5-3-2017 12:00 AM

Solving Capacitated Data Storage Placement Problems in Sensor Networks

Zhenfei Wu
The University of Western Ontario

Supervisor
Roberto Solis-oba
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

Abstract

Data storage is an important issue in sensor networks as the large amount of data collected by the sensors in such networks needs to be archived for future processing. In this thesis we consider sensor networks in which the information produced by the sensors needs to be collected by storage nodes where the information is compressed and then sent to a central storage node called the sink. We study the problem of selecting k sensors to be used as storage nodes so as to minimize the total cost of sending information from the sensors to the storage nodes and from the storage nodes to the sink. We formulate this problem as a version of the capacitated *k-median* problem and design an approximation algorithm for it with approximation ratio $(16 + 23\beta + \frac{15}{2}\beta^2)$ where β is a parameter that measures the relative cost of sending compressed information from the storage nodes to the sink compared to the cost of sending raw data from the sensors to the storage nodes. We assume that each storage node has limited capacity so it can collect information from only a restricted number of sensors. Our algorithm is based on an algorithm by Guha for the capacitated *k-median* problem. We also study the version of the problem where a storage node has unlimited capacity, so it can collect information from any number of sensors. We show that a local search algorithm by Arya et al. can be used for this problem and it produces a solution of cost at most 5 times the cost of an optimum solution.

Keywords: data storage, capacited k-median problem, approximation algorithm

Contents

Chapter 1

Introduction

Nowadays, wireless sensor networks are widely used in industrial applications. A wireless sensor network uses autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. Sensor networks have military, environmental, health, domestic and commercial applications [3].

A sensor network is composed of nodes. There may be hundreds or even thousands of nodes each connected to one or several sensors that collect data and pass it through the network to a central storage facility. In a military field, sensor networks can be deployed in a target area to gather battle damage assessment data before or after attacks. As operations evolve and new operational plans are prepared, new sensor networks can be deployed for battlefield surveillance.

There are some environmental applications of sensor networks including tracking the movements of birds, small mammals, and insects; monitoring environmental conditions that affect crops and livestock; deploying macroinstruments for large-scale Earth monitoring; detecting chemical/biological agents; performing precision agriculture; environmental monitoring in marine, soil, and atmospheric contexts; detecting forest fires; meteorological or geophysical research; flood detection; bio-complexity mapping of the environment and pollution studies [2, 6, 7, 9, 10, 17, 26, 27, 29, 35, 51, 53]. Since sensor nodes may be strategically deployed in a forest, sensor nodes can detect fires and rely this information to the end users before a fire is spread uncontrollably [11].

Health applications for sensor networks include integrated patient monitoring, diagnostics, drug administration in hospitals; telemonitoring of human physiological data, and tracking and monitoring of doctors and patients inside a hospital [9,35,44,47,53]. For example, each patient could have a small and light-weight sensor attached to them. Each sensor has a specific task. Some may detect the heart rate while other might detect the blood pressure. This technology can also be used in our homes, as smart sensor nodes and actuators can be embedded in appliances such as vacuum cleaners, micro-wave ovens, refrigerators, and VCRs [45].

The air conditioning and heating of most buildings are centrally controlled. The temperature inside a room can vary by a few degrees; because of the distributed central system, one side might be warmer than the other. Once we installed a distributed wireless sensor network for the system, the air flow and temperature can be controlled in every room. It is estimated that such distributed technology could reduce energy consumption by two quadrillion British Thermal Units (BTUs) in the US, which amounts to savings of $55 billion per year and reducing 35

million metric tons of carbon emissions [47].

In this thesis we consider data storage systems that store in a single base station (that we call the sink) data collected from sensors. These systems might not be very efficient because some sensors might be far away from the base station and so it is expensive to send their data to the sink. An example of this kind of system is shown in Figure 1.1.

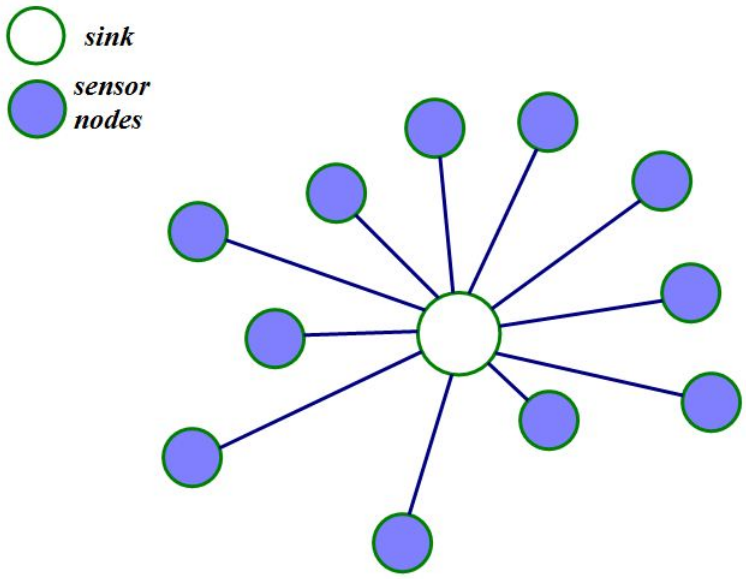

Figure 1.1: Example of a sensor network without storage nodes.

A more efficient system is one in which some sensors are designated as *storage nodes* which can collect data received from adjacent sensors. A storage node is an enhanced version of a sensor node. It can also generate data and has much larger storage space and a bigger power source than the sensor nodes. Each sensor sends the information that it collects to a storage node and storage nodes send a summary of the information collected from the sensors to the sink node. This way the total amount of information sent to the sink is much smaller than the total amount of information that the storage nodes receive.

Storage nodes have a higher cost than sensor nodes as they need to provide storage for the data that they collect. We assume then that there is a limited number k of storage nodes and that storage nodes have limited capacity, so that at most M sensors can send data to the same storage node, for some value $M > 0$. The problem that we consider in this thesis is given a wireless sensor network, select a set of at most k nodes to be designated as storage nodes that minimizes the cost of collecting the data from the sensor nodes and sending it to the sink. We call this problem the capacitated data storage placement problem (CDSP). A formal definition of the problem follows. Let N denote the number of sensor nodes including the sink. The sink is denoted as node 0. For each sensor $i \in N$ we define a variable

$$y_i = \begin{cases} 1 & \text{if } i \text{ is a storage node} \\ 0 & \text{otherwise} \end{cases}$$

For each pair i, j of sensors (i could be equal to j) we define a variable:

$$x_{ij} = \begin{cases} 1 & \text{if } y_i = 1 \text{ and sensor node } j \text{ sends its data to storage node } i \\ 0 & \text{otherwise} \end{cases}$$

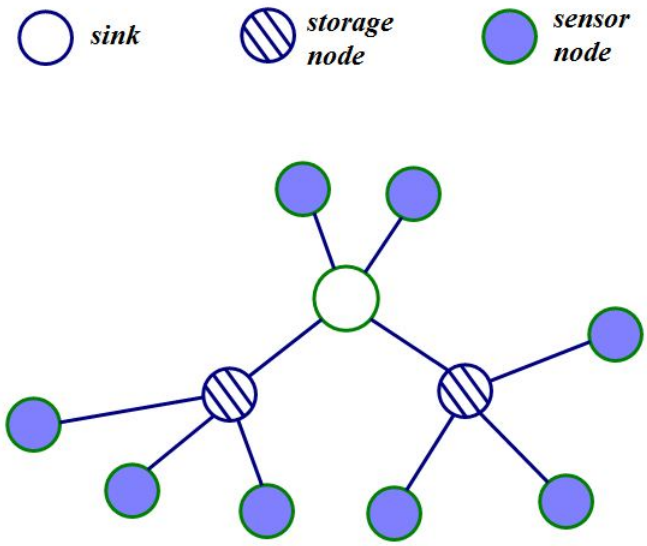

Figure 1.2: Example of a sensor network with storage nodes.

We use c_{ij} to denote the distance between nodes i and j and let δ_i be the distance between node i and the sink. The distance between two nodes might be the Euclidean distance or any other metric. The cost or energy used by node j to send data of size d to node i is $c_{ij} * d$.

The capacitated data storage placement (CDSP) problem can be formulated as the following integer program:

IP:

$$\begin{aligned} \min & \sum_{i,j \in N} x_{ij}(\alpha_1 c_{ij} + \alpha_2 \delta_i) \\ \text{s.t.} & \sum_{i \in N} x_{ij} = 1, \forall j \in N, \\ & \sum_{i \in N} y_i \leq k, \\ & \sum_{j \in N} x_{ij} \leq M y_i, \forall i \in N \\ & x_{ij} \leq y_i, \forall i, j \in N \\ & y_0 = 1, y_i, x_{ij} \in \{0, 1\}, \quad \forall i, j \in N. \end{aligned} \tag{1.1}$$

In this integer program, α_1 is the total size of the data transferred between a sensor node and its assigned storage node and α_2 is the size of the compressed data from a single sensor node

that a storage node needs to send to the sink. For determining α_1 and α_2, we assume that each node generates r_g readings per unit of time, and the data size of each reading is s_r. Also we assume there are r_q queries per unit of time and σ is the ratio between the raw data and the compressed data that a sensor node sends to the sink. Then $\alpha_1 = r_g s_r$ and $\alpha_2 = r_q \sigma s_r$. The objective function aims to minimize the total cost of transmitting information, which is equal to the sum of the cost of sending information from sensor nodes to storage nodes and the cost of sending information from storage nodes to the sink.

The first constraint, $\sum_{i \in N} x_{ij} = 1, \forall j \in N$ guarantees that each sensor node is assigned to a storage node. If node j is assigned to storage node i, then $x_{ij} = 1$ and for all other storage nodes h, $x_{hj} = 0$. If $x_{ij} = 1$ then sensor node j sends its data to storage node i. The second constraint, $\sum_{i \in N} y_i \leq k$ ensures that at most k storage nodes are selected. Each selected storage node i has $y_i = 1$. The third constraint, $\sum_{j \in N} x_{ij} \leq M y_i$ ensures that the total number of sensor nodes assigned to a storage node is at most M. This constraint is required because each storage node has limited capacity M. We also require that $x_{ij} \leq y_i$, which means if i has been chosen as a storage node, then y_i must be 1 and x_{ij} could be 0 (sensor node j not assigned to i) or 1 (sensor node j assigned to i); however, if i is not a storage node then the value of x_{ij} cannot be 1. The last constraint, $y_0 = 1$, guarantees that the sink is a storage node.

There is a well known optimization problem that is a special case of the CDSP problem called the *k-median problem*, which can be formulated as the following integer program:

IP:

$$
\begin{aligned}
\min & \sum_{i,j \in N} x_{ij} c_{ij} \\
\text{s.t.} & \sum_{i \in N} x_{ij} = 1, \forall j \in N, \\
& \sum_{i \in N} y_i \leq k, \\
& \sum_{j \in N} x_{ij} \leq M y_i, \forall i \in N \\
& x_{ij} \leq y_i, \forall i, j \in N \\
& y_i, x_{ij} \in \{0, 1\}, \quad \forall i, j \in N.
\end{aligned} \tag{1.2}
$$

The k-median problem is given a network, select k nodes to be storage nodes that minimize the total cost of transmitting information from the other nodes to the storage nodes. In the capacitated k-median problem we restrict the capacity of the storage nodes, which means that only a limited number of sensor nodes can be assigned to a storage node.

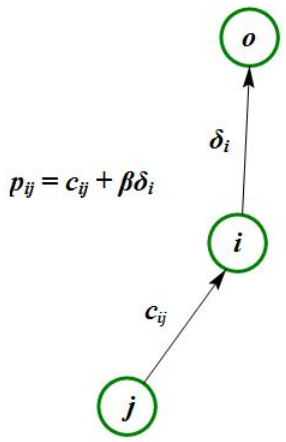

Figure 1.3: Definition of p_{ij}.

The k-median problem and the capacitated k-median problem are known to be NP-hard. A problem H is NP-hard if every decision problem in NP can be reduced in polynomial time to H [55]. If a problem is NP-hard, it is very unlikely that there exists a polynomial time algorithm for solving it as if such an algorithm existed then polynomial time algorithms would exist for all problems in NP. Hence, in this thesis we will concentrate on the design of approximation algorithms for the capacitated data storage placement problem. An approximation algorithm is an algorithms that can find approximate solutions to an optimization problem in polynomial time. Approximation algorithms are often used to deal with NP-hard problems, for which existing exact polynomial-time algorithms are too slow. The *approximation ratio* of an approximation algorithm A for a problem H is the maximum ratio $max\{\frac{SOL}{OPT}, \frac{OPT}{SOL}\}$ of the value SOL of the solution produced by the algorithm A for any instance I of problem H to the value OPT of an optimum solution for I.

For the CDSP problem, depending on the values of α_1 and α_2, the first term, $\sum_{i,j\in N} \alpha_1 c_{ij} x_{ij}$, or the second term, $\sum_{i,j\in N} \alpha_2 x_{ij} \delta_i$, in the objective function might be the dominant term. If α_1 is much larger than α_2, then the optimum solution will select a set of storage nodes that are close to the sensor nodes. Otherwise, if α_2 is much larger than α_1, then the optimum solution will select a set of storage nodes close to the sink. In general, we wish to find a set of storage nodes that are close to both the sensor nodes and the sink. For simplicity of notation we define $p_{ij} = c_{ij} + \beta\delta_i$ as the local cost, where $\beta = \frac{\alpha_2}{\alpha_1}$ and $0 < \beta < 1$, so we can simplify the objective function in the integer program (1.1) to min $\sum_{i,j\in N} x_{ij} p_{ij}$.

We consider in this thesis the case when the distance function c_{ij} satisfies the triangle inequality [37]. The triangle inequality states that for any three nodes, h, i, j, $c_{ij} + c_{jh} \geq c_{ih}$. Note that the cost function p_{ij} does not necessarily satisfy the triangle inequality even if c_{ij} does. Furthermore, p_{ij} might not be symmetric as $p_{ij} = c_{ij} + \beta\delta_i$ and $p_{ji} = c_{ji} + \beta\delta_j$.

We denote a solution for integer program (1.1) as (x, y). This solution assigns values to the variables y_j and x_{ij} and these values can be stored in a vector y and in a matrix x. For example, consider the network with five nodes A, B, C, D and E in the two dimension space in Figure 1.4. We assume that node E is the sink. A solution to the CDSP problem corresponding to this network and $k = 2$ is shown in Figure 1.4; nodes E and A are the storage nodes.

We can see from Figure 1.4 that nodes B, C and D are connected to the storage node A and A is connected to the sink E. In this solution, A and E are selected as storage nodes, so the values of y_A and y_E are 1, and the value of y_i for the other nodes is 0. Thus, the vector y is $(1, 0, 0, 0, 1)$ (entries are in alphabetical order). x_{EA}, x_{AB}, x_{AC}, x_{AD} and x_{EE} are all 1; all other variables x_{ij} have value zero. Thus, the matrix x is as follows:

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In this thesis we present an approximation algorithm for the capacitated data storage placement problem in sensor networks with approximation ratio $(16 + 23\beta + \frac{15}{2}\beta^2)$, where $\beta = \frac{\alpha_2}{\alpha_1}$. The solution produced by our algorithm violates the capacity constraints $\sum_{j \in N} x_{ij} \leq My_i$ by at most a factor of 3, i.e. for any solution (x, y) produced by our algorithm $\sum_{j \in N} x_{ij} \leq 3My_i$, or in other words the solution assumes that each storage node receives data from up to $3M$ sensor nodes.

