Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Information Processing Letters ●●● (●●●●) ●●●–●●●
Information Processing Letters
www.elsevier.com/locate/ipl

Joint object placement and node dimensioning for Internet content distribution [☆]

Nikolaos Laoutaris [*], Vassilios Zissimopoulos, Ioannis Stavrakakis

Department of Informatics and Telecommunications, University of Athens, 15784 Athens, Greece

Received 31 October 2003; received in revised form 15 December 2003

Communicated by L. Boasson

Abstract

This paper studies a resource allocation problem in a graph, concerning the joint optimization of capacity allocation decisions and object placement decisions, given a single capacity constraint. This problem has applications in Internet content distribution and other domains. The solution to the problem comes through a multi-commodity generalization of the single commodity k-median problem. A two-step algorithm is developed that is capable of solving the multi-commodity case optimally in polynomial time for the case of tree graphs, and approximately (within a constant factor of the optimal) in polynomial time for the case of general graphs.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Combinatorial problems; Content distribution; Multi-commodity k-median; Packing

1. Introduction

This paper studies a joint placement and dimensioning problem in a content distribution network (CDN) for the dissemination of Internet content. A CDN operator has at its disposal a total storage capacity for S unit-sized information objects. The target is to use this storage budget to selectively replicate information objects drawn from a set of available information objects, such that the average distance from all clients to all the requested objects be minimized. Producing the required object placement not only identifies the set of information object for each node but also returns the fraction of the total storage capacity that must be allocated to each node. In [6] we have formulated a specific instance of this problem and proposed heuristic algorithms for it.

The discussed problem will henceforth be called the *capacity allocation problem*, abbreviated CAP. At an abstract level, the goal of CAP is to select appropriate locations for the installment of different types of facilities, under known topological and demand pattern information and the constraint that the maximum number of opened facilities (of all types) does not exceed a given capacity. The name CAP owes much to its application in allocating generic capacity units in a

☆ This work and its dissemination efforts have been supported in part by the IST Programs of the European Union under contract IST-2001-32686 (Broadway) and IST-2001-33116 (FLAGS).

* Corresponding author.
E-mail addresses: laoutaris@di.uoa.gr (N. Laoutaris), vassilis@di.uoa.gr (V. Zissimopoulos), istavrak@di.uoa.gr (I. Stavrakakis).

0020-0190/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.ipl.2003.12.002

S0020-0190(03)00537-4/SCO AID:3014 Vol.●●●(●●●)
ELSGMLTM(IPL):m3 v 1.181 Prn:12/01/2004; 14:33 ipl3014 P.2(1-7)
by:PS p. 2

network, with each capacity unit being able to host a single facility chosen among a set of available types.

Multi-commodity placement problems that have appeared in the literature differ substantially from the stated CAP problem. Korupolu et al. [5] have studied a multi-commodity object placement problem in a tree and given an exact polynomial time algorithm for it. This work differs from our's in that it considers per-node capacity constraints and assumes a specific distance metric called ultra-metric. Under the ultra-metric, the distance between any two nodes equals the diameter of the smallest subtree containing both nodes. This is actually an approximate notion of distance and is not as precise as the usually employed notion of distance (length of a shortest path connecting two nodes). Here we allow for an additional degree of freedom by allowing the algorithm to dimension the nodes along with the placement of objects and assume a general metric space for distance. Notice that a solution to CAP does not stem from a trivial extension of [5], as the latter cannot guarantee a total allocated capacity. More recently Baev and Rajaraman [2] have provided approximation algorithms for multi-commodity placement in general graphs. Their work too considers per-node capacity constraints.

