Server Placement in the Presence of Competition

Pangfeng Liu[1], Yi-Min Chung[1], Jan-Jan Wu[2], and Chien-Min Wang[2]

[1] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.
[2] Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C.

Abstract. This paper addresses the optimization problems of placing servers in the presence of competition. We place a set of *extra servers* on a graph to compete with the set of *original servers*. Our objective is to find the placement that maximizes the benefit, which is defined as the profits from the requests made to the extra servers despite the competition, minus the cost of constructing those extra servers.

We propose an $O(|V|^3 k)$ time dynamic programming algorithm to find the optimal placement of k extra servers that maximizes the benefit in a tree with $|V|$ nodes. We also propose an $O(|V|^3)$ time dynamic programming algorithm for finding the optimal placement of extra servers that maximizes the benefit, without any constraint on the number of extra servers. For general connected graphs, we prove that the optimization problems are NP-complete. As a result, we present a greedy heuristic for the problems. Experiment results indicate that the greedy heuristic achieves good results, even when compared with the upper bounds found by a linear programming algorithm. The greedy heuristic yields performances within 15% of the upper bound in the worst case, and within 2% of the same theoretical upper bound on average.

1 Introduction

This paper considers a strategy for setting up servers to compete with existing ones. For example, we assume that there are originally a number of McDonald's restaurants in a city, but no Kentucky Fried Chicken (KFC) restaurants. Now, if we decide to set up a number of KFC restaurants in the same city, where should we place them? We need to determine the locations for KFC so that they can compete with McDonald's and maximize their profits. Due to heavy competition among business of similar nature, it is important to choose locations of new servers in the area where the competitors have deployed their servers.

We define the servers we would like to set up as *extra* servers, and the existing (competitor) servers as *original* servers. Thus, in the above example, KFC restaurants are the extra servers and McDonald's restaurants are the original servers.

We use a graph to model the locations of the servers and users. A node in the graph represents a geographic location, and an edge represents a path between two locations. Building servers in these locations enables users at a node to request services from the servers. Each edge has a communication cost. The distance between two nodes is the length of the shortest path that connects them.

For efficiency, We assume that requests from users always go to the nearest server. However, when the shortest distances from a user to the original and extra servers are the same, the user will go to the original server. That is, a user will NORMALLY go to the nearest restaurant, either McDonald's or KFC; however, if the distances to the two restaurants are the same, the user will go to McDonald's.

After extra servers have been established, users who previously went to McDonald's may now go to KFC. We define the benefit of an extra server placement to be the profit derived from user requests made to the server, minus the cost of constructing the server. The cost may vary, depending on the location of the extra server. This paper considers two placement problems related to extra servers, in the presence of competition from original servers.

1. Given the city configuration and a number k, locate k extra servers such that they will earn the most profit;
2. Given the city configuration, locate extra servers such that they earn the most profit, without any constraint on the number of extra servers.

We solve these two problems for a tree graph in $O(|V|^3k)$ and $O(|V|^3)$ time, respectively. For a general graph, we show that the two problems are intractable (NP-complete) and propose a heuristic to solve them. We also run experiments and compare our results for the heuristic with theoretical upper bounds.

Similar server placement problems, such as replica placement problems [4,3,6,10], p-Medians [5], and facility location problems [8], have been studied in the literature. For example, Kariv and Hakimi [5] formulate the p-median problem as locating p points such that the sum of each node's weight multiplied by its shortest distance to the p points is minimized. However, the p-median problem they considered does not take the building costs into account, and it minimizes the costs, instead of maximizing the profits. The facility location problem is similar to the p-median problem, with the additional consideration of the facility's costs.

Our extra server model differs from the model in [5] because it introduces the concept of competition. Extra servers must compete with original servers for user requests, in order to maximize their profits. The number of extra servers established is controlled by the building costs, which differ from location to location. Our dynamic programming model uses a similar technique to that in [4]. The presence of competition demands innovative proof techniques.