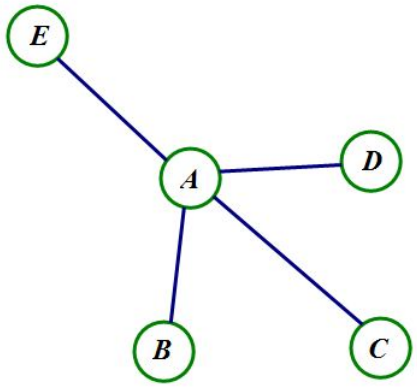

Figure 1.4: A solution for the CDSP problem. Nodes A and E are storage nodes.

Our algorithm first solves a modified integer program IP in which we allow variables y_i and x_{ij} to take fractional values. Such a modification produces a so called linear program. The reason why we do this is that linear programs can be solved in polynomial time, but solving an integer program is NP-hard. The drawback of solving a linear program is that variables y_i and x_{ij} can have fractional values. So, for example if $y_i = \frac{1}{2}$ and $y_j = \frac{1}{2}$ that means that nodes i and j are "half" storage nodes. Clearly such a solution, called a *fractional solution*, does not make sense for our problem.

To transform a fractional solution into a valid solution for our problem (this latter solution is technically called an integer solution) we need to round the fractional values of the variables to integer values. Such a rounding causes the capacity constraint to be violated. If storage nodes have hard capacities (i.e. they can not be violated), we could initially set $M' = \frac{M}{3}$ and solve the CDSP problem with capacities M'.

1.1 Related Work

The data storage placement problem on sensor networks was proposed by Sheng et al. [49]. They assumed that sensor nodes have unlimited capacity, or in other words, that a storage node can receive information from an arbitrary number of sensor nodes and presented an approximation algorithm for the problem with approximation ratio 10. In this thesis, we show that an algorithm by Arya et al. [5] can be used to solve the problem studied by Sheng et al. and the approximation ratio of the latter algorithm is only 5, greatly improving on the algorithm by Sheng et al.

The data storage placement problem is a generalization of the classical k-median problem. In the k-median problem, the goal is to select a set S of k nodes from a graph $G = (V, E)$ to be designated as centers, so that the sum of distances from the vertices in $V \setminus S$ to S is minimum [5, 13, 32, 33]. The k-median problem has been extensively studied by the research community as it has a large number of applications in many areas like networks [36], data mining [43] and the web [46].

In the metric k-median problem, the distance function on the edges of the input graph G satisfies the triangle inequality. Lin and Vitter proposed an approximation algorithm for the k-median problem with approximation ratio $1 + \varepsilon$ for any value $\varepsilon > 0$, but that selects a set of $(1 + \frac{1}{\varepsilon})(ln(n) + 1)k$ nodes as centers, where n is the number of nodes in the graph [41]. The first approximation algorithm with constant approximation ratio for the metric k-median problem was proposed by Charikar, Guha, Shmoys and Tardos and it has approximation ratio $6\frac{2}{3}$ [13]. The algorithm is based on the solution of a linear programming relaxation of the problem and a complicated rounding procedure. Jain and Vazirani [32] improved on that algorithm by proposing an elegant algorithm based on primal-dual linear programming with approximation ratio 6 and running time $O(m \log m(L + \log(n)))$ where m is the number of edges, n is the number of vertices, and $L = \log_2 d_{max}$, where d_{max} is the maximum length of the edges of the input graph. A primal dual approximation algorithm iteratively modifies two solutions: one for the linear program and one for its dual. A local search algorithm by Arya et al. [5] for the metric k-median problem is the currently best algorithm for the problem and it has approximation ratio $3 + \frac{2}{p}$ for any integer $p > 0$. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found.

A widely studied problem that is very related to the k-median problem is the facility location problem. Here we are given costs $f_i \geq 0$ for opening facilities. The problem is to select a subset of facilities to serve a group of clients such that the total facility cost plus the total service cost is minimized. The facility location problem is similar to the k-median problem except for the cost for opening the facilities. Thus, much of the research done on the k-median problem also considers the facility location problem. The facility location problem has many applications in operations research [20, 39], in network design problems such as placement of routers and caches [24, 40], agglomeration of traffic or data [4, 25], and web server replications in a content distribution network [34, 46]. In the last decade this problem has been studied extensively from the perspective of approximation algorithms [14, 18, 19, 23, 38, 50, 52].

Arya et al. [5] proposed a local search algorithm for the facility location problem and this algorithm has approximation ratio 3. Jain and Vazirani [32] used primal dual programming to design an approximation algorithm with approximation ratio 3 and running time $O(m \log m)$.

The capacitated facility location problem limits the number of clients that can connect to one facility. Jain et al. [31] gave a 3-approximation algorithm for a capacitated version of the facility location problem using primal-dual programming. Arya et al. [5] also proposed a local search algorithm for the capacitated facility location problem with approximation ratio $3.732 + \epsilon$ for any $\epsilon > 0$. When we set a capacity for each facility in the k-median problem, the problem is called the capacitated k-median problem. Charikar [15] presented an approximation algorithm for the capacitated k-median problem with approximation ratio 16; There is also a greedy algorithm by Mahdian et al [30] with approximation ratio 1.61 for the uncapacitated k-median problem.

1.2 Our Contributions

In this thesis we study a local search algorithm for the k-median problem by Arya et al. [5] and show that we can use it to approximately solve the uncapacitated data storage placement problem. We show that this algorithm has approximation ratio 5, greatly improving the algorithm by Sheng et al. which has approximation ratio 10.

We also propose an approximation algorithm for the capacitated version of the data storage placement problem in sensor networks with approximation ratio $(16 + 23\beta + \frac{15}{2}\beta^2)$ with the capacity of the facilities increased from M to $3M$. This is the first work that has considered the capacitated version of the data storage placement problem, which models real sensor networks. The thesis is organized as follows. In Chapter 2 we analyze the local search algorithm of Arya et al. [5] when used to solve the uncapacitated data storage placement problem and show that it has approximation ratio 5. In Chapter 3 we give an outline of the algorithm for the capacitated data storage placement problem and then gives the algorithm in detail. Chapter 4 analyzes the cost of the solution produced by the algorithm and its running time considering each node's demand. Chapter 5 gives our conclusions.

Chapter 2

Local Search Algorithm for the Uncapacitated Data Storage Placement Problem

2.1 The Local Search Algorithm

We first would like to mention that in the nomenclature commonly employed with the k-median and facility location problems, storage nodes are called centers and sensor nodes are called clients. In this chapter we show that the local search algorithm in [5] for the k-median problem can be used to solve the uncapacitated data storage placement problem.

The algorithm first selects as initial solution S any set of k sensors. Then the solution S is repeatedly modified by performing *swap operations*. In a swap operation, a storage node from S is swapped with a node not in S; if the new solution has a cost smaller than S, then the new solution is kept. Formally, the swap operation is defined as follows:

$swap < s, s' >:= S - s + s'$ for $s \in S$ and $s' \notin S$.

The algorithm is shown below.

Algorithm 1 Local-Search (G, k)

Input: Graph $G = (V, E)$ and value k.
Output: A set of k nodes to be used as storage nodes.
1. $S \leftarrow$ an arbitrary set of k nodes from G
2. While there is a swap operation $< s, s' >$ such that $cost(S - s + s') < cost(S)$ do
   $S \leftarrow S - s + s'$
3. return S

In Algorithm 1, $cost(S)$ denotes the sum of all the local costs p_{ij} defined in Chapter 1. Through the execution of the algorithm, there must be a time when we can not find two nodes to swap so that $cost(S - s + s') < cost(S)$; at that moment, the algorithm ends and returns the solution S. For an example, consider the instance shown in Figure 2.1 where $k = 4$. Initially the algorithm selects $S = \{s_1, s_2, s_3, sink\}$. The algorithm first considers the swap $< s_1, s'_1 >$, which yields a solution $S_2 = \{s'_1, s_2, s_3, sink\}$ of lower cost, so S_2 is adopted over S_1 (see Figure

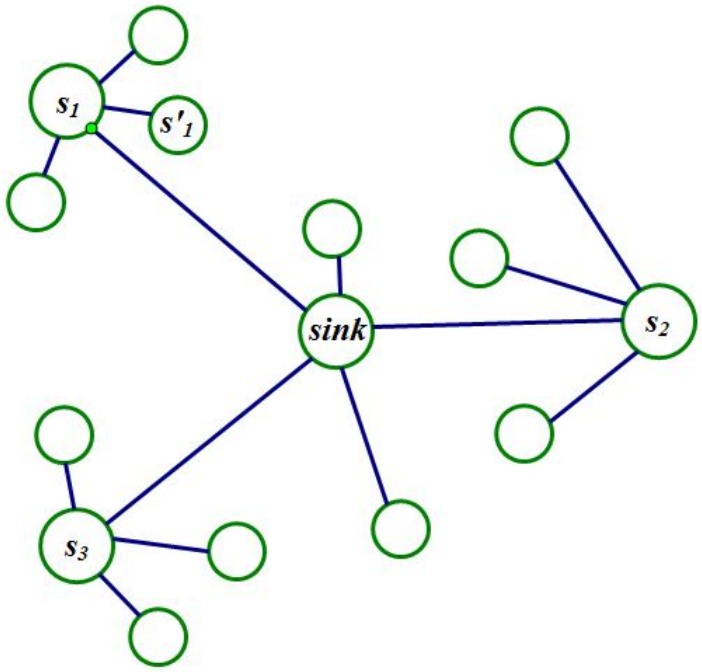

Figure 2.1: Solution S.

2.2). The swap $< s_2, s'_2 >$ then produces a solution $S_3 = \{s'_1, s'_2, s_3, sink\}$ of smaller cost than S_2, so S_3 is taken as the current solution (see Figure 2.3). No swap can further reduce the cost, so S_3 is a local optimum solution.

2.2 Analysis of the Local Search Algorithm for the Uncapcitated Data Storage Placement Problem

Let $N_S(s)$ denote the set of sensor nodes that send their data to storage node $s \in S$ in the local optimum solution produced by algorithm Local-Search and let $N_{S*}(o)$ denote the set of sensor nodes that send their data to storage node $o \in S*$ in an optimum solution $S*$. We now compare the costs of solutions S and $S*$ to determine the quality of the solution computed by *Local Search*. In the sequel, if a sensor node s' sends data to storage node s, we say that *s serves s'*. Sensor nodes and storage nodes will sometimes be called just nodes when the meaning is clear from the context.

Since S is a local optimal solution, we know that any swap $< s, o >$ involving two different nodes $s \in S$ and $o \in S * \backslash S$ is such that

$$cost(S - s + o) \geq cost(S) \tag{2.1}$$

For any nodes $s \in S$ and $o \in S*$ we define $p_s = N_{S*}(o) \cap N_S(s)$ as the set of sensor nodes that are served by, both, s in the local optimum solution and o in the optimum solution.

A *1-1 mapping* is a function $f : V \rightarrow W$ such that $f(v) = f(w) \Rightarrow v = w$ for $v \in V, w \in W$. An *onto mapping* is a function $f' : V \rightarrow W$ such that $\forall w \in W, \exists v \in V$ such that $f'(v) = w$.

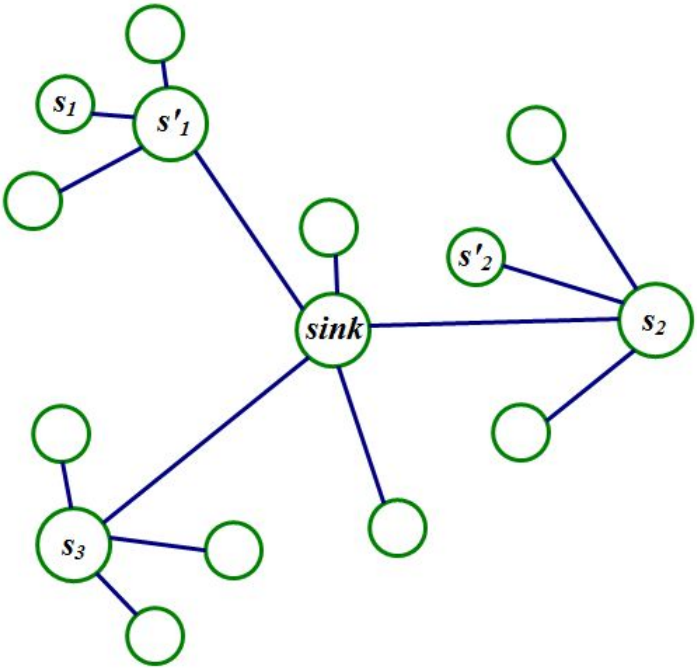

Figure 2.2: Solution S_2.

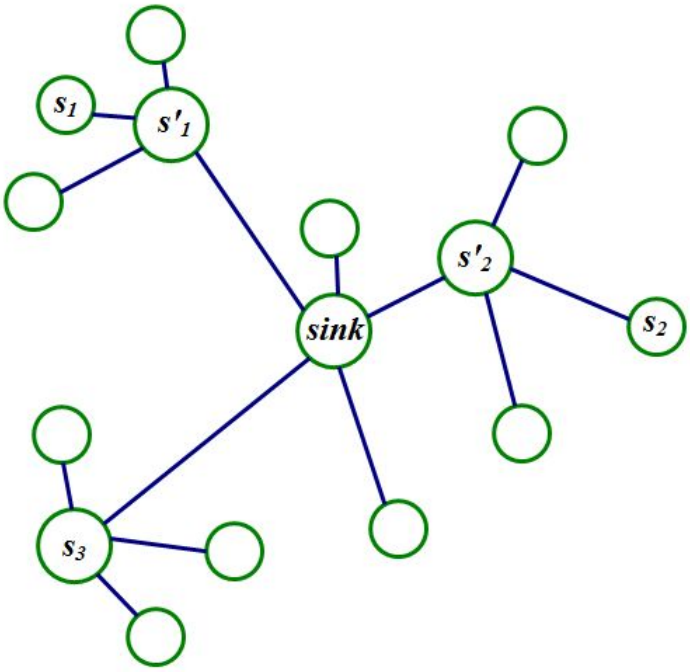

Figure 2.3: Solution S_3.

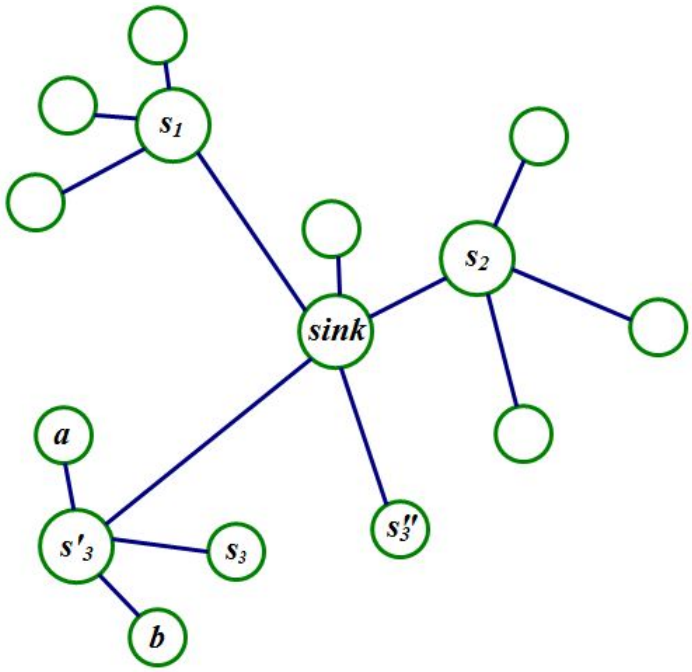

Figure 2.4: Solution S_1.