In the sequel the k-median problem is introduced, followed by a formal statement of CAP and the presentation of the two-step algorithm for its solution. A distinctive characteristic of the two-step algorithm is that it depends on the specific properties of the studied graph only through the specific k-median algorithm that it uses as a building block. This allows for a unified treatment of trees and general graphs. Using a polynomial time exact algorithm for the k-median in a tree in conjunction with the two-step algorithm leads to a polynomial time exact algorithm for CAP in a tree. For general graphs, the combination of a polynomial time approximation algorithm for k-median and the two-step algorithm leads also to a polynomial time approximation algorithm for CAP. In view of the fact that CAP is NP-hard for general graphs (it reduces to k-median when considering only one type of facility) the resulting approximation becomes particularly interesting as it has the property of retaining the good properties of the employed k-median algorithm. It is in fact possible to directly construct a constant factor approximation algorithm for CAP by employing a constant factor approximation algorithm for k-median. Arya et al. [1] have given the first polynomial constant factor approximation algorithm for the k-median in a general graph, with an approximation ratio of $3 + 2/p$ (p is a parameter of the local search heuristic employed in the work of Arya et al.). Using this algorithm in conjunction with the two-step algorithm leads to a $3 + 2/p$ approximation for CAP in a general graph. In effect, the two-step algorithm is able to guarantee for CAP an approximation ratio that is as good as the one for the k-median.

2. The k-median problem

Consider an undirected connected graph $G = (V, E)$ with node set $V = \{v_1, \dots, v_n\}$ and edge set E. Each edge is associated to a non-negative weight and the length of a path in G is equal to the sum of the weights of all edges in the path. Let $d(v_i, v_{i'})$ denote the distance between a pair of nodes $(v_i, v_{i'})$, given by the length of a shortest length path connecting v_i and $v_{i'}$. Each node has a demand for r_i units of service from its closest *supply node*. A k-placement, $P^{(k)}$, is a set of no more than k nodes out of the total n that are selected to act as supply nodes, offering service to the client nodes (a node may be both a client and a supply node). Let $\mathbb{P}^{(k)}$ denote the set of all possible k-placements. The k-median problem amounts to selecting an optimal k-placement, $\mathcal{P}^{(k)} \in \mathbb{P}^{(k)}$, that minimize the weighted sum of distances from all nodes (clients) to the their closest supply node, i.e., minimize

$$\phi(P^{(k)}) = \sum_{v_i \in V} r_i \cdot \min_{u \in P^{(k)}} d(v_i, u), \qquad P^{(k)} \in \mathbb{P}^{(k)}. \quad (1)$$

This is equivalent to maximizing

$$\psi(P^{(k)}) = \sum_{v_i \in V} r_i \cdot \Bigl(d_w - \min_{u \in P^{(k)}} d(v_i, u)\Bigr), \quad P^{(k)} \in \mathbb{P}^{(k)}, \quad (2)$$

where d_w is a constant. Kariv and Hakimi [3] have proved that k-median is NP-hard for general graphs. They showed however that for undirected trees the problem can be solved in $O(k^2 n^2)$. More recently the complexity of the undirected tree version has been reduced to $O(kn^2)$ by Tamir [10]. For directed trees the problem has been solved in $O(k^2 P)$ by Vigneron et al. [11], for the specific type of directed trees where all edges are directed upwards towards the root (here P denotes the path length of the tree which is $\mathrm{O}(n^2)$ in the worst case). For the case of k-median in a general graph and metric distance, Lin and Vitter [7] gave a $(2(1+\varepsilon), 1+(1/\varepsilon))$-approximation algorithm and Korupolu et al. [4] gave a $(1+\varepsilon, 3+5/\varepsilon)$ and a $(1+5/\varepsilon, 3+\varepsilon)$-approximation algorithm.[1] More recently Arya et al. [1] gave a $(3+2/p)$ polynomial time approximation algorithm using no more than k supply nodes (p is the maximum swap of facilities allowed by their local search procedure); this appears to be the first constant factor approximation algorithm without blowup in the number of supply nodes.