Tamir [9] described a dynamic programming model that solves p-median problems on a tree topology with building and access costs. The algorithm assumes that the cost for a client to request services is an increasing function of the distance between the client and the server. If the benefit function in our model is a decreasing function of the distance between the client and the server, our placement problem can be solved by transforming it into a p-median problem, and solving it by the dynamic programming described in [9]. However, the method proposed in this paper can deal with any arbitrary benefit functions, and still obtain the optimal solution for a tree topology.

The remainder of this paper is organized as follows. Section 2 formally describes our server placement models. In Section 3, we introduce the dynamic programming for finding the optimal extra server placement in a tree. Section 4 contains the proof that the problems are NP-complete for general graphs and presents a heuristic algorithm to solve them. Section 5 reports the experiment results, and Section 6 contains our conclusions.

2 Problem Formulation

We consider a connected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each edge $(u, v) \in E$ has a positive integer distance denoted by $d(u, v)$. For any two nodes $u, v \in V$, $d(u, v)$ also denotes the distance of the shortest path between them. For ease of representation, we also let $d(v, S) = \min_{u \in S} d(v, u)$ be the length of the shortest path from v to any node in $\mathcal{X}$, where $\mathcal{X} \subseteq \mathcal{V}$.

We consider *servers* that provide service to nodes in the graph. Every node v must go to the *nearest* server u for service. If a server is located at node v, then v will be serviced by that server. To simplify the concept of "the nearest server", we assuem that for every node v, its distances to all other nodes are different, i.e., $d(v, u) \neq d(v, w)$ for $u \neq w$. As a result the nearest server for every node is *uniquely* defined.

By serving a client v, a server node u earns a benefit of $b(v, u)$. Note that the function b can be arbitrary. For example, unlike [9], we do *not* assume that, for the same client node v, the function value must be monotonic with respect to the distance between v and the server node u.

We assume that there are a number of original servers $\mathcal{O} \subseteq V$ in G. In addition to the original server set $\mathcal{O}$, and we would like to add a number of extra servers to G to obtain the maximum benefit. Let $c(v)$ be the cost of building a server at node $v \in V$, and $\mathcal{X}$ be the set of new servers we would like to add into the system. A node $v \in V$ goes to either $\mathcal{O}$ or $\mathcal{X}$ for service - v goes to $\mathcal{X}$ for service when $d(v, \mathcal{X}) < d(v, \mathcal{O})$; otherwise $(d(v, \mathcal{X}) > d(v, \mathcal{O}))$, v goes to $\mathcal{O}$ for service. Let $V_{\mathcal{X}}$ denote the set of nodes that go to $\mathcal{X}$ for service, and $V_{\mathcal{O}} = V - V_{\mathcal{X}}$ be the set of nodes that go to $\mathcal{O}$ for service.

We define the *nearest servers* $NS(v)$ of v as the server v uses. Consequently $NS(v) \in \mathcal{O}$ if $v \in V_{\mathcal{O}}$, and $NS(v) \in \mathcal{X}$ if $v \in V_{\mathcal{X}}$. We can now define the *benefit function* of adding the servers $\mathcal{X}$ as follows.

$$B(\mathcal{X}) = \sum_{v \in V_{\mathcal{X}}} b(v, NS(v)) - \sum_{v \in \mathcal{X}} c(v). \tag{1}$$

We now define the problem as follows.

k-Extra-Server Problem. Given an integer k, $1 \leq k \leq |V - X|$, we want to find the optimal placement of k extra servers such that the benefit function is maximized (Equation (2)).

$$\max_{\mathcal{X} \subseteq (V - X), |\mathcal{X}| = k} B(\mathcal{X}) \tag{2}$$

Extra-Server Problem. We want to place extra servers to maximize the benefit function, without any constraint on the number of the extra servers. We call this optimization problem the **extra-server problem**.

3 Finding Extra Server Locations

We present algorithms that utilize global information to solve server placement problems. The use of global information facilitates the optimality of the algorithm and the assumption of global information is reasonable since we are dealing with a city or grid configuration and the location of servers are static and can be known completely in advance.

