first pick one facility location i^* to be added to S, and then for each $i \in S$, we see whether there is an improving net effect if we delete i and reassign to i^* *all* of the clients that i currently serves; if so, we delete i and make this reassignment. It is easy to check whether there is a move of either type that yields an improved solution; if so, the algorithm modifies S by (an arbitrarily chosen) one of these allowed moves.

We shall call S a *local optimum* if there is no move of either type that yields an improved solution.

Let S^* denote an optimal solution, and let F^* and C^* denote the total facility and assignment costs associated with it. Consider an arbitrary feasible solution S, and let F and C denote its associated facility and assignment costs. We will prove that the following two claims hold:

1. if no add move from S decreases the cost, then $C \leq C^* + F^*$.
2. if no swap move from S decreases the cost, then $F \leq F^* + C^* + C$.

We first argue that if we establish both of these claims, then this will imply a performance guarantee on the objective function value of an arbitrary local optimum S. There is no improving add move from S, and so, $C \leq C^* + F^*$. Similarly, there is no improving swap move from S, and hence $F \leq F^* + C^* + C$. If we add twice the former inequality to the latter one, we see that $2C + F \leq 3F^* + 3C^* + C$, and hence $C + F \leq 3F^* + 3C^*$. But this means that the cost of the local optimum S is at most three times the optimal cost.

So now we must prove that the two claims hold. Before proceeding with their proofs, we first introduce a bit more notation. For the optimal solution S^*, let $\iota^*(j)$ denote the facility $i \in S^*$ to which j is assigned, for each $j \in \mathcal{D}$. Furthermore, for each $i \in S^*$, let D_i^* denote the set of locations j for which $\iota^*(j) = i$. Similarly, for the current solution S, let $\iota(j)$ denote the facility $i \in S$ to which j is currently assigned, for each $j \in \mathcal{D}$, and let D_i denote the set of locations $j \in \mathcal{D}$ currently assigned to i (i.e., $\iota(j) = i$).

The proof of the first claim, about the add move, is relatively straightforward. Suppose that no add move exists that improves the cost of the current solution S. For each $i \in S^*$, consider the net change in cost of adding i to the current solution. Ordinarily, one would assume that not only is i added to the solution, but each client j that is closer to i than to its current facility is reassigned to i. We will consider instead an inferior move, in which i is added to the solution, and each location $j \in D_i^*$ is assigned to i, whether or not this is an improvement (and these are the only changes in the assignment). Of course, none of these moves will improve the overall cost of the solution either. That is, we have that

$$0 \leq f_i + \sum_{j \in D_i^*} c_{ij} - c_{\iota(j)j}. \tag{4.1}$$

By adding (4.1) for each $i \in S^*$, we get that

$$\begin{aligned} 0 &\leq \sum_{i \in S^*} f_i + \sum_{j \in D_i^*} (c_{ij} - c_{\iota(j)j}) \\ &= \sum_{i \in S^*} f_i + \sum_{i \in S^*} \sum_{j \in D_i^*} c_{ij} - \sum_{i \in S^*} \sum_{j \in D_i^*} c_{\iota(j)j} \\ &= F^* + C^* - C, \end{aligned}$$

where the last equality follows from the fact that D_i^*, $i \in S^*$, constitutes a partition of $\mathcal{D}$ (based on the facility to which the clients are assigned in the optimal solution). Of course, this implies that $C \leq F^* + C^*$, which is what we wished to prove.

The proof of the second claim is somewhat more complicated, but has the same flavor; rather than analyzing the original swap move, one analyzes an inferior (and unimplementable) modified swap move. The modified swap move can be described as follows. For each $i \in S$, consider the set $\hat{S} = \cup_{j \in D_i} \iota^*(j)$, and find the facility

$i^* \in \hat{S}$ such that c_{ii^*} is smallest. In this case, we say that i *maps to* i^*. For each $i^* \in S^*$, let S_{i^*} denote the set of locations $i \in S$ that map to i^*. In our *modified swap move*, we add i^*, delete S_{i^*}, and assign each client $j \in \cup_{i \in S_{i^*}} D_i$ to i^*.