3. The capacity allocation problem

The capacity allocation problem (CAP) is a generalization of the k-median problem, involving multiple types of service and the requirement that all nodes receive all types of service, either locally, or from the closest supply node for a given type of service. As the number of different services may exceed the total number of allowed medians, an additional node w is introduced that lays outside G and is capable of offering all the types of service. Node w is accessible by all nodes in G at the same distance $d_w = d(v, w),\ v \in V$, that is larger than the diameter of G. For the purpose of this paper it will be assumed that a service is the ability to offer a certain information object (e.g., a web page) that is stored locally but this does not limit the applicability of the model to other circumstances as well. There exist N distinct unit-sized objects comprising the object set $O = \{o_1, \ldots, o_N\}$, o_j denoting the jth object. The demand of each node v_i is described by a rate vector over O, $r_i = \{r_{i1}, \ldots, r_{iN}\}$, r_{ij} denoting the rate at which node v_i requests object o_j. The origin node w is assumed to be holding the entire object set O. A (j,k)-placement, $P_j^{(k)}$, is a set of no more than k nodes out of the total n that are selected as supply nodes for object o_j. Let $\mathbb{P}_j^{(k)}$ denote the set of all possible (j,k)-placements. An S-placement, $P^{(S)} = \{P_{j_1}^{(k_{j_1})}, \ldots, P_{j_R}^{(k_{j_R})}\}$, is a set of (j_h, k_{j_h})-placements, $P_{j_h}^{(k_{j_h})}$, $1 \leqslant h \leqslant R \leqslant S$, with R members, having the property:

$$\sum_{h=1}^{R} k_{j_h} \leqslant S, \quad k_{j_h} > 0, \forall h.$$

A node-object pair (u, o_j) is said to belong to $P^{(S)}$, $(u, o_j) \in P^{(S)}$, if and only if there exists h: $1 \leqslant h \leqslant R$, $j_h = j$, $u \in P_{j_h}^{(k_{j_h})}$. Let $\mathbb{P}^{(S)}$ denote the set of all possible S-placements. CAP amounts to identifying an optimal S-placement, $\mathcal{P}^{(S)} \in \mathbb{P}^{(S)}$, that minimizes the weighted sum of distances from all nodes to their closest supply node, for all objects in O, i.e., minimize

$$f(P^{(S)}) = \sum_{v_i \in V} \sum_{o_j \in O} r_{ij} \cdot \min_{(u,o_j) \in (P^{(S)} \cup w)} d(v_i, u),$$
$$P^{(S)} \in \mathbb{P}^{(S)}, \tag{3}$$

$\mathcal{P}^{(S)}$ has the property of maximizing the following expression.

$$\begin{aligned} g(P^{(S)}) &= \sum_{v_i \in V} \sum_{o_j \in O} r_{ij} \cdot \Bigl(d_w - \min_{(u,o_j) \in (P^{(S)} \cup w)} d(v_i, u)\Bigr) \\ &= \sum_{v_i \in V} \sum_{\substack{o_j \in O: \\ \exists u \in G: (u,o_j) \in P^{(S)}}} r_{ij} \\ &\quad \times \Bigl(d_w - \min_{(u,o_j) \in P^{(S)}} d(v_i, u)\Bigr), \\ P^{(S)} &\in \mathbb{P}^{(S)}, \end{aligned} \tag{4}$$

$\min\{f(P^{(S)})\}$ is equivalent to $\max\{g(P^{(S)})\}$ because $g(P^{(S)}) = (d_w \cdot \sum_{i=1}^{n} \sum_{j=1}^{N} r_{ij}) - f(P^{(S)})$. The transition from the first to the second line of (4) is because $(d_w - \min_{(u,o_j) \in (P^{(S)} \cup w)} d(v_i, u)) = 0$ when $\min_{(u,o_j) \in (P^{(S)} \cup w)} d(v_i, u) = d_w$, i.e., for all objects o_j that do not have a replica in G (i.e., for objects o_j that there does not exist a $u \in G$ such that $(u, o_j) \in P^{(S)}$). Using the function,

$$\psi_j\bigl(P_j^{(k)}\bigr) = \sum_{v_i \in V} r_{ij} \cdot \Bigl(d_w - \min_{u \in P_j^{(k)}} d(v_i, u)\Bigr),$$
$$P_j^{(k)} \in \mathbb{P}_j^{(k)} \tag{5}$$

and the definition of $P^{(S)}$, $g(P^{(S)})$ may be re-written as:

$$g(P^{(S)}) = \sum_{h=1}^{R} \psi_{j_h}\bigl(P_{j_h}^{(k_{j_h})}\bigr). \tag{6}$$

[1] A (a, b)-approximation algorithm is a polynomial time algorithm that computes a solution using at most $b \cdot k$ supply nodes and with distance at most a times worse than the distance under the optimal algorithm using k supply nodes.