We focus on the case where the graph $G = (V, E)$ is a tree. Let T be the tree and r be the root of T. For each node $v \in V$, let T_v be the subtree of T rooted at v. If v is an internal node, then we use $child(v) = \{v_1, v_2, \ldots, v_{|child(v)|}\}$ to denote the children of v. Following the notations in [4], let $T_v^{(i)}$ be the subtree of T that consists of v and the subtrees rooted at the first i children of v, i.e., $T_v^{(i)} = \{v\} \cup \cup_{j=1}^{i} T_{v_j}$.

Definition 1 (Benefit function, B). *For nodes $v, u \in V$, an integer k, and an integer i between 0 and $|child(v)|$, we define $B_{k,i}^{v,u}$ to be the maximum benefit derived by placing k extra servers in $T_v^{(i)}$, under the condition that $u = NS(v)$. Consequently u is either an original server or an extra server.*

We now consider the benefit function $B_{k,i}^{v,u}$ by placing $\mathcal{X}$ in $T_v^{(i)}$. We define $\mathcal{X}$ to be the set of k extra servers that maximize the following benefit function. Recall that $\mathcal{O}$ is the set of original servers.

$$B_{k,i}^{v,u} = \max_{\mathcal{X}}\{ \sum_{w \in T_v^{(i)}, NS(w) \in \mathcal{X} \cup u} b(w, NS(w)) - \sum_{s \in \mathcal{X}} c(s)\},\ u \notin \mathcal{O},$$

$$B_{k,i}^{v,u} = \max_{\mathcal{X}}\{ \sum_{w \in T_v^{(i)}, NS(w) \in \mathcal{X}} b(w, NS(w)) - \sum_{s \in \mathcal{X}} c(s)\},\ u \in \mathcal{O}.$$

The definition indicates that the benefit includes those nodes that will either go to the extra servers $\mathcal{X}$ or u (when $u \notin \mathcal{O}$) for service, minus the construction cost of the extra server set $\mathcal{X}$.

For the case where u is not in $\mathcal{O}$, by definition u is v's nearest server, so u has an extra server. However, u can be a node outside of $T_v^{(i)}$, – in which case it will not be in $\mathcal{X}$ because $\mathcal{X}$ is a subset of $T_v^{(i)}$. We still need to add the benefit from $T_v^{(i)}$ to u, since we assume that an extra server is placed in u.

Lemma 1. *For every node $v \in V$ and every child v_i of v, if $u \in T_{v_i}$ is the nearest server to v, then u is also the nearest server to v_i.*

Proof. We prove this lemma by contradictions and assume that the nearest server for v_i is u', not u. Since u' is the nearest server to v_i, the distance $d(v_i, u')$ must be *strictly* smaller than $d(v_i, u)$. The length of the shortest path between v and u' is $d(v, u') \leq d(v, v_i) + d(v_i, u') < d(v, v_i) + d(v_i, u) = d(v, u)$, which suggests that u' is closer to v than u; however, this contradicts the assumption that u is the nearest server of v. ■

For ease of discussion of the following lemma, we define a node set $V_{v,u,i}$. This set contains those nodes in T_{v_i} that could be the nearest server for v_i, under the condition that u is the nearest server for v, but not for v_i, i.e., $NS(v) = u$ and $NS(v_i) \neq u$. Intuitively, the set $V_{v,u,i}$ stands for those nodes in T_{v_i} that are far enough from v so that it will not be the nearest server for v (when compared with u), but close enough to v_i so that it is the nearest server of v_i.

Definition 2 ($\mathbf{V_{v,u,i}}$). *Let u be the nearest server of v and i be an integer between 1 and $|child(v)|$. $V_{v,u,i}$ is the subset of those u' in T_{v_i} such that u' is the nearest server to v_i, but it is* not *the nearest server to v. That is, $V_{v,u,i} = \{u' | u' \in T_{v_i}, d(v_i, u') < d(v_i, u),\ d(v, u)d(v, u')\}$*

Lemma 2. *For every node $v \in V$ and every child v_i of v, if $u \notin T_{v_i}$ is the nearest server of v, then either u is the nearest server of v_i or there exists a node $u' \in V_{v,u,i}$ that is the nearest server of v_i.*