For the modified swap move, we are closing the facilities at S_{i^*} and instead opening a facility at i^*. Therefore, we need to reassign all clients that were currently assigned to a facility $i \in S_{i^*}$. The natural thing is to reassign them to the newly opened facility at i^*, and we want to upper bound the increase in their assignment cost by this change. We shall argue that, for each $j \in D_i$,

$$c_{i^*j} \leq 2c_{ij} + c_{\iota^*(j)j}; \tag{4.2}$$

in other words, the reassignment changes the assignment cost of j from c_{ij} to at most $c_{ij} + (c_{ij} + c_{\iota^*(j)j})$, or equivalently, the increase is at most $c_{\iota(j)j} + c_{\iota^*(j)j}$ (since $\iota(j) = i$). The cost from i^* to j can be upper bounded by the sum of the cost from i^* to i, and the cost from i to j; that is,

$$\begin{aligned} c_{i^*j} &\leq c_{i^*i} + c_{ij} \\ &\leq c_{\iota^*(j)i} + c_{ij} \\ &\leq c_{\iota^*(j)j} + c_{ij} + c_{ij}, \end{aligned}$$

where first and last inequalities follow from the triangle inequality, and the middle one follows from our choice of i^*.

This modified swap move has the limitation that, having fixed i^* as the new facility, and the one to which all reassignments will be made, we simply delete all current facilities i that map to i^* rather than focusing on the set of current facilities for which the savings in deleting i outweighs the additional assignment cost. Consequently, if there is no improving swap move, there is also no improving modified swap move.

Consider the change in cost associated with a modified swap move associated with $i^* \in S^*$. This change is

$$f_{i^*} - \sum_{i \in S_{i^*}} f_i + \sum_{i \in S_{i^*}} \sum_{j \in D_i} (c_{\iota(j)j} + c_{\iota^*(j)j}) \geq 0,$$

where the inequality follows from that fact that the move is not an improving one. By summing this inequality over all choices $i^* \in S^*$, we get that

$$\begin{aligned} 0 &\leq \sum_{i^* \in S^*} \left[f_{i^*} - \sum_{i \in S_{i^*}} f_i + \sum_{i \in S_{i^*}} \sum_{j \in D_i} (c_{\iota(j)j} + c_{\iota^*(j)j}) \right] \\ &= \sum_{i \in S^*} f_i - \sum_{i \in S} f_i + \sum_{j \in \mathcal{D}} (c_{\iota(j)j} + c_{\iota^*(j)j}) \\ &= F^* - F + C + C^*, \end{aligned}$$

which implies the second claim.

We have shown that any locally optimal solution (with respect to these two types of moves) has cost at most three times the optimum. We have not, however, proved that this local search algorithm is a 3-approximation algorithm; we would also need to prove a polynomial upper bound on the number of moves needed to reach a local optimum. We do not know if such a bound holds, and hence we settle for a weaker guarantee. Instead of focusing on whether there is a move that improves the solution at all, we can instead focus on whether there is a move that improves the solution by a $1 + \epsilon$ factor, for some constant $\epsilon > 0$. Now it is easy to show that a polynomial number of $(1+\epsilon)$ improving moves suffices to reach such a "nearly" local optimal solution (where there might be improving moves, but none that improves the cost by a $1+\epsilon$ factor). Furthermore, an analysis similar to the one above shows that this approach leads to a $(3+\epsilon)$-approximation algorithm, for any constant $\epsilon > 0$.

5. Lagrangean relaxation and the k-median problem

Lagrangean relaxation is a technique often applied in combinatorial optimization, where the constraints of a problem can be partitioned into two parts: the "easy" ones and the "hard" ones. If one ignores the hard constraints, then the optimization problem (subject to just the easy constraints) becomes (more) easily solvable. Thus, one can add a Lagrangean multiplier (or dual variable) for each hard constraint that penalizes the violation of that constraint, and this Lagrangean penalty function can then be added to the original objective function; we then optimize the combined problem to find the "appropriate" value for the Lagrangean multiplier.

Consider the natural integer programming formulation of the k-median problem:

$$\text{Minimize } \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} c_{ij}x_{ij} \tag{5.1}$$

subject to

$$\sum_{i\in\mathcal{N}} x_{ij} = 1, \qquad \text{for each } j\in\mathcal{N}, \tag{5.2}$$

$$\sum_{i\in\mathcal{N}} y_i = k, \tag{5.3}$$

$$x_{ij} \le y_i, \qquad \text{for each } i\in\mathcal{N},\ j\in\mathcal{N}, \tag{5.4}$$

$$x_{ij} \ge 0, \qquad \text{for each } i\in\mathcal{N},\ j\in\mathcal{N}. \tag{5.5}$$

If we relax the constraint (5.3), and introduce a penalty z for violating the constraint, thereby adding the term $z(\sum_{i\in\mathcal{N}} y_i - k)$, we obtain the uncapacitated facility location problem in which the fixed costs for opening the facilities are now equal to the common value z (ignoring the $-kz$ term in the objective function, which is a constant for any fixed value of z).