S0020-0190(03)00537-4/SCO AID:3014 Vol.●●●(●●●)
ELSGMLTM(IPL):m3 v 1.181 Prn:12/01/2004; 14:33 ipl3014 P.4 (1-7)
by:PS p. 4

Notice that the $\psi_j()$ function of (5) is a k-median objective function for object o_j; its sole difference to the $\psi()$ function of (2) is that r_i is replaced by r_{ij}. In effect $\psi_j(P_j^{(k)})$ captures the total gain in terms of reduced access distance achieved by placing k replicas of object o_j in G thus not having to fetch o_j from the origin node but rather from a node in G. The function $g(P^{(S)})$ sums all the gains for the R individual objects o_{j_h}, $1 \leqslant h \leqslant R$ that get to have at least one replica in the context of an S-placement $P^{(S)}$.

4. An exact algorithm for CAP

This section describes an exact solution algorithm for CAP. The algorithm involves two steps: (1) a *decomposition step* involving the solution of a series of k-median problems relating to the original CAP problem; (2) a *composition step* involving the selection of a number of optimal solutions from the first step, that when combined together yield the exact optimal solution for the original CAP. The decomposition step solves a sufficient number of single commodity k-median *location problems*. The composition step is a *packing problem* that selects among the optimally solved k-medians to construct an optimal S-placement.

4.1. Decomposition step: k-medians

In this step $N \cdot S'$ k-medians, where $S' = \min\{n, S\}$, are solved and their solutions stored for use in the second step. Specifically for each object $o_j \in O$ a series of S' k-medians, $1 \leqslant k \leqslant S'$, is solved using an appropriate algorithm for the given type of graph G. In fact the largest possible median of the o_j series, i.e., the one with $k = n$, occurring only when $S \geqslant n$, does not have to be solved because its solution is trivial—replicate one copy of object o_j in each of the n nodes. Let G_{jk} denote the *gain* under $\mathcal{P}_j^{(k)}$, the optimal placement of k replicas of object o_j in G. G_{jk} is defined to be the difference between the maximum sum of distances incurred when all nodes fetch o_j from w minus the minimum sum of distances incurred when k replicas of o_j are installed in G optimally, thus:

$$G_{jk} = \max_{P_j^{(k)} \in \mathbb{P}_j^{(k)}} \psi_j\bigl(P_j^{(k)}\bigr). \tag{7}$$

The G_{jk} values are the input of the subsequent step that decides how many replicas from each object to include in the final S-placement.

The complexity of the decomposition step depends on the properties of the studied graph. Let $F_G(k,n)$ denote the asymptotic complexity of solving the k-median problem in a graph G with n nodes, using the best algorithm for the specific type of G. Then solving $S' \leqslant n-1$ k-medians for each object o_j, $1 \leqslant j \leqslant N$, independently, would lead to an overall complexity $\mathrm{O}(nN \cdot F_G(n-1,n))$ for the decomposition step. This complexity can be reduced to $\mathrm{O}(N \cdot F_G(n-1,n))$ for algorithms that solve the k-median using dynamic programming which by construction solves all the smaller k'-medians, $1 \leqslant k' < k$, in order to solve the k-median. The algorithms of Tamir and Vigneron et al. fall in this category and thus allow to solve the entire series of k-medians for a given object o_j—"all-in-once"—by solving only the largest problem, i.e., the S'-median. The third and the fourth columns of Table 1 summarize the complexity of the k-median ($F_G(k,n)$) and of the resulting decomposition step (that uses the k-median algorithm of the previous column), for the following cases: directed tree, undirected tree, and general graph.