Consider a 1-1 and onto mapping $\pi : N_{S*}(o) \rightarrow N_{S*}(o)$ with the property that for all $s \in S$ such that $|p_s| \leq \frac{1}{2}|N_{S*}(o)|$, $\pi(p_s) \bigcap p_s = \emptyset$. To see how to build the mapping π, the reader is referred to [5].

We say that a storage node $o \in S*$ is *captured* by a storage node $s \in S$ if $|N_S(s) \bigcap N_{S*}(o)| > \frac{1}{2}|N_{S*}(o)|$. We call a storage node $s \in S$, *bad* if it captures at least one storage node in $S*$ and *good* otherwise. To clarify these concepts consider the example shown in Figures 2.4 and 2.5. The optimum solution $S* = \{s_1, s_2, s_3\}$ for this example is shown in Figure 2.5 and a (non-optimum) solution $S_1 = \{s_1, s_2, s'_3\}$ is shown in Figure 2.4. There are four nodes that send data to storage node s_3 in $S*$: a, b, s'_3 and s_3; also node s'_3 receives data from four nodes in solution S_1: a, b, s_3 and s'_3. Thus, we say that s'_3 captures s_3 because it satisfies $|N_{S_1}(s'_3) \bigcap N_{S*}(s_3)| > \frac{1}{2}|N_{S*}(s_3)|$ and hence s'_3 is a *bad* storage node.

Consider now the solution S_4 shown in Figure 2.6. Node s''_3 is a *good* storage node because it does not *capture* any storage nodes in $S*$ as $|N_{S_4}(s''_3) \bigcap N_{S*}(s_3)| = \{b, s_3\}$, so the number of sensor nodes that send information to s''_3 is just half of the number of nodes in $N_{S*}(s_3)$.

To compare the cost of the solution S computed by the local search algorithm with the cost of an optimum solution $S*$, we consider the following set of k swap operations $< s, o >$ between storage nodes in S and storage nodes in $S*$:

1. If s is a bad storage node that captures only one storage node $o \in S*$, we consider the $swap < s, o >$.

2. Bad storage nodes capturing at least 2 storage nodes from $S*$ will not be swapped.

3. Each good storage node in S is swapped with a storage node in $S*$ which is not considered in Step 1, so that no node in $S*$ is swapped more than twice. Note that since the bad

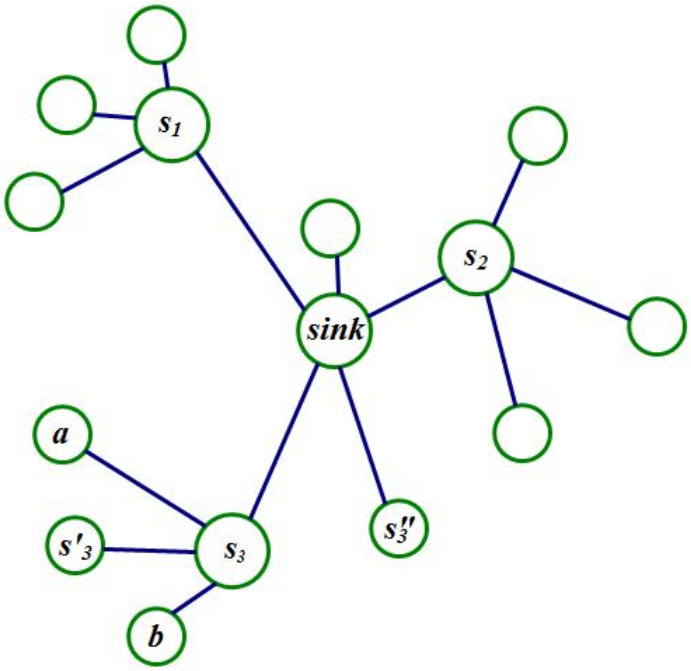

Figure 2.5: Optimum Solution $S*$.

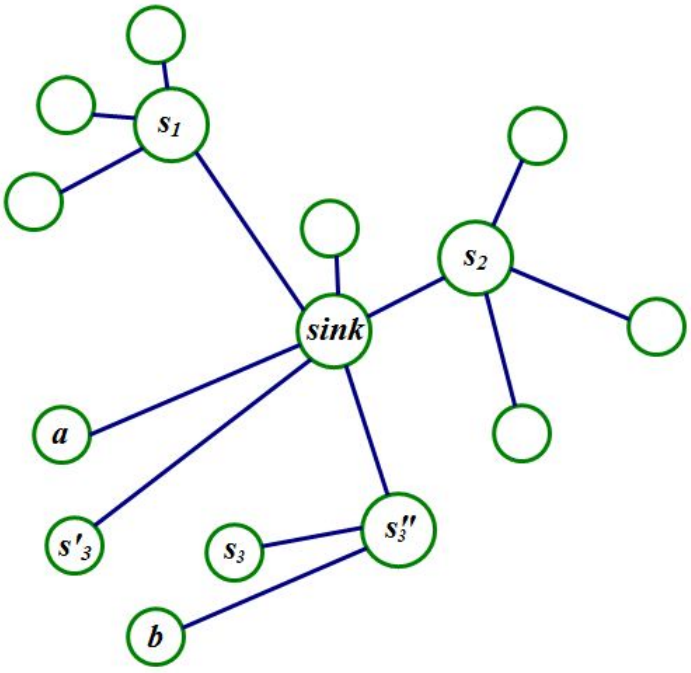

Figure 2.6: Solution S_4.

storage nodes in S capturing at least 2 nodes of $S*$ are not swapped, then the number of good storage nodes in S is at least half of the number of storage nodes in $S*$ which are not considered in step 1.

Note that in the above set of swap operations, if a swap $< s, o >$ is considered, the storage node s does not capture any node $o' \in S*$, $o' \neq o$. When a *swap* $< s, o >$ operation is considered we need to reassign the sensor nodes in $N_S(s) \cup N_{S*}(o)$ to the storage nodes in $S - s + o$. After we perform the swap operation $< s, o >$ all sensor nodes $j \in N_{S*}(o)$ are assigned to o and each sensor node j is assigned to the storage node $s_j \in S - s + o$ for which $p_{s_j j}$ is minimum.

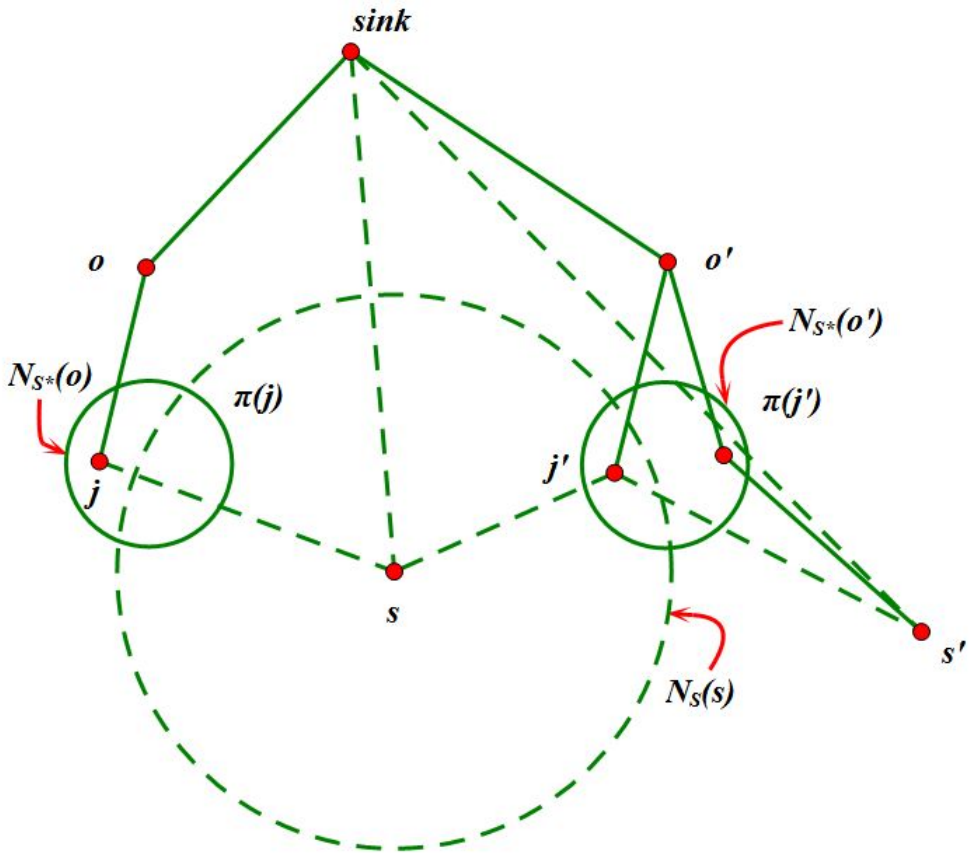

Figure 2.7: Reassigning the nodes in $N_S(s) \cup N_{S*}(o)$.

Consider a node $j' \in N_S(s) \cap N_{S*}(o')$, for $o' \neq o$. As s does not capture o', then $|N_S(s) \cap N_{S*}(o')| \leq \frac{1}{2}|N_{S*}(o')|$ and by the way in which mapping π has been defined, we have that $\pi(j') \notin N_S(s)$. So let $\pi(j') \in N_S(s')$. Note that the cost of assigning j' to a storage node in $S - s + o$ is at most $c_{j's'}$ (see Figure 2.7). Since storage nodes need to forward information to the sink, as explained in Chapter 1, the cost function in which we are interested is $p_{s'j'} = c_{s'j'} + \beta\delta_{s'}$, which indicates that the communication cost between s' and j' is equal to the cost of transmitting information from node j' to the storage node s' plus the cost of sending information from the storage node s' to the sink. So let us analyze $p_{s'j'}$.

Lemma 2.2.1 $p_{s'j'} \leq p_{o'j'} + p_{o'\pi(j')} + p_{s'\pi(j')} - 2\beta\delta_{o'}$

Proof We know that by the triangle inequality, $c_{s'j'} \leq c_{s'\pi(j')} + c_{o'\pi(j')} + c_{o'j'}$, then

$$\begin{aligned} p_{s'j'} &= c_{s'j'} + \beta\delta_{s'} \leq c_{o'j'} + c_{o'\pi(j')} + c_{s'\pi(j')} + \beta\delta_{s'} \\ &\leq c_{o'j'} + c_{o'\pi(j')} + p_{s'\pi(j')} \\ &\leq c_{o'j'} + c_{o'\pi(j')} + p_{s'\pi(j')} + (2\beta\delta_{o'} - 2\beta\delta_{o'}) \\ &= c_{o'j'} + \beta\delta_{o'} + c_{o'\pi(j')} + \beta\delta_{o'} + p_{s'\pi(j')} - 2\beta\delta_{o'} \\ &= p_{o'j'} + p_{o'\pi(j')} + p_{s'\pi(j')} - 2\beta\delta_{o'} \end{aligned}$$

□

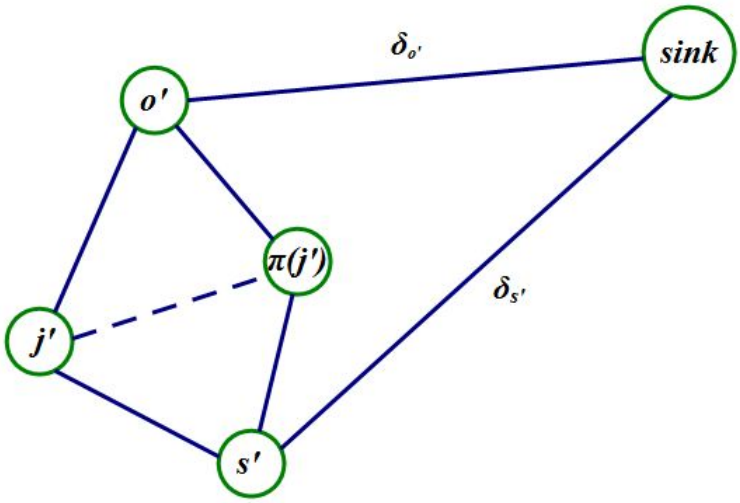

Figure 2.8: Relationship between the nodes in Lemma 2.2.1.

We can now compare $cost(S)$ and $cost(S*)$. First, by inequality (2.1),

$$\begin{aligned} 0 &\leq cost(S - s + o) - cost(S) \\ &= \sum_{j \in N_{S*}(o)} (p_{oj} - p_{sj}) + \sum_{j \in N_S(s), j \notin N_{S*}(o)} (p_{s'_j j} - p_{sj}) \end{aligned}$$

$$\leq \sum_{j \in N_{S*}(o)} (p_{oj} - p_{sj}) + \sum_{j \in N_S(s), j \notin N_{S*}(o)} (p_{oj} + p_{o\pi(j)} + p_{s\pi(j)} - p_{sj} - 2\beta\delta_o) \qquad (2.2)$$

where the last inequality follows from Lemma 2.2.1.

Inequality (2.2) holds for each swap operation $< s, o >$ where $s \in S$ and $o \in S*$. If we consider the set of k swap operations described above, then note that each storage node $s \in S$ participates in at most two swaps; hence the last term in inequality (2.2) added over all k swaps is at most $2 \sum_{j \in N}(p_{o_j j} + p_{o_j \pi(j)} + p_{s_j \pi(j)} - p_{s_j j} - 2\beta\delta_{o_j})$, where o_j is the storage node that collects

the information from sensor node j in the optimum solution $S*$ and s_j is the storage node that collects information from node j in solution S.

Since π is a $1-1$ and onto mapping, then each node j is mapped to a unique node $\pi(j)$ and so

$$\sum_{j \in N} p_{o_j j} = \sum_{j \in N} p_{o_j \pi(j)} = cost(S*).$$

Similarly, $\sum_{j \in N} p_{s_j j} = \sum_{j \in N} p_{s_j \pi(j)}$. Therefore, by the above equation

$$2 \sum_{j \in N} (p_{o_j j} + p_{o_j \pi(j)} + p_{s_j \pi(j)} - p_{s_j j} - 2\beta\delta_o) = 4cost(S*) - 4\beta \sum_{j \in C} \delta_{o_j}.$$

By inequality 2.2 added over all k swap operations, we get:

$$\begin{aligned} 0 &\leq \sum_{j \in N} (p_{o_j j} - p_{s_j j}) + \sum_{j \in N} (p_{o_j j} + p_{o_j \pi(j)} + p_{s_j \pi(j)} - p_{s_j j} - 2\beta\delta_{o_j}) \\ &\leq cost(S*) - cost(S) + 4cost(S*) - 4 \sum_{j \in N} \beta\delta_{o_j}. \\ &\leq 5cost(S*) - cost(S) \end{aligned} \tag{2.1}$$

Hence: $cost(S) \leq 5cost(S*)$

This shows that the approximation ratio of our algorithm is at most 5.

Chapter 3

Approximation Algorithm for the Capacitated Data Storage Placement Problem

3.1 Outline of the Algorithm

In the linear program relaxation of an integer program the requirement that the variables take only integer values is relaxed by allowing them to take real values. We will consider a more general version of integer program (1.1) by allowing each node j to have a non-negative demand d_j. This demand can be regarded as the importance of the node or the size of the data that it generates. We will use this more general formulation of the problem as it is more convenient for expressing and analyzing our algorithm. The linear programming relaxation of this more general version of (1.1) is the following:

$$\begin{aligned} \min & \sum_{i,j \in N} d_j p_{ij} x_{ij} \\ \text{s.t.} & \sum_{i \in N} x_{ij} = 1, \forall j \in N, \\ & \sum_{i \in N} y_i \leq k, \\ & \sum_{j \in N} d_j x_{ij} \leq M y_i, \forall i \in N \\ & y_i \geq x_{ij} \geq 0, \forall i, j \in N, y_0 = 1 \end{aligned} \tag{3.1}$$

In the data storage placement problem, the demand d_j is set to 1 for all nodes j. However, later to analyze the algorithm we will modify the demands of the nodes and that is why we have added the terms d_j to the objective function. Recall that p_{ij} is neither symmetric (i.e. $p_{ij} = c_{ij} + \beta\delta_i$ is not necessarily equal to $p_{ji} = c_{ji} + \beta\delta_j$) nor proportional to the Euclidean distance between i and j, as $p_{ij} = c_{ij} + \beta\delta_i$.