Proof. If u *is* the nearest server of v_i, the lemma follows. Otherwise, we conclude that the nearest server of v_i must be within T_{v_i}, since the path from v_i to nodes not in T_{v_i} must pass through v, which already has u as its nearest server. The lemma then follows by the definition of $V_{v,u,i}$. ■

Theorem 1. *For every node $v \in V$ and an integer i between 0 and $|child(v)|$, if u is the nearest server of v, then for every node w in T_{v_i}, we can find the nearest server for w in $T_{v_i} \cup \{u\}$.*

Proof. The only way a shortest path from a node w in T_{v_i} to any node outside T_{v_i} is to go through the edge (v_i, v). However, any such shortest path must end at node u since u is the nearest server for v; otherwise we will be able to find a closer server for v other than u – a contradiction to the fact that $NS(v) = u$. ■

Terminal Conditions. We first derive two terminal conditions for the recursion of B, the benefit function.

$\mathbf{k = 0}$. When k is 0, we do not place any extra servers in $T_v^{(i)}$. If u is an original server in $\mathcal{O}$, every node in $T_v^{(i)}$ will go to $\mathcal{O}$ for service, so the benefit is 0. If u is not in $\mathcal{O}$, we consider two cases. First if u is not in $T_v^{(i)}$, every node in $T_v^{(i)}$ will either go to an original server or to u for service; thus, the benefit can be determined by Equation (3).

$$B' = \sum_{w \in T_v^{(i)}, d(w,u) < d(w,\mathcal{O})} b(w, u) \tag{3}$$

In the second case, u is not an original server but u is in $T_v^{(i)}$, which means that there is at least one extra server in $T_v^{(i)}$. This contradicts the assumption that k is 0. For the purpose of dynamic programming, we define the benefit to be $-\infty$.

$\mathbf{k = 1, u \notin \mathcal{O}, u \in T_v^{(i)}}$**.** When k is 1, u is in $T_v^{(i)}$, so it is not an original server, but it is definitely the only extra server in $T_v^{(i)}$. Every node in $T_v^{(i)}$ will either go to $\mathcal{O}$ or u for service; thus, the benefit can be calculated in the same way as $B' - c(u)$. Note that, since u is now in the $\mathcal{X}$ that maximizes the benefit of $T_v^{(i)}$, $c(u)$ should be deducted from the benefit.

Recursion. Next, we derive the recursion function for $B_{k,i}^{v,u}$.

$$B_{k,i}^{v,u} = \begin{cases} 0, & \text{if } k = 0 \text{ and } u \in \mathcal{O} \\ B', & \text{if } k = 0,\ u \notin \mathcal{O}, \text{ and } u \notin T_v^{(i)} \\ B' - c(u), & \text{if } k = 1,\ u \notin \mathcal{O}, \text{ and } u \in T_v^{(i)} \\ B'', & \text{if } u \in T_{v_i} \\ \max\{B'', B'''\}, & \text{if } u \notin T_{v_i} \\ -\infty, & \text{otherwise,} \end{cases} \tag{4}$$

where

$$B'' = \max_{0 \le j \le k} \left\{ B_{k-j,i-1}^{v,u} + B_{j,|child(v_i)|}^{v_i,u} \right\}, \tag{5}$$

and

$$B''' = \max_{0 \le j \le k} \left\{ B_{k-j,i-1}^{v,u} + E_{j,i}^{v,u} \right\}. \tag{6}$$

The first three cases were discussed as the terminal conditions in Section 3, so we only need to consider the rest.

$\mathbf{u \in T_{v_i}}$

If $u \in T_{v_i}$, u will also be the nearest server to v_i by Lemma 1, since u is the nearest server of v. Then, by Theorem 1, every node in T_{v_i} goes to either T_{v_i} or u for service. In addition, u is the nearest server to v. By Theorem 1, all nodes in $T_v^{(i-1)}$ obtain service from u or $T_v^{(i-1)}$.