Given an input to the k-median problem, we can use this connection to the uncapacitated facility location problem in the following way. Suppose that we apply the primal-dual algorithm from Section 3 to our input with a small common fixed z. Then the algorithm is likely to open many (if not all) of the facilities. On the other hand, if the value z is sufficiently large, then the algorithm will open only one facility. Suppose that we have set z so that the algorithm outputs a solution with exactly k open facilities. Let $(\tilde{x},\tilde{y})$ and $(\tilde{v},\tilde{w})$ denote the primal and dual solutions constructed for this value $\tilde{z}$. If we let z denote the variable dual to the additional constraint (5.3), then the dual of the linear programming relaxation given above is:

$$\text{Maximize } \sum_{j\in\mathcal{N}} v_j \; - \; kz$$

subject to

$$\begin{array}{rcll} \sum_{j\in\mathcal{N}} w_{ij} & \leq & z, & \text{for each } i \in \mathcal{N}, \\ v_j - w_{ij} & \leq & c_{ij}, & \text{for each } i \in F,\ j \in \mathcal{N}, \\ w_{ij} & \geq & 0 & \text{for each } i \in \mathcal{N},\ j \in \mathcal{N}. \end{array}$$

Since $(\tilde{x}, \tilde{y})$ opens exactly k facilities, this is a feasible integer solution for the k-median LP formulation. Furthermore, since we have used the variable z to reflect the common fixed cost of opening a facility, $(\tilde{v}, \tilde{w}, \tilde{z})$ is a feasible solution for the dual LP. The performance guarantee that we proved for the primal-dual algorithm for the uncapacitated facility location problem implies that

$$3\sum_{i\in\mathcal{N}} \tilde{z}\tilde{y}_i + \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} c_{ij}\tilde{x}_{ij} \leq 3\sum_{j\in\mathcal{N}} \tilde{v}_j,$$

or equivalently, that

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} c_{ij}\tilde{x}_{ij} \leq 3\sum_{j\in\mathcal{N}} \tilde{v}_j - 3\tilde{z}(\sum_{i\in\mathcal{N}} \tilde{y}_i) = 3(\sum_{j\in\mathcal{N}} \tilde{v}_j - kz).$$

This is, we have found a feasible integer solution and a feasible dual solution for the k-median problem that are within a factor of three of each other.

By focusing on one problem domain, and exploring a few algorithmic techniques in depth, we have attempted to give the reader a sense of the mathematics involved in this area of algorithm design for discrete optimization problems. We have focused on results that were easy to present, rather than the state of the art for each of the approaches. Many more results are summarized in [**5**], but even that recent survey misses several interesting results from the past couple of years.

References

[1] E. L. Aarts and J. K. Lenstra, eds. *Local search in combinatorial optimization*, Princeton University Press, Princeton, 2003.

[2] V. Chvátal, *Linear programming*, W.H. Freeman, NY, 1983.

[3] D. S. Hochbaum, ed., *Approximation algorithms for NP-hard problems*, PWS Publishing Company, Boston, MA, 1997.

[4] D. B. Shmoys, *Computing near-optimal solutions to combinatorial optimization problems*, in: W. Cook, L. Lovász, and P. Seymour, eds., *Combinatorial optimization*, AMS, Providence, RI, 1994, pages 355–397.

[5] D. B. Shmoys, *Approximation algorithms for facility location problems*, in: K. Jansen, S. Khuller, eds., *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Lecture Notes in Computer Science 1913*, Springer, Berlin, 2000, pages 27–33.

[6] V. V. Vazirani, *Approximation algorithms*, Springer Verlag, Berlin, 2001.

School of Operations Research & Industrial Engineering and Department of Computer Science, Cornell University, Ithaca, New York 14853

Current address: School of Operations Research & Industrial Engineering and Department of Computer Science, Cornell University, Ithaca, New York 14853

E-mail address: `shmoys@cs.cornell.edu`