The steps of our algorithm for the capacitated data storage placement problem are outlined below. Each step is analyzed in detail in the following sections.

Step 1: Compute a solution $(\bar{x}, \bar{y})$ for the linear program (3.1). Let $\bar{C}_j^p$ indicate the cost of processing the information generated by sensor node j in solution $(\bar{x}, \bar{y})$; i.e.

$$\bar{C}_j^p = \sum_{i \in N} p_{ij} \bar{x}_{ij}$$

Then the value of the solution $(\bar{x}, \bar{y})$ can be written as:

$$\bar{C}_{LP}(\bar{x}, \bar{y}) = \sum_{j \in N} d_j \bar{C}_j^p$$

Next we find a set N' of nodes that satisfies the following two properties.

- For all $j \in N - N'$ there exists $i \in N'$ such that $c_{ij} \leq 4\bar{C}_j^p$.
- For all $i, j \in N'$, $c_{ij} > 4\, max\{\bar{C}_i^p, \bar{C}_j^p\}$.

We call N' the set of *core nodes*. For every core node i in N' there is a radius $2\bar{C}_i^p$ such that every node $j \in N - N'$ within this radius is closer to i than to any other core node (see Figure 3.1).

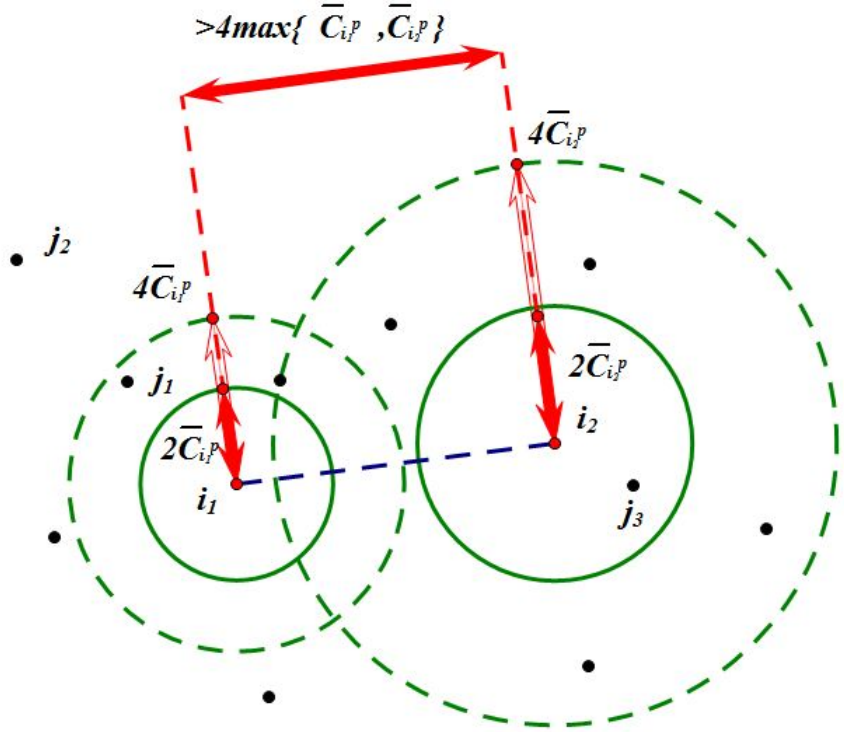

Figure 3.1: Core nodes.

To understand the notion of core nodes, consider the example in Figure 3.1. From the above figure we can see that $c_{i_1 i_2}$ is larger than $4\, max\{\bar{C}_{i_1}^p, \bar{C}_{i_2}^p\}$ which satisfies the second above

property. Node $j_1 \in N - N'$ is no more than $4\bar{C}^p_{i_1}$ away from i_1, so j_1 satisfies the first property. On the other hand, node $j_2 \in N - N'$ is not within the $4\bar{C}^p_{i_1}$ radius of i_1, but it must be within that distance from other core node. Additionally, j_3 is within the $2\bar{C}^p_{i_2}$ radius of i_2, so it is closer to i_2 than to any other core node.

Step 2: Modify the solution $(\bar{x}, \bar{y})$ computed in Step 1 by combining fractional values (a value z is called *fractional* if it is not integer) $\bar{x_{ij}}$ and $\bar{y}_i$ to obtain a new solution (x', y') with fewer variables with positive values. A node $i \in N - N'$ is called a *fractional center* if $\bar{y}_i$ is not integer. Each fractional center is assigned to the closest node in N' (breaking ties arbitrarily). For every fractional center i we consider the set of sensor nodes assigned to it and redistribute them among node i and the node in N' closest to it so that every variable y'_i has value at least $\frac{1}{2}$. Note that the solution created in this manner could violate the capacity constraints because too many sensor nodes could be assigned to a storage node.

Step 3: Further modify the solution obtained in the previous step to produce a {1/2, 1}-integral solution for the linear program. In a {1/2, 1}-integral solution, all variables have value $0, \frac{1}{2}$ or 1.

Step 4: Convert the {1/2,1}-integral solution into an integral solution $(\hat{x}, \hat{y})$ with at most a factor 3 increase in the capacities. In the next sections we explain in detail each step of the algorithm.

3.2 Step 1: Identifying Core Nodes

We first compute a solution $(\bar{x}, \bar{y})$ for the linear program (3.1) using the Ellipsoid algorithm [1, 8] and identify a set N' of core nodes in the following manner. Initially, $N' = \phi$; let $N = \{1, \ldots, n\}$ be the set of nodes indexed in increasing order of $\bar{C}^p_j$, i.e. $\bar{C}^p_1 \leq \cdots \leq \bar{C}^p_n$. Consider the nodes $j = 1, \ldots, n$ in this order. For each node j, if there is no node $i < j$, $i \in N'$ for which $c_{ij} \leq 4\bar{C}^p_j$, then add j to N'. For a node $j \in N$, let $l(j)$ denote the closest node to j in N' breaking ties arbitrarily.

Corollary 3.2.1 $c_{l(j)j} \leq 4\bar{C}^p_j$

Proof Because of the way in which the set N' is chosen, for any node $j \in N$, $c_{l(j)j} \leq 4\bar{C}^p_j$. □

For each node $i \in N'$ we define M_i as the set of nodes in N closer to i than to any other node in N', i.e. $M_i = \{j \in N : l(j) = i\}$. We define the *fractional center value* of M_i as $z_i = \sum_{j \in M_i} y_j$. Since for any $i, h \in N'$, $c_{ih} > 4\bar{C}^p_i$, then each node $j \in N$ within a $2\bar{C}^p_i$ radius of $i \in N'$ must be closer to i than to any other node in N' and so it must belong to M_i.

Note that for any node $j \in N$, $\bar{C}^p_j = \sum_{i \in N} p_{ij}\bar{x}_{ij}$ can be regarded as the weighted average of the costs p_{ij}, where $\bar{x}_{ij}$ is the weight of p_{ij}. Since in any weighted average, less than half of the total weight can be given to values more than twice the average, then

$$\sum_{i: p_{ij} > 2\bar{C}^p_j} \bar{x}_{ij} < \frac{1}{2} \tag{3.2.1}$$

Since $\sum_{i \in N} \bar{x}_{ij} = 1$ and $\bar{x}_{ij} \leq \bar{y}_i$ for each $i \in N$, then

$$\sum_{i: p_{ij} \leq 2\bar{C}^p_j} \bar{y}_i \geq \sum_{i: p_{ij} \leq 2\bar{C}^p_j} \bar{x}_{ij} \geq \frac{1}{2} \tag{3.2.2}$$

Additionally, because $p_{ij} \geq c_{ij}$, then

$$\sum_{c_{ij} \leq 2\bar{C}_j^p} \bar{y}_i \geq \sum_{p_{ij} \leq 2\bar{C}_j^p} \bar{y}_i \geq \frac{1}{2}$$

Thus, for any feasible fractional solution $(\bar{x}, \bar{y})$ of linear program (3.1), $\sum_{i:c_{ij} \leq 2\bar{C}_j^p} \bar{y}_i \geq \frac{1}{2}$ for each $j \in N$. Therefore, the total fractional center value z_i of the set M_i is greater than or equal to $\frac{1}{2}$.

3.3 Step 2: Consolidating Storage Nodes

We modify the solution $(\bar{x}, \bar{y})$ by performing a series of transfer operations as indicated by the following algorithm.

Algorithm 2 Transfer ($\bar{x}$,$\bar{y}$, M_i, i)

Input: Solution $(\bar{x}, \bar{y})$ of linear program (3.1), core node i and set of nodes M_i.
Output: Modified solution (x', y').
1. Initially, set $(x', y') = (\bar{x}, \bar{y})$. Sort the nodes j in M_i in increasing order of c_{ij}, i.e. if the sorted nodes are $j_1, j_2, \ldots, j_m$, then $c_{ij_1} \leq c_{ij_2} \leq \cdots \leq c_{ij_m}$.
2. Let j_t be the first node in the sequence $j_1, \ldots, j_m$ such that $y'_{j_t} < 1$. Let j_s be the first node in the sequence $j_{t+1}, \ldots, j_m$ such that $y'_{j_s} > 0$. If either j_s or j_t does not exist, go to step 6.
3. $q \leftarrow \min(y'_{j_s}, 1 - y'_{j_t})$.
4. For all nodes $j' \in N$, set

$$x''_{j_t j'} \leftarrow x'_{j_t j'} + \frac{q}{y'_{j_s}} x'_{j_s j'}$$
$$x''_{j_s j'} \leftarrow x'_{j_s j'} - \frac{q}{y'_{j_s}} x'_{j_s j'}$$

Set $y''_{j_t} \leftarrow y'_{j_t} + q$, $y''_{j_s} \leftarrow y'_{j_s} - q$.
5. For all $j' \in N$, set $x'_{j_t j'} \leftarrow x''_{j_t j'}$, $x'_{j_s j'} \leftarrow x''_{j_s j'}$, $y'_{j_t} \leftarrow y''_{j_t}$, $y'_{j_s} \leftarrow y''_{j_s}$ and then go back to step 2.
6. If there is a node j_t, $1 < t \leq m$ such that $0 < y'_{j_t} < 1$, then we assign the demand served by j_t to j_{t-1}: For all nodes $j' \in N$, we set

$$x'_{j_{t-1} j'} \leftarrow x'_{j_{t-1} j'} + x'_{j_t j'}$$
$$x'_{j_t j'} \leftarrow 0$$
$$y'_{j_t} \leftarrow 0$$

7. Return (x', y')

We now show that the solution (x'', y'') computed in Step 4 of algorithm Transfer satisfies the third constraint of linear program (3.1) and so in this solution the maximum load of any storage node is M. The proof is by induction on the number of iteration of the loop consisting of steps 2-5 of Transfer. If the number of iteration is zero the claim holds trivially. Assume the claim holds after i iterations. Then for the $(i + 1) - th$ iteration note that the third constraint

is satisfied for all $j_1, j_2, \ldots, j_{t-1}$ by induction hypothesis and the fact that $(\bar{x}, \bar{y})$ is a solution of (3.1). To see that the constraint holds for j_s, note that

$$\begin{aligned}\sum_{j' \in N} d_j x''_{j_s j'} &= \sum_{j' \in N} d_j (x'_{j_s j'} - \frac{q}{y'_{j_s}} x'_{j_s j'}) \\ &= (1 - \frac{q}{y'_{j_s}}) \sum_{j' \in N} d_j x'_{j_s j'} \\ &\leq (1 - \frac{q}{y'_{j_s}}) M y_{j's} \quad \text{by induction hypothesis} \\ &= M(y'_{j_s} - q) = M y''_{j_s}\end{aligned}$$

A very similar argument shows that the constraint also holds for j_t.

Step 6 of Transfer, however might increase the load of node j_{t-1} to at most $2M$ as the whole load of node j_t is transferred to j_{t-1}.

To show how algorithm Transfer works, consider an instance where for core node i, $M_i = \{j_1, j_2, j_3, j_4, j_5\}$, and suppose that $y' = (\frac{2}{3}\ \frac{1}{2}\ \frac{1}{3}\ 0\ 0)$; the values $x'_{j_i j_h}$ for $1 \leq i, h \leq 5$ are given by the following matrix:

$$\begin{pmatrix} \frac{2}{3} & \frac{1}{2} & \frac{1}{2} & \frac{2}{3} & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{3} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{6} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In this example, the sum of values in each column is 1. In the first execution of steps 2-5 algorithm Transfer sets $j_t = j_1$ and $j_s = j_2$; the values in y' change to $(1\ \frac{1}{6}\ \frac{1}{3}\ 0\ 0)$. The matrix x' becomes

$$\begin{pmatrix} \frac{8}{9} & \frac{5}{6} & \frac{13}{18} & \frac{2}{3} & \frac{5}{6} \\ \frac{1}{9} & \frac{1}{6} & \frac{1}{9} & 0 & \frac{1}{6} \\ 0 & 0 & \frac{1}{6} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

because part of the load from the nodes assigned to j_2 is now transferred to j_1. In the second execution of steps 2-5 the algorithm chooses $j_t = j_2$ and $j_s = j_3$. The new vector y' is $y' = (1\ \frac{1}{2}\ 0\ 0\ 0)$ and x' is

$$\begin{pmatrix} \frac{8}{9} & \frac{5}{6} & \frac{13}{18} & \frac{2}{3} & \frac{5}{6} \\ \frac{1}{9} & \frac{1}{6} & \frac{5}{18} & \frac{1}{3} & \frac{1}{6} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Since the value of y'_{j_2} is greater than 0 and less than 1, step 6 is executed to produce the final solution $y' = (1\ 0\ 0\ 0\ 0)$ and

$$x' = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Note that $y'_{j_2} = 0$ which means this node will not receive message from any other nodes and so the load of node j_2 is transferred to j_1; the load of j_1 increases to at most $2M$.

In step 4, variable $x'_{j_s j'}$ is combined with $x'_{j_t j'}$. We say that $x'_{j_s j'}$ is *terminally modified* to node j_t in step 4 if and only if $z_{l(j_s)} = \sum_{i \in M_{l(j_s)}} y'_i \geq 1$, otherwise, the variable is said to be *non-terminally modified*. The reason we call these variables terminally modified and non-terminally modified is because if $z_{l(j_s)} \geq 1$, then the values y'_i of the nodes i in $M_{l(j_s)}$ can be transferred to $l(j_s)$ so $l(j_s)$ will be one of the chosen storage nodes and so sensor node j_s will be "terminally" assigned to $l(j_s)$; otherwise, nodes in $M_{l(j_s)}$ do not accumulate enough value $z_{l(j_s)}$ and thus $l(j_s)$ will not be a storage node, so the sensor nodes in $M_{l(j_s)}$ will be assigned to other nodes. The following two lemmas bound the cost of the solution produced by algorithm Transfer.

Lemma 3.3.1 *For variable $x'_{ij} > 0$ terminally modified* to node $i' \in M_{l(i)}$, $p_{i'j} \leq (3 + 2\beta)p_{ij} + 8(1 + \beta)\bar{C}^p_j$.