Assume that there are j extra servers in T_{v_i}, then there will be $k-j$ extra servers in $T_v^{(i-1)}$, where $0 \le j \le k$. To obtain the best $\mathcal{X}$ that maximizes the benefit, we need to consider all possible values of j, as formulated in Equation (5). The recursion follows.

$\mathbf{u \notin T_{v_i}}$

If u is not in T_{v_i}, we need to consider two sub-cases.

Case 1: If u is the nearest server of v_i, the value of $B_{k,i}^{v,u}$ is defined as in Equation (5), because we can isolate two subtrees, as we did in the previous case where $u \in T_{v_i}$.

Case 2: If the nearest server of v_i is *not* u, by Lemma 2, we can find the nearest server u' for v_i in T_{v_i}. We formulate the benefit as B''' in Equation (6).

Consider these two sub-cases, if $u \notin T_{v_i}$, $B_{k,i}^{v,u}$ is formulated as $\max\{B'', B'''\}$.

Now, in order to finish the recursion the only missing element is the new cost function $E_{k,i}^{v,u}$.

Definition 3 ($E_{k,i}^{v,u}$). *For nodes $v, u \in V$, an integer k, and the i-th child of node v (denoted by v_i), we define $E_{k,i}^{v,u}$ to be the maximum benefit derived by placing k extra servers in the subtree T_{v_i}, where $u \notin T_{v_i}$ is the nearest server of v, but u is not the nearest server of v_i. Instead, the nearest server of v_i is a u' in T_{v_i}. The benefit is similarly defined in Equation (7):*

$$E_{k,i}^{v,u} = \max_{\mathcal{X}}\{ \sum_{w \in T_{v_i}, NS(w) \in \mathcal{X}} b(w, NS(w)) - \sum_{s \in \mathcal{X}} c(s)\}. \tag{7}$$

From the above discussion, the maximum benefit $E_{k,i}^{v,u}$ is derived by Equation (8). That is, we need to enumerate all the possible u' and use the one that maximizes $B_{k,|child(v_i)|}^{v_i,u'}$. The set $V_{v,u,i}$ is exactly the possible set to select u' from, since v_i will go to u' for service, but not to u. This is exactly the definition of $V_{v,u,i}$.

$$E_{k,i}^{v,u} = \max_{u' \in V_{v,u,i}} \left\{ B_{k,|child(v_i)|}^{v_i,u'} \right\}. \tag{8}$$

The Final Solution. Finally, the maximum benefit of locating k extra servers in the tree T can be calculated by Equation (9):

$$\max_{u \in T} \left\{ B_{k,|child(r)|}^{r,u} \right\}. \tag{9}$$

The possible candidates for u are subject to the following constraints: If u is an original server $d(r, u)$ must be $d(r, \mathcal{O})$, i.e., u is the nearest original server to the root. If u is not an original server, the distance $d(r, u)$ must be smaller than $d(r, \mathcal{O})$ to ensure that u is the nearest extra server to the root.

Theorem 2. *Given a tree $T = (V, E)$ and a set $\mathcal{O} \subseteq V$ as the original servers, the k-extra-server problem for T can be solved in $O(|V|^3 k)$ time, where $0 \leq k \leq |V - \mathcal{O}|$ is an integer.*

Proof. The problem can be solved by Equations (3) to (9). The time of the dynamic programming is derived by calculating all the entries of $B_{k,i}^{v,u}$ and $E_{k,i}^{v,u}$. Consider each pair of v and i, so that there are totally $\sum_{v \in V} |child(v)| = |V| - 1$ pairs. Thus, the number of entries of $B_{k,i}^{v,u}$ is $(k+1) \cdot |V| \cdot (|V| - 1) = O(|V|^2 k)$, and it takes $O(|V|)$ time to calculate each entry; hence, the time required to calculate all the entries of $B_{k,i}^{v,u}$ is bounded by $O(|V|^3 k)$. Similarly, there are $O(|V|^2 k)$ entries of $E_{k,i}^{v,u}$, and it takes $O(|V|)$ time to calculate each entry; therefore, the time required to calculate all the entries of $E_{k,i}^{v,u}$ is $O(|V|^3 k)$. The total time required is therefore $O(|V|^3 k)$. ■

Using similar techniques we derive the following theorem. The proof is removed due to space limitation.