Proof By Steps 1 and 2 of algorithm Transfer, i' must appear before i in the ordering of $M_{l(i)}$, thus,

$$c_{l(i)i'} \leq c_{l(i)i}. \tag{3.3.1}$$

Since for any node $j \in N$, $p_{i'j} = c_{i'j} + \beta\delta_{i'}$ and $\delta_{i'} = c_{oi'}$, where o is the sink, then by the triangle inequality (see Figure 3.2),

$$\begin{aligned}
p_{i'j} &= c_{i'j} + \beta\delta_{i'} \leq c_{i'l(i)} + c_{l(i)i} + c_{ij} + \beta(\delta_i + c_{i'i}) \\
&\leq 2c_{l(i)i} + c_{ij} + \beta(\delta_i + c_{i'l(i)} + c_{l(i)i}), \quad \text{by inequality (3.3.1) and the triangle inequality} \\
&\leq 2c_{l(i)i} + c_{ij} + \beta(\delta_i + 2c_{l(i)i}), \quad \text{by inequality (3.3.1)} \\
&\leq 2(1 + \beta)c_{l(j)i} + c_{ij} + \beta\delta_i, \quad \text{as } i \text{ is closer to } l(i) \text{ than to } l(j) \\
&\leq 2(1 + \beta)(c_{l(j)j} + c_{ij}) + c_{ij} + \beta\delta_i, \quad \text{by the triangle inequality} \\
&= 2(1 + \beta)c_{l(j)j} + (3 + 2\beta)c_{ij} + \beta\delta_i \\
&\leq 2(1 + \beta)c_{l(j)j} + (3 + 2\beta)p_{ij} \\
&\leq 2(1 + \beta)p_{l(j)j} + (3 + 2\beta)p_{ij}, \quad \text{as } p_{l(j)j} \geq c_{l(j)j} \\
&\leq 8(1 + \beta)\bar{C}^p_j + (3 + 2\beta)p_{ij}, \quad \text{by Corollary 3.2.1}
\end{aligned}$$

□

Lemma 3.3.2 *For variable $\bar{x}_{ij} > 0$ non-terminally modified to $l(i)$, we have $p_{l(i)j} \leq (2 + \beta)p_{ij} + 4(1 + \beta)\bar{C}^p_j$.*

Proof By the triangle inequality: $c_{l(j)i} \leq c_{ij} + c_{l(j)j}$ and $c_{l(i)j} \leq c_{ij} + c_{l(i)i} \leq c_{ij} + c_{l(j)i}$ because i is closer to $l(i)$ than to $l(j)$. Combining these two inequalities we get $c_{l(i)j} \leq 2c_{ij} + c_{l(j)j}$. In addition, $\beta\delta_{l(i)} \leq \beta(c_{l(i)i} + \delta_i) \leq \beta(c_{l(j)i} + \delta_i) \leq \beta(c_{ij} + c_{l(j)j} + \delta_i)$ by the triangle inequality. Thus, $p_{l(i)j} = c_{l(i)j} + \beta\delta_{l(i)} \leq 2c_{ij} + c_{l(j)j} + \beta c_{ij} + \beta c_{l(j)j} + \beta\delta_i \leq (2 + \beta)p_{ij} + 4(1 + \beta)\bar{C}^p_j$ by Corollary 3.2.1. □

We define the modified demand d'_i of a core node $i \in N'$ as follows.

$$d'_i \leftarrow \sum_{j \in N} d_j x'_{ij}$$

For all nodes $i \in N - N'$, we define $d'_i \leftarrow 0$.

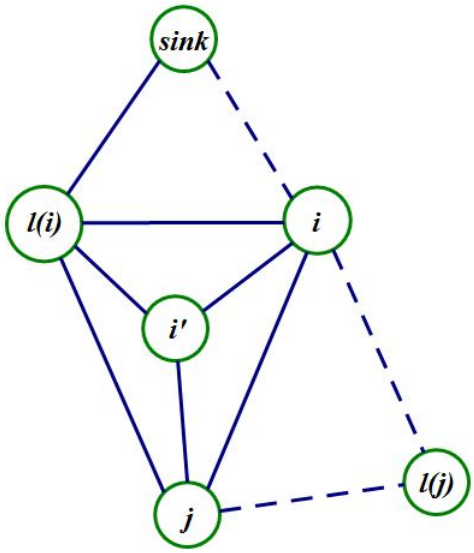

Figure 3.2: Relationship between the nodes in the proof of Lemma 3.3.1.

3.4 Step 3: Obtaining a $\left\{\frac{1}{2}, 1\right\}$-Integral Solution

We now define two new sets of nodes: $N_1 = \left\{i \in N : y'_i > 0\right\}$ and $N_2 = \left\{i \in N : 0 < y'_i < 1\right\}$. Note that according to algorithm Transfer the only nodes that belong to N_2 are those core nodes $i' \in N'$ for which $y'_i < 1$ and $y'_j = 0$ for all $j \in M_i$. Therefore, N_2 is a subset of N'. Let $|N_1| = \ell$ and $|N_2| = m$.

Lemma 3.4.1 *For each node $i \in N_2$, $y'_i \geq \frac{1}{2}$.*

Proof Since i is a core node and $y'_j = 0$ for all $j \in M_i$ except i, then i collects all the information sent by the nodes in M_i. By the discussion at the end of Section 3.2,

$$\frac{1}{2} \leq z_i = \sum_{j \in M_i} y'_j = y'_i$$

□

To obtain a $\left\{\frac{1}{2}, 1\right\}$-integral solution $(\hat{x}, \hat{y})$ from the solution (x', y') computed by algorithm Transfer we first set $\hat{y}_j \leftarrow 1$ for all nodes $j \in N_1 - N_2$. We sort the nodes j in N_2 in increasing order of value $d'_j p_{s(j)j}$: let these nodes be $j_1, \ldots, j_m$. We give the first $2(\ell - k)$ nodes j in this order value $\hat{y}_j = \frac{1}{2}$ and for the rest we set $\hat{y}_j = 1$. We also sort the nodes in N_1 in the same above order. For each node $j \in N_2$, let $s(j)$ be the node in $N_1 - \{j\}$ that has the minimum value $p_{s(j)j}$ breaking ties in favor of a node with smaller index.

Observe that the new solution $(x', \hat{y})$ could be infeasible because the last constraint,

$$\hat{y}_i \geq x'_{ij} \quad \forall i, j \in N$$

of linear program (3.1) might not be satisfied as some values $\hat{y}_j$ are smaller than the corresponding values y'_j.

The following lemma shows that for every node i with value $\hat{y}_i = \frac{1}{2}$ the total demand served by it is at most M.

Lemma 3.4.2 *For any node i,*

1. *If* $\hat{y}_i = \frac{1}{2}$, *then* $\sum_{j \in N} d_j x'_{ij} \leq M$.
2. *If* $\hat{y}_i = 1$, *then* $\sum_{j \in N} d_j x'_{ij} \leq 2M$.

Proof The only nodes i with value $\hat{y}_i = \frac{1}{2}$ belong to N_2, i.e. core nodes for which $y' < 1$ and $y'_j = 0$ for all $j \in M_i$. Algorithm Transfer does not change the value y'_i for these nodes and so by the third constraint of linear program (3.1), $\sum_{j \in N} d_j x'_{ij} \leq M y'_i \leq M$. The last step of algorithm Transfer might change the values of some variables y'_i so that $1 < y'_i < 2$. For such variables y'_i the value of the corresponding variable $\hat{y}_i$ is 1. By the third constraint of linear program (3.1), $\sum_{j \in N} d_j x'_{ij} \leq M y'_i < 2M$. Finally, for those variables y'_i that have value 1, $\hat{y}_i = 1$ and $\sum_{j \in N} d_j x'_{ij} \leq M y'_i \leq M < 2M$. □

The following lemma will help us bound the total cost of our solution.

Lemma 3.4.3

$$\sum_{i \in N_2} (1 - \hat{y}_i) d'_i p_{s(i)i} \leq \sum_{i \in N_2} (1 - y'_i) d'_i p_{s(i)i}$$

.

Proof Note that

$$\sum_{i \in N_1} (1 - y'_i) d'_i p_{s(i)i} = \sum_{i \in N_2} (1 - y'_i) d'_i p_{s(i)i} + \sum_{i \in N_1 - N_2} (1 - y'_i) d'_i p_{s(i)i} = \sum_{i \in N_2} (1 - y'_i) d'_i p_{s(i)i}$$

because $y'_i = 1$ for all $i \in N_1 - N_2$. Hence, to prove the lemma we only need to show that the function $f(y_1, y_2, \ldots, y_\ell) = \sum_{i \in N_1} (1 - y_j) d_i p_{s(i)i}$ has its minimum value when $y_1, y_2, \ldots, y_\ell = \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_\ell$ given that $y_i \geq \frac{1}{2}$ for all $i = 1, 2, \ldots, \ell$.

Since the variables y_i are indexed in non-decreasing order of $d_i p_{s(i)i}$, and $d_i p_{s(i)i} > 0$ for all $i = 1, 2, \ldots, \ell$, function f will have its minimum value when the first variables y_i (with smallest $d_i p_{s(i)i}$ value) have value $\frac{1}{2}$ and the last variables y_i (with large $d_i p_{s(i)i}$ value) have value 1. Since by the second constraint of linear program (3.1), $\sum_{i \in N_1} y_i = k$, then the minimum is achieved when the first $2(\ell - k)$ variables have value $\frac{1}{2}$ and the remaining ones have value 1. Note that with this assignment of values to the variables y_i,

$$\begin{aligned} \sum_{i \in N_1} y_i &= \frac{1}{2} 2(\ell - k) + \ell - (2(\ell - k)) \\ &= \ell - k + \ell - 2\ell + 2k = k \quad \text{as required.} \end{aligned}$$

Thus f achieves its minimum value at $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_\ell$. □

3.5 Step 4: Rounding to an Integral Solution

The last step of the algorithm transforms the solution $(x', \hat{y})$ into an integral solution. To do this we first build a directed graph as follows. Create a node for each $i \in N_1$. For each $i \in N_2$ with $\hat{y}_i = \frac{1}{2}$, add a directed edge from i to $s(i)$; see Figure 3.3.

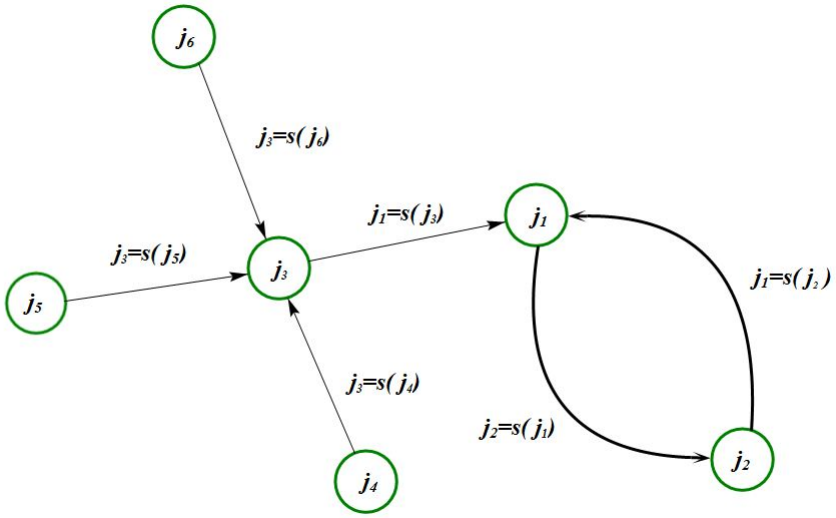

Figure 3.3: Directed graph induced by $s(j)$.

Note that because of the way in which $s(i)$ is chosen for a node i, the only cycles in this directed graph are formed by pairs of vertices (i, j) such that $s(i) = j$ and $s(j) = i$.

For each cycle, we designate one of the vertices as a root node and delete the edge directed from the root to the other vertex to form a tree. If the directed graph has several components, this process transforms the graph into a collection of trees. In the figure, we can see that the edges between nodes j_1 and j_2 form a cycle. We choose j_1 as root and remove the edge from j_1 to j_2 (see Figure 3.4). Observe that the nodes with only directed edges pointing to them are the roots of the trees.

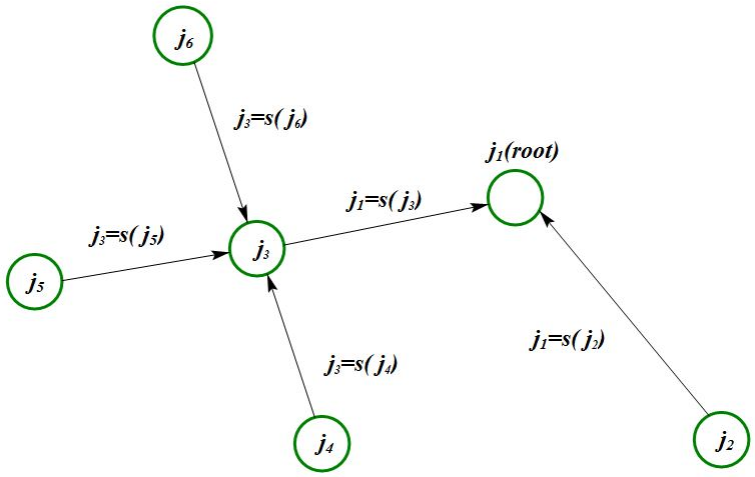

Figure 3.4: Converting the directed graph into a tree.

The above process produces a collection of rooted trees spanning the nodes $i \in N_2$ with $\hat{y}_i = \frac{1}{2}$. In the solution $(x', \hat{y})$ we choose all the nodes i with $\hat{y}_i = 1$ as storage nodes. Additionally, we will choose half of the nodes i with $\hat{y}_i = \frac{1}{2}$ as storage nodes as described below.

We define the *level* of a node in a tree to be the number of edges on the path from the node to the root of the tree. Consider any tree T in the collection. We decompose T into a collection of stars that will be used to select the nodes that will be chosen as storage nodes. Start from a leaf node i of T of highest level and remove the subtree rooted at its parent $s(i)$. This creates a star rooted at $s(i)$. Repeat this procedure until the tree is empty or it consists of a single node i. In the latter case, if $\hat{y}_i = \frac{1}{2}$, we add i to the star rooted at $s(i)$. Recall that $s(i)$ is the node that has the minimum value $p_{s(i)i}$ in $N_1 - \{i\}$. The figure below shows how to split a tree into stars for the above example.

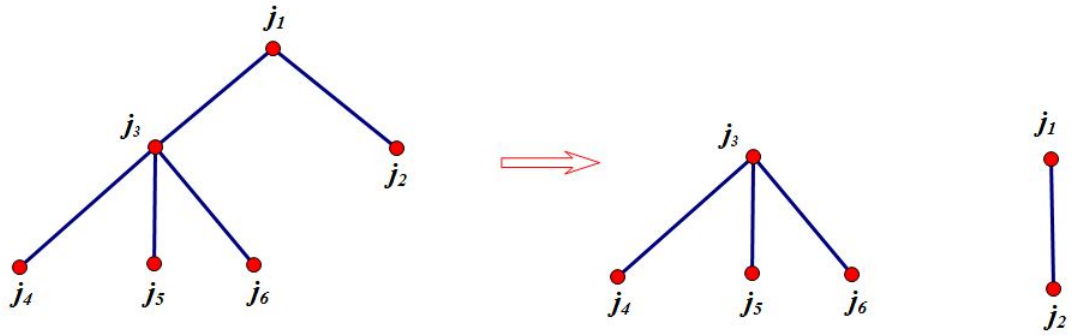

Figure 3.5: Splitting a tree into stars.

Definition A star is called *even* if the sum of the $\hat{y}_i$ values of the nodes i in the star is integer. A star that is not an even star is called an *odd* star.