Theorem 3. *Given a tree graph $T = (V, E)$ and $\mathcal{O} \subseteq V$ are the original servers of T, the extra-server problem for T can be solved in $O(|V|^3)$ time.*

Proof. The proof is similar to that of Theorem 2. There are $O(|V|^2)$ entries of $B_i^{v,u}$ and $O(|V|^2)$ entries of $E_i^{v,u}$, and the calculation of each entry requires at most $O(|V|)$ computing time. Hence, the problem can be solved in $O(|V|^3)$ time. ■

4 NP-Completeness

The NP-complete proof is derived from the *dominating set* problem [2], and is removed due to space limitation. A subset $V' \subseteq V$ is a dominating set if for all $u \in V - V'$, there is a $v \in V'$ such that the edge (u, v) is in E. The decision problem of the *dominating set* can be formulated as follows: Given a graph $G = (V, E)$ and a positive integer $K \leq |V|$, is there a dominating set of size K or less?

k-EXTRA-SERVER. We now consider the k-extra-server problem and define the corresponding decision problem as follows: In a k-extra-server problem instance, is there a placement of k extra servers such that the benefit is at least B?

EXTRA-SERVER. Similarly, we define the decision problem of EXTRA-SERVER as follows: In a extra-server problem instance, is there a placement of extra servers such that the benefit is at least B?

Theorem 4. *The k-EXTRA-SERVER problem is NP-complete.*

Theorem 5. *The EXTRA-SERVER problem is NP-complete.*

Since the k-extra-server problem and the extra-server problem are both NP-complete, we propose a greedy heuristic (denoted as **Greedy**) for these problems. Here, we only describe **Greedy** for the k-extra server problem because the method for the extra-server problem is very similar.

The greedy method works in rounds. In each round, we locate an extra server that maximizes its benefit. We add the benefit produced by the selected extra server to the total benefit, which was set to 0 initially, and then mark the selected server as an *original* server. We repeat the process until k extra servers are selected.

5 Experiment Results

We conduct simulations to compare performance of **Greedy** with the linear programming optimal solutions acquired using GLPK (*GNU Linear Programming Kit*) [7] for the *k-extra-server* problem. GLPK is a set of routines designed to solve large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is written in ANSI C and organized in the form of a library [7]. Let the 0-1 variable X_u and $u \in V$ denote whether there is an extra server on u, and let the 0-1 variable Z_{uv}, $u \in V$, $v \in V$ denote whether v is a client of u. The integer programming for the k-extra-server problem is formulated as follows:

$$\text{maximize} \quad \sum_{u \in (V-X)} \sum_{v \in V} Z_{uv} b(v,u) - \sum_{u \in V} X_u c(u), \tag{10}$$

subject to

$$\begin{aligned}
X_u &\in \{0,1\}, && \text{for each } u \in V, && \text{(11a)}\\
Z_{uv} &\in \{0,1\}, && \text{for each } u \in V,\ v \in V, && \text{(11b)}\\
X_u &= 0, && \text{for each } u \in \mathcal{O}, && \text{(11c)}\\
\sum_{u \in V} X_u &= k, && && \text{(11d)}\\
\sum_{u \in V} Z_{uv} &= 1, && \text{for each } v \in V, && \text{(11e)}\\
X_u - Z_{uv} &\geq 0, && \text{for each } u \in (V - \mathcal{O}), \text{ each } v \in V, && \text{(11f)}\\
Z_{uv} &= 0, && \text{for each } u \in V, \text{ each } v \in V, \text{ and } d(v,u) > d(v,\mathcal{O}). && \text{(11g)}
\end{aligned}$$

Consider the 0-1 variables X_u and Z_{uv} in constraints (11a) and (11b) respectively. We replace them with constraints (12a) and (12b) respectively, so that we have a linear programming formulation.

$$\begin{aligned}
0 \leq X_u &\leq 1, && \text{for each } u \in V, && \text{(12a)}\\
0 \leq Z_{uv} &\leq 1, && \text{for each } u \in V,\ v \in V. && \text{(12b)}
\end{aligned}$$

The optimal benefit gained from linear programming only serves as a upper bound, since it allows a fraction number of an extra server to be placed on a node. However, in our experiments, we find that, in most cases, linear programming produces integer solutions, i.e., X_u and Z_{uv} are in the range $\{0,1\}$.

5.1 Experiment Setting

In our experiments, we use GT-ITM [1] to generate random graphs according to Waxman model [11]. Each of the graphs is connected, and nodes are added randomly in a $s \times s$ square. The probability of an edge between u and v is given by

$$p(u,v) = \alpha e^{-d/\beta L},$$

where $0 < \alpha, \beta \leq 1$, d is the Euclidean distance between u and v, and $L = \sqrt{2}s$ is the largest possible distance between any two nodes. In our experiments, we set s to 20, α to 0.2 and β to 1.

For each v, we set a value $r(v)$ to be a random integer between 20 and 40, and set the building cost $c(v)$ to be $r(v)$ plus a random integer between 1 and 10. The benefit function $b(v,u)$ is defined as $r(v)$ divided by the distance from v to u. Finally, we place original servers randomly in the graph. We simulate up to 150 nodes since this is a reasonable size for city or grid configuration.

5.2 Effect of α

We evaluate the performance of **Greedy** compared with the upper bounds found by linear programming under different values of α. In these experiments, for each α we set $|V|$ from 50 to 150, and for each $|V|$ we set $|\mathcal{O}|$ from 0 to $0.1|V|$. As a result, we have 1066 graphs to simulate, and for each graph we set k from 1 to $0.1|V|$. Figure 1 shows that when α increases the average degree of each node also increases. Figure 1 shows that **Greedy** performs very well; on average, its performance differs from the theoretical upper bounds by only 1% and in the worst case the difference is no more than 15% of the upper bound.

Figure 1 also shows that as α increases, the average difference between **Greedy** and the upper bound derived by linear programming also increases. Since the average degree of each node increases as α increases, there is a higher probability that the extra servers will affect each other. However, to maximize the benefit, **Greedy** only considers the current configuration when it selects the next location to place an extra server; thus, it can not predict the "long range" effects and the interaction among the extra servers. Hence, as α increases, the average difference (as a percentage) between **Greedy** and the upper bound also increases.

$\lvert V\rvert$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
50	3.56	5.37	7.11	8.78
150	10.45	15.59	20.87	26.12
Average	8.11	12.01	16.03	20.03

α	Avg. difference	Max. difference
0.2	0.43%	9.54%
0.3	0.49%	14.35%
0.4	0.52%	13.20%
0.5	0.58%	11.95%

Fig. 1. The average degree of a node under different values of α and the average difference (as a percentage) between **Greedy** and the upper bound under different values of α

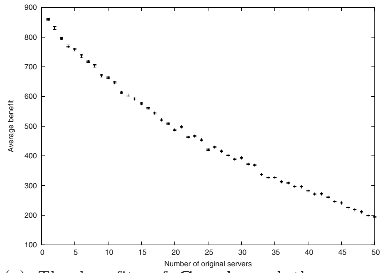
5.3 Effect of the Number of Original Servers

We now consider the effect of the number of original servers on the average difference as a percentage of the upper bounds. In these experiments we set $|V|$ to 100, $|\mathcal{O}|$ from 1 to 50, and k to 10.

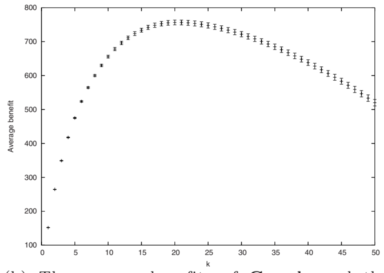
Figure 2 (a) shows the error-bar between **Greedy** and the upper bounds derived by the linear programming. The upper markers are the average upper bounds and the lower markers are the average benefits of **Greedy**. In the figure, the average benefits produced by **Greedy** are so close to the upper bounds that they coincide. Furthermore, the figure suggests that as $|\mathcal{O}|$ increases the benefit will decrease. This is reasonable since a large number of competitors only have a negative impacts on the extra servers.