To determine which of the nodes in a star is going to be selected as a storage node, we need to consider 4 cases depending on whether the star is even or odd and whether the root i of the star has value $\hat{y}_i = 1$ or $\hat{y}_i = \frac{1}{2}$. For each one of these cases we will consider two different selections of storage nodes. At the end we will choose either the selection of storage nodes of smaller cost or the selection with smaller number of nodes, as explained later.

In all cases the root i of a star is selected as a storage node.

1. Even star rooted at a node i with $\hat{y}_i = 1$. Let $j_1, \ldots, j_{2r}$ be the children of i in non-decreasing order of p_{ij_k}. Note that since the star is even and $\hat{y}_i = 1$ then the number of children of i must be even. We consider the two following selections of storage nodes.

   a) Select $j_1, j_3, \ldots, j_{2r-1}$ as storage nodes. Node j_{2s} is assigned to node j_{2s-1} for each $s = 1, 2, \ldots, r$, which means that the demand of j_{2s} is transferred to j_{2s-1}. To reflect this fact the variables x' need to be updated: For each node $i \in N$, set $x'_{j_{2s-1}i} \leftarrow x'_{j_{2s-1}i} + x'_{j_{2s}i}$ and $x'_{j_{2s}i} \leftarrow 0$. In the sequel we will not explicitly indicate how the values of the variables x' are updated. For this case, nodes j_{2s-1}, $s = 1, 2, \ldots, r$, have demand at most $2M$ by Lemma 3.4.2. By the same lemma the demand of node i is at most $2M$.

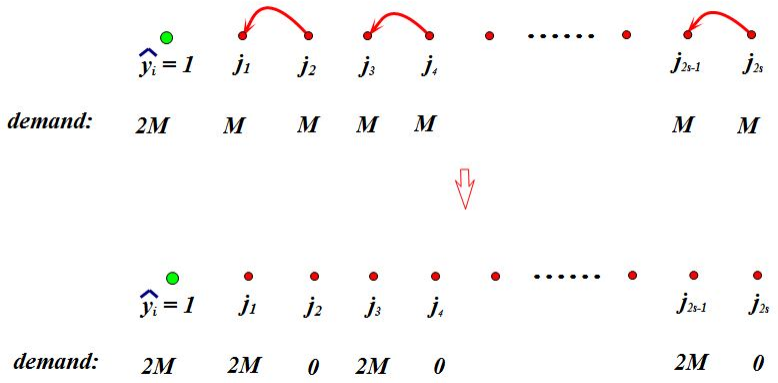

Figure 3.6: Case (1), selection (*a*).

   b) Select nodes $j_2, j_4, \ldots, j_{2r}$ as storage nodes. Node j_1 is assigned to the root i and node j_{2s+1} is assigned to j_{2s} for all $s = 1, 2, \ldots, r-1$. By Lemma 3.4.2 the root i has demand at most $2M$ because $\hat{y}_i = 1$; hence, its demand becomes at most $3M$ after assigning j_1 to it.

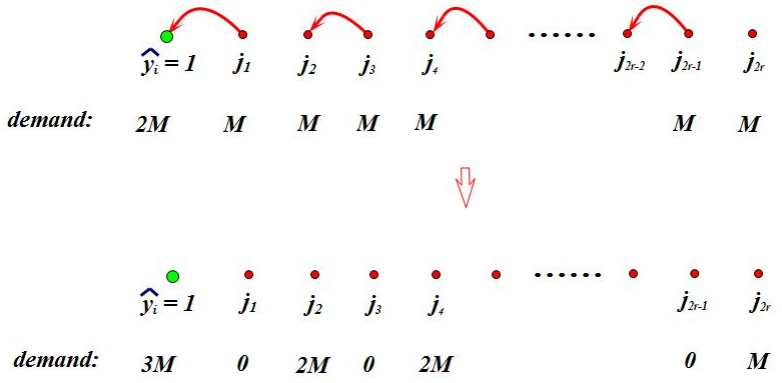

Figure 3.7: Case (1), selection (b).

2. Even star rooted at a node i with $\hat{y}_i = \frac{1}{2}$. Let $j_1, \ldots, j_{2r+1}$ be the children of i in non-decreasing order of p_{ij_k}

   a) Select nodes $j_2, j_4, \ldots, j_{2r}$ as storage nodes. Node j_1 is assigned to the root i and node j_{2s+1} is assigned to node j_{2s} for all $s = 1, 2, \ldots, r$; this will cause the demand of the storage nodes to increase to at most $2M$.

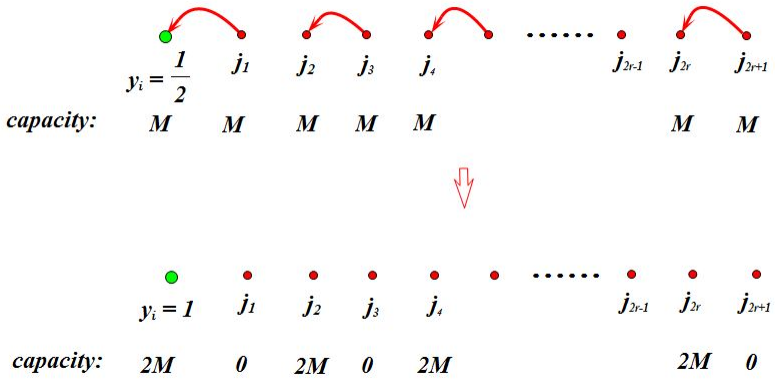

Figure 3.8: Case (2), selection (a).

   b) Select nodes $j_3, j_5, \ldots, j_{2r+1}$ as storage nodes. Nodes j_1 and j_2 are assigned to the root i and node j_{2s+2} is assigned to j_{2s+1} for $s = 1, 2, \ldots, r-1$. The root i will have a demand of at most $3M$ after assigning j_1 and j_2 to it.

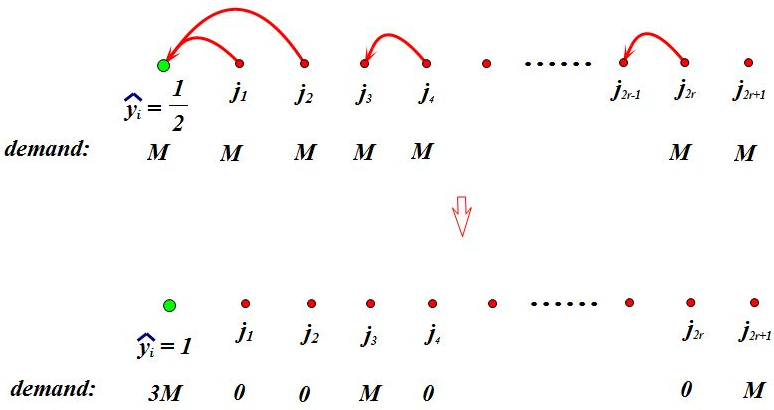

Figure 3.9: Case (2), selection (b).

3. Odd star rooted at node i with $\hat{y}_i = 1$. Let $j_1, \ldots, j_{2r+1}$ be the children of i in non-decreasing order of p_{ij_k}.

   a) Select nodes $j_2, j_4, \ldots, j_{2r}, j_{2r+1}$ as storage nodes. Node j_1 is assigned to the root i. Node j_{2s+1} is assigned to j_{2s} for all $s = 1, 2, \ldots, r-1$. This will cause an increase in the demand of some storage nodes to at most $2M$. Since j_1 moved its demand to the root, the demand for the root increased to at most $3M$.

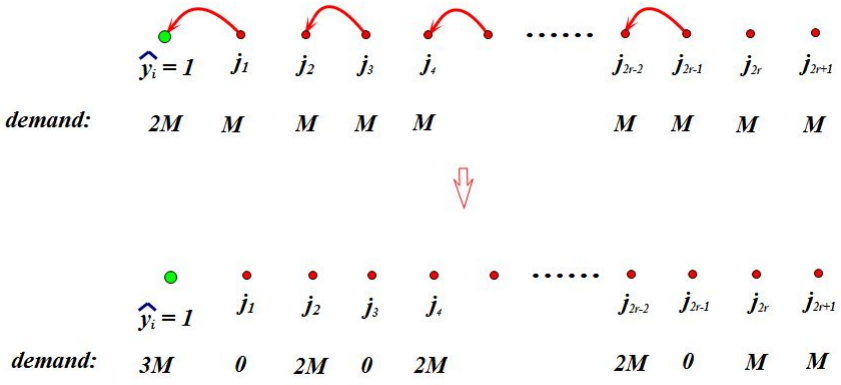

Figure 3.10: Case (3), selection (a).

   b) Select nodes $j_1, j_3, j_5, \ldots, j_{2r-1}$ as storage nodes. Node j_{2r+1} is assigned to the root i and node j_{2s} is assigned to j_{2s-1} for all $s = 1, 2, \ldots, r$. From Lemma 3.4.2, the demand of the root i is $2M$ and then i will have a demand at most $3M$ after assigning j_{2r+1} with demand M to it.

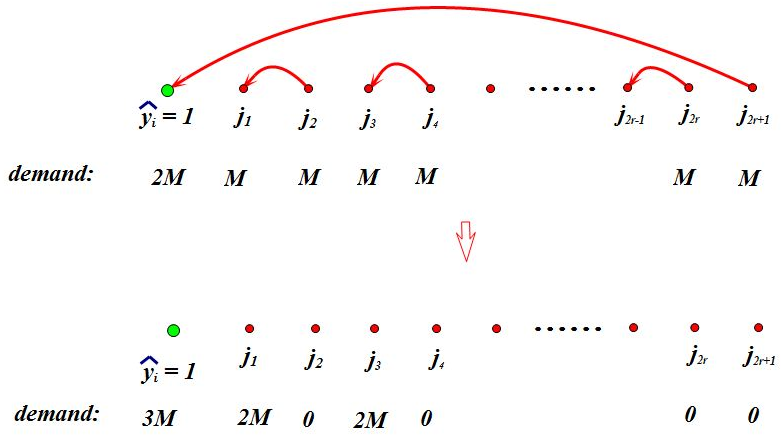

Figure 3.11: Case (3), selection (b).

4. Odd star rooted at i with $\hat{y}_i = \frac{1}{2}$. Let $j_1, \ldots, j_{2r}$ be the children of i in non-decreasing order of p_{ij_k}.

   a) Select nodes $j_2, j_4, \ldots, j_{2r}$ as storage nodes. Node j_1 is assigned to the root. Each node j_{2s+1} is assigned to node j_{2s} for all $s = 1, 2, \ldots, r-1$. In this case, some storage nodes have a demand of at most $2M$.

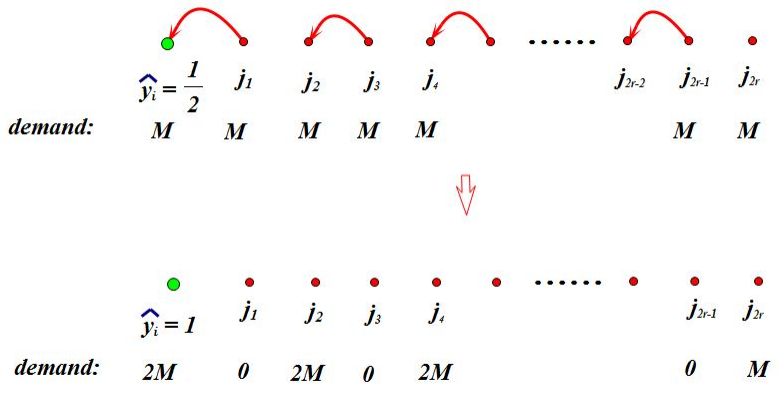

Figure 3.12: Case (4), selection (a).

   b) Select nodes $j_3, j_5, \ldots, j_{2r-1}$ as storage nodes. Nodes j_1 and j_2 are assigned to the root i and each one of the remaining even nodes j_{2s} is assigned to node j_{2s-1}. In this case, the root i has a capacity of at most $3M$.

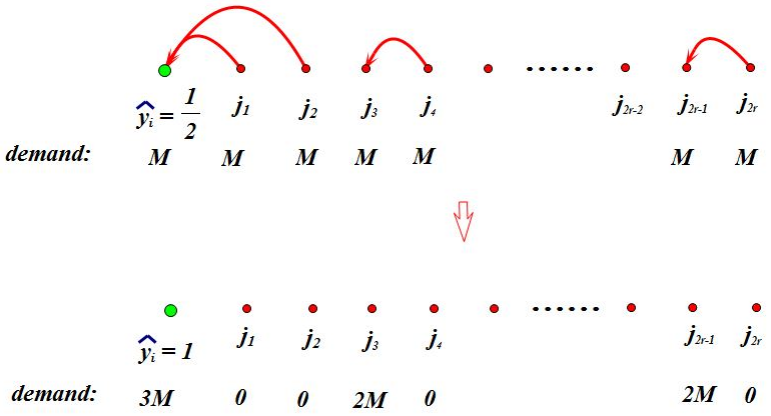

Figure 3.13: Case (4), selection (b).

From the above discussion, we see that the demand for storage nodes can be increased to at most $3M$.

Let $r(i)$ be the storage node to which node i was assigned in the above process. The cost of a selection S of storage nodes is defined to be $\sum_{i \in S} d'_i p_{r(i)i}$. For each even star we choose the selection of storage nodes (a) or (b) of smaller cost. For odd stars we proceed as follows. Note that there is an even number of odd stars $S_1, S_2, \ldots, S_{2q}$ because the total sum of the $\hat{y}_i$ variables must be integer. Let $C_a(\ell)$ denote the cost of star S_ℓ under selection (a) and $C_b(\ell)$ for selection (b). We let the stars be numbered in increasing order of value $|C_a(\ell) - C_b(\ell)|$. We choose the lower cost selection for $S_{q+1}, \ldots, S_{2q}$ and choose the selection with smaller number of centers for $S_1, \ldots, S_q$. Now we analyze the total cost of the stars.

Lemma 3.5.1 *The total cost of the stars computed by the above procedure is bounded by the sum over all stars of the average cost of the two selections (a) and (b).*

Proof For the even stars, we choose the selection of storage nodes of smaller cost, so their total cost is at most the sum of average costs of (a) and (b). For the odd stars, we need to be more careful, as we might choose the larger cost selection for some of them. Consider pairs of odd stars $(S_\ell, S_{2q-\ell+1})$ for $\ell \leq q$; the worst case is when the cost for S_ℓ is the higher of (a) and (b). Note that given two non-negative values f and g,

$$\max(f, g) = \frac{1}{2}|f - g| + \frac{1}{2}|f + g| \qquad \text{and} \tag{3.5.1}$$

$$\min(f, g) = \frac{1}{2}|f + g| - \frac{1}{2}|f - g| \tag{3.5.2}$$

Thus, the cost for pair $(S_\ell, S_{2q-\ell+1})$ can be bounded by

$$\begin{aligned}
&\min(C_a(2q-\ell+1), C_b(2q-\ell+1)) + \max(C_a(\ell), C_b(\ell)) \\
&= \frac{1}{2}|C_a(2q-\ell+1) + C_b(2q-\ell+1)| + \frac{1}{2}|C_a(\ell) + C_b(\ell)| \\
&+ \frac{1}{2}|C_a(\ell) - C_b(\ell)| - \frac{1}{2}|C_a(2q-\ell+1) - C_b(2q-\ell+1)|, \quad \text{By equality 3.5.1 and 3.5.2}
\end{aligned}$$

Consider the last two terms in the above expression. Because of the way in which the costs of the stars S_ℓ were ordered we know that

$$|C_a(\ell) - C_b(\ell)| \leq |C_a(2q - \ell + 1) - C_b(2q - \ell + 1)|.$$

Thus,

$$\begin{aligned}
&\min(C_a(2q - \ell + 1), C_b(2q - \ell + 1)) + \max(C_a(\ell), C_b(\ell)) \\
&\leq \frac{1}{2}|C_a(2q - \ell + 1) + C_b(2q - \ell + 1)| + \frac{1}{2}|C_a(\ell) + C_b(\ell)|
\end{aligned}$$

□

We now use the previous lemma to bound the cost of the stars chosen by the algorithm.

Lemma 3.5.2 *The cost of the stars chosen by the algorithm can be bounded by* $(1+\frac{\beta}{2}) \sum_{j \in N_2, \hat{y}_j = \frac{1}{2}} d'_j p_{s(j)j}$.

Proof Recall that in a star S_ℓ all the children nodes j have value $\hat{y}_j = \frac{1}{2}$. The cost of the root i of S_ℓ is zero as i is selected as a storage node. For the children nodes j we need to consider tree cases:

(1) In one of the selections (a) or (b), node j is chosen as a storage node (so its cost is zero) and in the other selection j is assigned to one of its siblings j'. In the latter case the cost of j is $d'_j p_{j'j}$.

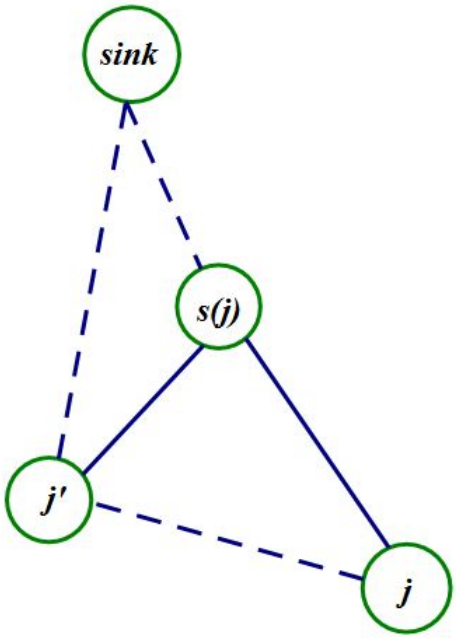

Figure 3.14: Relationship between nodes in Lemma 3.5.2.

Note that (see Figure 3.14)

$$\begin{aligned} p_{j'j} &= c_{j'j} + \beta\delta_{j'} \le c_{s(j)j} + c_{s(j)j'} + \beta(\delta_{s(j)} + c_{s(j)j'}), \quad \text{by the triangle inequality} \\ &\le 2c_{s(j)j} + \beta c_{s(j)j} + \beta\delta_{s(j)}, \quad c_{s(j)j'} \le c_{s(j)j} \text{ by the way in which storage nodes are chosen} \\ &< (2+\beta)c_{s(j)j} + \beta\delta_{s(j)} \\ &< (2+\beta)p_{s(j)j} \end{aligned}$$

Hence, the average cost of j is at most $\frac{1}{2}(0 + (2+\beta)d'_j p_{s(j)j}) = (1 + \frac{\beta}{2})d'_j p_{s(j)j}$.

(2) In one of the selections (a) or (b), j is chosen as a storage node and in the other it is assigned to the root. In the second case the cost of j is $d'_j p_{s(j)j}$.

(3) Node j is assigned to the root in both selections (a) and (b). The cost of j is then $d'_j p_{s(j)j}$.

In the three cases the average cost of j is at most $(1 + \frac{\beta}{2})d'_j p_{s(j)j}$. By Lemma 3.5.1 the cost of the stars is as stated. □

We are now ready to bound the cost of the non-terminal assignments.

Lemma 3.5.3 *In the solution provided by our algorithm, the cost of the non-terminal assignments increases by the cost of the stars, which is* $2(1 + \frac{\beta}{2}) \sum_{j \in N_2} (1 - y'_j) d'_j p_{s(j)j}$.

Proof Recall that we have defined $d'_j = \sum_{j \in N} d_j x'_{ij}$. Consider all the non-terminal assignments from nodes j to node i, that are re-assigned in the above procedure to $r(i)$:

$$\begin{aligned} &\sum_j d_j x'_{ij} p_{r(i)j} = \sum_j d_j x'_{ij} c_{r(i)j} + \sum_j d_j x'_{ij} \beta\delta_{r(i)} \\ &\le \sum_j d_j x'_{ij} c_{r(i)i} + \sum_j d_j x'_{ij} c_{ij} + \sum_j d_j x'_{ij} \beta\delta_{r(i)}, \quad \text{by the triangle inequality} \\ &\le \sum_j d'_j p_{r(i)i} + \sum_j d_j x'_{ij} p_{ij} \end{aligned}$$

The first term of the right hand side of the last inequality is the cost of the stars and the second term is the cost of the non-terminal assignments before step 4. The nodes in the stars are all in N' and for all the children nodes j, $y'_j = \frac{1}{2}$, so using Lemma 3.5.2, the first term can be bounded by:

$$(1 + \frac{\beta}{2}) \sum_{j \in N_2, \hat{y}_j = \frac{1}{2}} d'_j p_{s(j)j} = (1 + \frac{\beta}{2}) \sum_{j \in N_2} 2(1 - \hat{y}_j) d'_j p_{s(j)j} \le (1 + \frac{\beta}{2}) \sum_{j \in N_2} 2(1 - y'_j) d'_j p_{s(j)j}$$

The last inequality follows from Lemma 3.4.3. □

Chapter 4

Computing the Total Cost of the Solution and the Time Complexity of the Algorithm

In this chapter we analyze the total cost of the solution for the data storage placement problem produced by our algorithm and compute its time complexity. We have proven some lemmas in the previous chapters that will help us compute the total cost of the solution.

4.1 Analyzing the Total Cost of the Solution

In Section 3.3 we computed a solution (x', y'), and for some of the sensor nodes j we produced terminally modified assignments x_{ij}. These assignments are part of the final solution, i.e. if $x_{ij} = 1$ then i is a storage node in the final solution and sensor node j is assigned to it. For other sensor nodes j we produced non-terminally modified assignments. For such nodes the final selection of storage nodes to which they will be assigned is made in Section 3.4 and 3.5. To determine the total cost of the solution we need to compute the cost of the terminally and non-terminally modified assignments. The cost of the terminally modified assignments is given by Lemma 3.3.1:

$$\sum_{j \in N} \sum_{i: z_{l(i)} \geq 1, i \in N'} ((3 + 2\beta)p_{ij} + 8(1 + \beta)\bar{C}_i^p)d_j x'_{ij} \tag{4.1}$$

Computing the cost of the non-terminal assignments is more complicated. By Lemma 3.3.2 the cost of these assignments after step 2 is

$$\sum_{j \in N} \sum_{i: z_{l(i)} < 1} ((2 + \beta)p_{ij} + 4(1 + \beta)\bar{C}_j^p)d_j x'_{ij} \tag{4.2}$$

Steps 3 and 4 change the non-terminal assignments and by Lemma 3.5.3 they increase the cost of these assignments by

$$\begin{aligned}
2(1+\frac{\beta}{2})\sum_{j\in N_2}(1-y'_j)d'_j p_{s(j)j} &= 2(1+\frac{\beta}{2})\sum_{j\in N_2, z_j<1}(1-y'_j)d'_j p_{s(j)j} \quad \text{(as } z_j<1 \text{ for all } j\in N_2\text{)} \\
&\le 2(1+\frac{\beta}{2})\sum_{i\in N}\sum_{j\in N_2, z_j<1}(1-y'_j)d_i x'_{ji} p_{s(j)j}, \quad \text{by the definition of } d' \\
&= 2(1+\frac{\beta}{2})\sum_{i\in N}[\sum_{j\in N_2, z_j<1, j\neq l(i)}(1-y'_j)d_i x'_{ji} p_{s(j)j} + \sum_{j\in N_2, z_j<1, j=l(i)}(1-y'_j)d_i x'_{ji} p_{s(j)j}]
\end{aligned} \tag{4.3}$$

We consider each one of the last two terms separately. First we consider the case $j \neq l(i)$. Because $s(j)$ was defined as the node in $N_1 - \{j\}$ with minimum value $p_{s(j)j}$, then $p_{s(j)j} \le p_{l(i)j}$. Recall that $M_j = \{i' | l(i') = j\}$. Hence for each $i' \in M_j$:

$$p_{s(j)j} \le p_{l(i)j} \le c_{ji'} + c_{i'l(i)} + \beta\delta_{l(i)} \le 2c_{i'l(i)} + \beta\delta_{l(i)} \le 2p_{l(i)i'} \tag{4.4}$$

The third inequality holds as $i' \in M_j$. Using the fact that $c_{i'l(i)} \le c_{i'i} + c_{il(i)}$, we get

$$\begin{aligned}
p_{l(i)i'} &= c_{i'l(i)} + \beta\delta_{l(i)} \\
&\le c_{i'i} + c_{il(i)} + \beta\delta_{l(i)} \\
&\le c_{i'i} + c_{il(i)} + \beta(\delta_{i'} + c_{il(i)} + c_{i'i}) \qquad \text{(by the triangle inequality)} \\
&\le (1+\beta)(p_{i'i} + c_{il(i)})
\end{aligned}$$

Thus, from inequality (4.4) we get

$$p_{s(j)j} \le 2(1+\beta)(p_{i'i} + c_{il(i)}) \tag{4.5}$$

By Corollary 3.2.1 and Inequality (4.5) and since $x'_{ji} = \sum_{i', l(i')=j} \bar{x}_{i'i}$, the first term of inequality (4.3) can be bounded by

$$\begin{aligned}
&2(1+\frac{\beta}{2})\sum_{i\in N}\sum_{j\in N_2, z_j<1, j\neq l(i)}(1-y'_j)d_i x'_{ji} p_{s(j)j} \\
&\le 2(1+\frac{\beta}{2})\sum_{i\in N}\sum_{j\in N_2, z_j<1, j\neq l(i)}\sum_{i'\in N, l(i')=j}(1-y'_j)d_i \bar{x}_{i'i} 2(1+\beta)(p_{i'i} + 4\bar{C}_i^p)
\end{aligned} \tag{4.6}$$

Since we assume $j \neq l(i)$ and in the above summation $j = l(i')$ then $l(i) \neq l(i')$ for all i'. Furthermore, by Lemma 3.4.1, $y'_j \ge \frac{1}{2}$ for all non-zero y'_j, so $2(1-y'_j) \le 1$. Thus the right hand side of Inequality (4.6) is bounded by

$$(1+\frac{\beta}{2})\sum_{i\in N}\sum_{i'\in N, z_{l(i')}<1, l(i')\neq l(i)} d_i \bar{x}_{i'i}(1+\beta)(2p_{i'i} + 8\bar{C}_i^p) \tag{4.7}$$

We now consider the second term in the right hand side of (4.3), so let $i, j \in N_2$ and $j = l(i)$. Since $j \in N_2$ then $y'_j < 1$. Hence, by the first and forth constraints of linear program

(3.1), there must be at least one node $i' \notin M_j$ for which $x_{i'j} > 0$. Furthermore, $\bar{C}_j^p = \sum_{h \in N} p_{hj}\bar{x}_{hj} = \sum_{h \in M_j} p_{hj}\bar{x}_{hj} + \sum_{h \notin M_j} p_{hj}\bar{x}_{hj}$. Let $i' \notin M_j$ be the node with minimum $p_{i'j}$ value, then $\sum_{h \notin M_j} p_{i'j}\bar{x}_{hj} \leq \sum_{h \notin M_j} p_{hj}\bar{x}_{hj} \leq \bar{C}_j^p$. By the first and fourth constraints of (3.1), $\sum_{h \notin M_j} \bar{x}_{hj} = 1 - y'_j$ and therefore

$$\sum_{h \notin M_j} p_{i'j}\bar{x}_{hj} = p_{i'j}(1 - y'_j) \leq \bar{C}_j^p \tag{4.8}$$

Note that since $i' \notin M_j$ then $l(i') \neq j$, and so $l(i')$ will be closer to i' than j, so

$$c_{l(i')i'} \leq c_{ji'}. \tag{4.9}$$

For each node $j \in N_2$ we defined $s(j)$ as the node in $N_1 - \{j\}$ for which $p_{s(j)j}$ is minimum, so

$$p_{s(j)j} \leq p_{l(i')j} \tag{4.10}$$

By the triangle inequality, $p_{l(i')j} = c_{l(i')j} + \beta\delta_{l(i')} \leq (c_{ji'} + c_{l(i')i'}) + \beta\delta_{l(i')} \leq (c_{ji'} + c_{l(i')i'}) + \beta(\delta_{i'} + c_{l(i')i'}) \leq 2c_{ji'} + \beta(\delta_j + c_{ji'})$. Therefore, $p_{s(j)j} \leq p_{l(i')j} \leq 2c_{ji'} + \beta(\delta_{i'} + c_{ji'}) \leq 2(1+\beta)c_{ji'} + \beta\delta_{i'} \leq 2(1+\beta)p_{i'j}$. Since $j = l(i)$ then in Step 1 of the algorithm (see Section 3.2), node j was indexed before node i, so $\bar{C}_j^p \leq \bar{C}_i^p$, and hence by Inequality (4.8):

$$(1 - y'_j)p_{s(j)j} \leq 2(1+\beta)(1 - y'_j)p_{i'j} \leq 2(1+\beta)\bar{C}_j^p \leq 2(1+\beta)\bar{C}_i^p \tag{4.11}$$

Finally, the second term in the right hand side of inequality (4.3) can be written as

$$2(1+\frac{\beta}{2}) \sum_{i \in N} \sum_{j \in N_2, z_j < 1, j = l(i)} \sum_{i' : l(i') = j} d_i \bar{x}_{i'i}(1 - y'_j)p_{s(j)j} \leq (1+\frac{\beta}{2}) \sum_{i \in N} \sum_{i' \in N, z_{l(i')} < 1, l(i') = l(i)} 4(1+\beta)\bar{C}_i^p d_i \bar{x}_{i'i} \tag{4.12}$$

Adding (4.2) to the increase, (4.7) and (4.12), the total cost of the non-terminal assignments is at most

$$(1+\frac{\beta}{2}) \sum_{j \in N} \sum_{i' \in N, z_{l(i')} < 1} ((4+3\beta)p_{i'j} + 12(1+\beta)\bar{C}_j^p)d_j \bar{x}_{i'j}$$

Adding this to the total cost of the terminal assignments in (4.1) gives an upper bound on the total cost of the solution:

$$\begin{aligned}
&(1+\frac{\beta}{2}) \sum_{j \in N} \sum_{i' \in N} ((4+3\beta)p_{i'j} + 12(1+\beta)\bar{C}_j^p)d_j \bar{x}_{i'j} \\
&\leq (1+\frac{\beta}{2}) \sum_{j \in N} ((4+3\beta)\bar{C}_j^p + 12(1+\beta)\bar{C}_j^p)d_j \\
&\leq (1+\frac{\beta}{2}) \sum_{j} (16+15\beta)\bar{C}_j^p d_j = (1+\frac{\beta}{2})(16+15\beta)\bar{C}_{LP}(\bar{x},\bar{y}) = (16+23\beta+\frac{15}{2}\beta^2)\bar{C}_{LP}(\bar{x},\bar{y})
\end{aligned}$$

where $\bar{C}_{LP}(\bar{x},\bar{y})$ is the value of the solution for linear program (3.1). Recall that $\bar{C}_{LP}(\bar{x},\bar{y})$ is less than or equal to the optimum solution for the capacitated data storage placement problem. Thus our algorithm has approximation ratio $16 + 23\beta + \frac{15}{2}\beta^2$,

4.2 Time Complexity of Our Algorithm

First we analyze the amount of time needed for solving the linear program (3.1). We solve (3.1) by using Khachiyan's ellipsoid algorithm [1], described below. Linear program (3.1) can be written as follows:

$$\begin{aligned}
\text{LP: min } & \sum_{i,j\in N} d_j p_{ij} x_{ij} \\
\text{s.t. } & \sum_{i\in N} x_{ij} = 1, \forall j \in N, \\
& \sum_{i\in N} y_i - k = 0, \\
& My_i - \sum_{j\in N} d_j x_{ij} \geq 0, \forall i \in N \\
& y_i - x_{ij} \geq 0 \forall i, j \in N. \\
& y_i \geq 0, x_{ij} \geq 0, \forall i, j \in N, y_0 = 1.
\end{aligned}$$

We let A be the constraint matrix for this linear program which then can be written as

$$\begin{aligned}
&\min c^T x \\
&\text{s.t. } Ax \geq b
\end{aligned}$$

where b is a vector containing the right hand side values of the constraints and c is the cost vector. Since there are n nodes in N, the matrix A has $\eta = n * n$ columns as there is a variables x_{ij} for each $i \in N$ and $j \in N$ and it has $2n^2 + n + 1$ rows as that is the number of constraints.