5.4 Effect of k

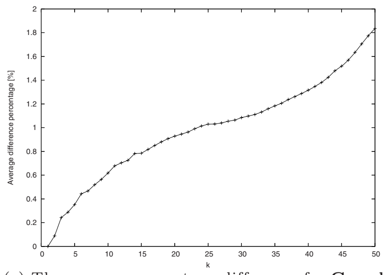
Next, we consider the effects of k on the average difference as a percentage between **Greedy** and the theoretical upper bound. In these experiments we set



(a) The benefits of **Greedy** and the average upper bounds under different numbers of original servers.



(b) The average benefits of **Greedy** and the upper bounds under different values of k.



(c) The average percentage difference for **Greedy** under different values of k.

Fig. 2. Average benefits under different number of original and extra servers ((a) and (b)), and derivation percentage from the theoretical bounds (c)

$|V|$ to 100 and $|\mathcal{O}|$ to 10, so we generate 100 graphs in total. For each graph we set k from 1 to 50, which gives us 5000 simulation results.

Figure 2 (b) shows the error-bars in our simulations. We observe that the benefit of **Greedy** is extremely close to the theoretical upper bounds. The figure also shows that, initially, as k increases, the benefit increases because we can make more profit. As the number of extra servers increases substantially, the benefit decreases due to the cost of constructing the extra servers.

Figure 2(c) shows that as k increases the average difference between **Greedy** and the theoretical upper bound also increases. This is because **Greedy** places an extra server to maximize the benefit at each step because it can not consider the overall situation; thus, the difference accumulates at each step – more servers means a larger difference between **Greedy** and the upper bound.

In summary, we conclude that the **Greedy** algorithm performs extremely well. Considering all the simulation parameter setting, the greedy algorithm yields average benefits that are within 2% of the average theoretical upper bounds. It is also extremely efficient and easy to implement.

6 Conclusion

We have formulated two optimization problems, the k-extra-server problem and the extra-server problem. We consider the profit and construction costs at each location, and place extra servers to maximize the benefit in the presence of competition from original servers. For trees, we formulate dynamic programming algorithms to solve the k-extra-server problem and the extra-server problem in $O(|V|^3k)$ time and $O(|V|^3)$ time, respectively. For general graphs, we prove that the problems are NP-complete and propose a greedy heuristic to solve them. The experiment results demonstrate that the greedy heuristic yields performances within 15% of the theoretical upper bound in the worst case, and within 2% of the same theoretical upper bound on average.

In the future we will investigate the possibility of designing efficient and effective algorithms for graphs other than trees. For example, our greedy algorithms perform well on general graphs, so we should be able to show that the greedy algorithm performance is guaranteed to be within a constant factor of the optimum. We would also like to generalize dynamic programming to other graphs, such as planar graphs.

References

1. K. Calvert and E. Zegura. Gt-itm: Georgia tech internetwork topology models. http://www-static.cc.gatech.edu/projects/gtitm/.
2. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.
3. X. Jia, D. Li, X. Hu, W. Wu, and D. Du. Placement of web-server proxies with consideration of read and update operations on the internet. *The Computer Journal*, 46(4):378–390, 2003.
4. K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, June 2001.
5. O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. ii: The p-medians. *SIAM J. Appl. Math.*, 37(3):539–560, 1979.
6. B.-J. Ko and D. Rubenstein. A greedy approach to replicated content placement using graph coloring. In *SPIE ITCom Conference on Scalability and Traffic Control in IP Networks II*, Boston, MA, July 2002.
7. A. Makhorin. http://www.gnu.org/software/glpk/glpk.html.
8. D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proc. 29th ACM STOC.*, pages 265–274, 1997.
9. A. Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996.
10. O. Unger and I. Cidon. Optimal content location in multicast based overlay networks with content updates. *World Wide Web*, 7(3):315–336, 2004.
11. B. M. Waxman. Routing of multipoint connections. pages 347–352, 1991.