We define $L = n^2 * n^2 + \sum_{i,j\in N} \log(c_{ij}) \leq n^4 + n^2 \log c_{max} = O(n^4)$ where c_{max} is the largest distance between a sensor node and a storage node; we assume that the distances c_{ij} are polynomial in the number of nodes, so $\log c_{max}$ is $O(logn)$. L is an upper bound on the number of bits needed to store matrix A and vector b, c. The Ellipsoid Algorithm is presented below, in Algorithm 3.

Now we prove that the time needed by the ellipsoid algorithm to solve the above linear program is $O(n^{42} \log^2 n)$.

Lemma 4.2.1 *The running time of the ellipsoid algorithm on linear programming LP is* $O(n^{42} \log^2 n)$

Proof The number of iterations of Step 2 of the ellipsoid algorithm is $B = \eta^2 L = O(n^8)$.

Each iteration of the loop in step 2 needs to verify that $(x, y)_k$ is a feasible solution. This requires $O(n^2)$ arithmetic operations as all constraints need to be tested. Further more, each iteration of the loop in Step 2 performs $O(n^6)$ operations for the computations in Step 2.2 because the size of the matrix A_k is $O(n^2) \times O(n^2)$ and v is a vector of length n^2; therefore, the total number of the operations performed by the loop is $O(n^{14})$. Each entry in A_0 has $O(\log n)$ bits and every distance c_{ij} is encoded with $O(\log n)$ bits, therefore through all the iterations of the loop the largest number of bits needed to encode any value is the total number of arithmetic operations times the maximum number of bits which is at most $O(n^{14}) * O(\log n) = O(n^{14} \log n)$.

Each iteration of the loop performs $O(n^6)$ operations on values encoded with $O(n^{14} \log n)$ bits; the most expensive operations are multiplications and each one of them needs $O(n^{28} \log^2 n)$

Algorithm 3 Ellipsoid (A, b, C, n, L)

Input: Constraint matrix A, right hand side vector b, cost vector C for linear program LP, number of nodes n and bound L for the number of bits needed to store A, b and c.
Output: An optimum solution for program LP, if one exists.
0. Let η be the number of columns of A.
1. Set the value of each entry in initial solution $(x, y)_0$ to be 0 i.e. $(x, y)_0 = 0$ and $A_0 = \eta^2 L^2 I$, where I is the identity matrix of size $\eta * \eta$ and $(x, y)_0$ is an $\eta - vector$.
2. For all values $k = 0, 1, \ldots, \eta^2 L$ do:

(2.1) If $(x, y)_k$ is a feasible solution for LP, label k as a *feasible index* and set $v = C$. Else let $(x, y)_k$ violate the $i - th$ constraint of LP; set v to be the $i - th$ row of A and label k as an *infeasible index*.

(2.2) Compute

$$q_k = \frac{A_k v}{\sqrt{v^T A_k v}}$$

$$(x, y)_{k+1} = (x, y)_k - \frac{q_k}{n+1}$$

$$A_{k+1} = \frac{n^2}{n^2 - 1}(A_k - \frac{2}{n+1} q_k q_k^T)$$

3. Output the solution $(x, y)_k$ for which $C^T(x, y)_k = \max\{C^T(x, y)_i | i \text{ is a feasible index}\}$.

time, so one iteration of the loop can be performed in $O(n^{34} \log^2 n)$ time. Since the loop is repeated $O(n^8)$ times, the total time is $O(n^{42} \log^2 n)$. In step 1 of the Ellipsoid algorithm, the time needed to build the matrix A_0 is $O(\eta) = O(n^2)$ because we only deal with the diagonal element of the matrix I. In step 3, we find out the maximum $C^T(x, y)_k$ and that costs $O(k) = O(\eta^2 L) = O(n^8)$ time. Overall, the total running time of the algorithm is $O(n^{42} \log^2 n)$ □

We now analyze the running time of the remaining steps of our algorithm for the capacitated data storage placement problem (CDSP).

In Step 1 described in Section 3.2, which identifies the core nodes, sorting the sensor nodes in increasing order of $\bar{C}_j^p$ value needs time $O(n \log n)$. Computing each value $\bar{C}_j^p$ needs $O(n)$ time. To identify the nodes in N' we need to consider one by one the nodes and for each one of them we need to perform a linear search over the set N' to find the node closest to it. Since a linear search takes $O(n)$ time and we need to perform them n times, the total running time of Step 1 is $O(n^2)$.

In Step 2 described in Section 3.3, the nodes in each set M_i need to be sorted and that takes $O(|M_i| \log |M_i|)$ time. Then the nodes in each set M_i are sequentially scanned and the values of the variables x', y' are updated; this takes time $O(|M_i|)$. Since the total number of nodes in all the sets M_i is n, the total time needed by this step is $O(n \log n)$.

In Step 3 described in Section 3.4, we need to compute $s(j)$ for each node j. This requires $O(n^2)$ time. The nodes in N_2 are then sorted and the values of the variables $\hat{y}_i$ are computed; this needs $O(n \log n)$ time. Thus Step 3 needs $O(n^2)$ time.

In Step 4 described in Section 3.5, we make a set of trees in $O(n)$ time. The trees are traversed and split into stars. This needs $O(n)$ time. In each star the children of the root are sorted and then the demands are modified. This requires $O(n \log n)$ time.

Solving the linear program requires the largest amount of time, so the total time complexity of the algorithm for the CDSP problem is $O(n^{42} \log^2 n)$.

Chapter 5

Conclusions

The data storage placement problem models many applications in sensor networks e.g., monitoring learning behavior of children, senior care systems and environment sensing. In sensor networks the storage nodes are needed to reduce the amount of information that is sent to the sink; sensors have limited storage space and battery life. In some applications, the data accumulated on each storage node can be transported periodically to a data warehouse by robots or vehicles using physical mobility such as Data Mule [48]. In early sensor networks, the sink node periodically queries the sensor nodes for information by broadcasting query messages [27, 28, 42]. Sensors return data back to the sink upon receiving a query. Those systems were not efficient as they do not take into account the amount of information that needs to be sent to the sink and the limited storage capacity of the sensor nodes. We are the first to consider the limited capacity of storage nodes in the data storage placement problem and we call this the capacitated data storage placement problem.

In this thesis, we study the capacitated storage node placement problem in sensor networks. We have designed an approximation algorithm for the capacitated data storage placement problem based on Guha's algorithm for the capacitated k-median problem. The approximation ratio of our algorithm is $(16 + 23\beta + \frac{15}{2}\beta^2)$ where β measures the relative cost of sending information to the sink compared to the cost of sending information to the storage nodes. The solution produced by our algorithm might surpass the capacities of the storage nodes by factor of at most 3. The algorithm consists of 4 steps: Obtaining a solution for a linear programming formulation of the problem; consolidating storage nodes identified by this solution; transforming the solution to a $\left\{\frac{1}{2}, 1\right\}$-integral solution; rounding the solution to an integral one. We also analyzed a local search algorithm of Arya et al. [5] when applied to the uncapacitated data storage placement problem and show that it has approximation ratio 5, improving on the algorithm by Sheng et al. which has approximation ratio 10.

We note that Guha's algorithm for the capacitated k-median problem is not applicable to our problem because our cost function is not symmetric and it does not satisfy the triangle inequality. Guha's algorithm strongly relies on these properties.

One way in which we could try to improve the approximation ratio of our algorithm is by using a primal-dual approach, as described in [12] or a combination of a greedy approach, scaling, and a linear programming approach as used in [12]. One of the benefits of the primal-dual method is that it leads to a very general methodology for the design of approximation algorithms for NP-hard problems. It is one of the few existing general methods for designing

approximation algorithm [22].

There are some open problems we can consider in the future. It would be interesting to extend our algorithm to the hierarchical placement problem (HPP). The HPP requires distributing caches amoung nodes in a hierarchical network [24]. For example, the small caches should be placed close to the users while larger ones should reside the backbone of the network [16, 54]. This problem arises when trying to alleviate network congestion and accelerate access to users information access and solving the problem could improve network performance [21]. Another interesting problem is when there is an opening cost for each storage node. This problem is related to the k-facility location problem.

Bibliography

[1] The ellipsoid algorithm for linear programming. `https://www.cs.princeton.edu/courses/archive/fall05/cos521/ellipsoid.pdf`.

[2] Jon Agre and Loren Clare. An integrated architecture for cooperative sensing networks. *Computer*, 33(5):106–108, 2000.

[3] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

[4] Matthew Andrews and Lisa Zhang. The access network design problem. In *39th Annual Symposium on Foundations of Computer Science, 1998. Proceedings*, pages 40–49. IEEE, 1998.

[5] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[6] Manish Bhardwaj, Timothy Garnett, and Anantha P Chandrakasan. Upper bounds on the lifetime of sensor networks. In *IEEE International Conference on Communications, 2001. ICC 2001*, volume 3, pages 785–790. IEEE, 2001.

[7] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Querying the physical world. *Personal Communications, IEEE*, 7(5):10–15, 2000.

[8] Stephen P Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15. SIAM, 1994.

[9] Nirupama Bulusu, Deborah Estrin, Lewis Girod, and John Heidemann. Scalable coordination for wireless sensor networks: Self-configuring localization systems. In *International Symposium on Communication Theory and Applications (ISCTA 2001), Ambleside, UK*, 2001.

[10] Alberto Cerpa and Deborah Estrin. Ascent: Adaptive self-configuring sensor networks topologies. *IEEE Transactions on Mobile Computing*, 3(3):272–285, 2004.

[11] Anantha Chandrakasan, Rajeevain Amirtharajah, SeongHwan Cho, James Goodman, Gangadhar Konduri, Joanna Kulik, Wendi Rabiner, and Alice Wang. Design considerations for distributed micro-sensor systems. In *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference, San Diego, CA, May 1999*, pages 279–286. IEEE, 1999.

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

[13] Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM, 1999.

[14] Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 642–651. Society for Industrial and Applied Mathematics, 2001.

[15] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 30–39. ACM, 2003.

[16] Cho-Yu Chiang, Ming T Liu, and Mervin E Muller. Caching neighborhood protocol: A foundation for building dynamic web caching hierarchies with proxy servers. In *1999 International Conference on Parallel Processing, 1999. Proceedings*, pages 516–523. IEEE, 1999.

[17] SeongHwan Cho and Anantha P Chandrakasan. Energy efficient protocols for low duty cycle wireless microsensor networks. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01)*, volume 4, pages 2041–2044. IEEE, 2001.

[18] Fabián A Chudak. *Improved Approximation Algorithms for Uncapacitated Facility Location*. Springer, 1998.

[19] Fabián A Chudak and David B Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

[20] Gerard Cornuejols, George L Nemhauser, and Lairemce A Wolsey. The uncapacitated facility location problem. Technical report, DTIC Document, 1983.

[21] Amitabha Das, Hung Keng Pung, Francis Bu Sung Lee, and Lawrence Wong Wai Choong. *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet: 7th International IFIP-TC6 Networking Conference Singapore, May 5-9, 2008, Proceedings*, volume 4982. Springer, 2008.

[22] Michel X Goemans and David P Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation Algorithms for NP-hard Problems*, pages 144–191, 1997.

[23] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

[24] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Hierarchical placement and network design problems. In *41st Annual Symposium on Foundations of Computer Science, 2000. Proceedings*, pages 603–612. IEEE, 2000.

[25] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Improved combinatorial algorithms for single sink edge installation problems. Technical report, Technical Report STAN-CS-TN00-96, Stanford University, 2000.

[26] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174–185. ACM, 1999.

[27] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 56–67. ACM, 2000.

[28] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[29] Chaiporn Jaikaeo, Chavalit Srisathapornphat, and Chien-Chung Shen. Diagnosis of sensor networks. In *IEEE International Conference on Communications, 2001. ICC 2001*, volume 5, pages 1627–1632. IEEE, 2001.

[30] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.

[31] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 731–740. ACM, 2002.

[32] Kamal Jain and Vijay V Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *40th Annual Symposium on Foundations of Computer Science, 1999*, pages 2–13. IEEE, 1999.

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

[34] Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. On the placement of internet instrumentation. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 295–304. IEEE, 2000.

[35] Joseph M Kahn, Randy H Katz, and Kristofer SJ Pister. Next century challenges: Mobile networking for smart dust. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 271–278. ACM, 1999.

[36] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. ii: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[37] Mohamed A Khamsi and William A Kirk. *An Introduction to Metric Spaces and Fixed Point Theory*, volume 53. John Wiley & Sons, 2011.

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

[39] Alfred A Kuehn and Michael J Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

[40] Bo Li, Mordecai J Golin, Giuseppe F Italiano, Xin Deng, and Kazem Sohraby. On the optimal placement of web proxies in the internet. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1282–1290. IEEE, 1999.

[41] Jyh-Han Lin and Jeffrey Scott Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

[42] Samuel Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[43] Olvi L Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 1(2):183–201, 1997.

[44] Norbert Noury, Thierry Hervé, Vicent Rialle, Gilles Virone, Eric Mercier, Gilles Morey, Aldo Moro, and Thierry Porcheron. Monitoring behavior in home using a smart fall sensor and position sensors. In *Conference on Microtechnologies in Medicine and Biology, 1st Annual International. 2000*, pages 607–610. IEEE, 2000.

[45] Emile M Petriu, Nicolas D Georganas, Dorina C Petriu, Dimitrios Makrakis, and Voicu Z Groza. Sensor-based information appliances. *Instrumentation & Measurement Magazine, IEEE*, 3(4):31–35, 2000.

[46] Lili Qiu, Venkata N Padmanabhan, and Geoffrey M Voelker. On the placement of web server replicas. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1587–1596. IEEE, 2001.

[47] Jan M Rabaey, M Josie Ammer, Julio L da Silva Jr, Danny Patel, and Shad Roundy. Picoradio supports ad hoc ultra-low power wireless networking. *Computer*, 33(7):42–48, 2000.

[48] Rahul C Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2):215–233, 2003.

[49] Bo Sheng, Qun Li, and Weizhen Mao. Data storage placement in sensor networks. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 344–355. ACM, 2006.

[50] David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pages 265–274. ACM, 1997.

[51] Sasha Slijepcevic and Miodrag Potkonjak. Power efficient organization of wireless sensor networks. In *IEEE International Conference on Communications, 2001. ICC 2001*, volume 2, pages 472–476. IEEE, 2001.

[52] Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2):405–432, 2005.

[53] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer SJ Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[54] Duane Wessels. Configuring hierarchical squid caches. *National Laboratory for Advanced Network Research*, 1997.

[55] Herbert S Wilf. *Algorithms and Complexity*, volume 986.