Table 1
The six columns of the table contain the following information: (1) type of the studied graph G; (2) type of available solutions for the k-median and CAP for the given type of G; (3) complexity of best k-median algorithm for G; (4) complexity of the decomposition step of CAP that uses the k-median algorithm of the previous column; (5) complexity of the composition (packing) step of CAP using the general dynamic programming algorithm; (6) overall complexity of CAP

Graph	Solution	Single k-median	Decomposition	Composition	Overall CAP
undirected tree	exact	$\mathrm{O}(kn^2)$ (Tamir)	$\mathrm{O}(n^3N)$	$\mathrm{O}(n^2N^2)$	$\mathrm{O}(\max\{n^3N, n^2N^2\})$
directed tree	exact	$\mathrm{O}(k^2n^2)$ (Vigneron)	$\mathrm{O}(n^4N)$	$\mathrm{O}(n^2N^2)$	$\mathrm{O}(\max\{n^4N, n^2N^2\})$
general	approx.	$\mathrm{O}(n^p)$ (Arya)	$\mathrm{O}(n^{p+1}N)$	$\mathrm{O}(n^2N^2)$	$\mathrm{O}(\max\{n^{p+1}N, n^2N^2\})$

4.2. Composition step: a packing problem

In this step the following packing problem, which is a special type of integer linear program (ILP), is solved optimally by appropriate polynomial time algorithms.

Maximize:

$$\sum_{j=1}^{N}\sum_{k=1}^{S'} G_{jk}\cdot Y_{jk}. \tag{8}$$

Subject to:

$$\sum_{k=1}^{S'} Y_{jk}\leqslant 1,\quad 1\leqslant j\leqslant N \tag{9}$$

and,

$$\sum_{j=1}^{N}\sum_{k=1}^{S'} k\cdot Y_{jk}\leqslant S, \tag{10}$$

$Y_{jk}\in\{0,1\},\ 1\leqslant j\leqslant N,\ 1\leqslant k\leqslant S'$ is a binary integer variable taking the value one if and only if the optimal (j,k)-placement, $\mathcal{P}_j^{(k)}$, is selected for inclusion in the optimal S-placement, $\mathcal{P}^{(S)}$, for CAP. For this packing problem two exact polynomial time algorithms are described; the first is a dynamic programming algorithm appropriate for all instances of CAP, while the second is a greedy algorithm that yields the exact optimal solution for a broad family of CAP instances having concave G_{jk}'s. When applicable, the greedy algorithm provides for a substantial reduction of complexity as compared to the more general dynamic programming algorithm.

4.2.1. Exact dynamic programming algorithm for the packing problem

Fig. 1 gives an illustrative abstraction of the packing problem described in the aforementioned ILP formulation. There are N "boxes" with indices j, $1\leqslant j\leqslant N$, and each box contains S' "objects" of type j with indices (j,k), $1\leqslant k\leqslant S'$. Object (j,k) of box j has a *value* G_{jk} and a *weight* k. There is also a knapsack that can hold a weight that is at most equal to S. The packing problem amounts to filling the knapsack (constraint (10)) by selecting at most one object from each box (constraint (9)) such that the total value of the objects that fit in the knapsack be maximized (objective function (8)). This description resembles much

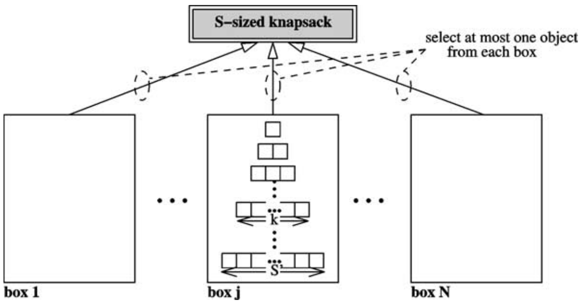

Fig. 1. The packing problem involved in the composition step.

the 0/1 knapsack problem [8,9] with the main difference be that whereas in the 0/1 knapsack there is only one object of type j, here there are S' objects of type j each having its own weight and value. This packing problem can be solved by an appropriately modified version of the dynamic programming algorithm that solves the original 0/1 knapsack [9].

Define $V[j,s]$ to be the maximum value of the objects that fit into a knapsack of size s when the selection is confined among objects of type $1,\dots,j$, $1\leqslant j\leqslant N$ and $0\leqslant s\leqslant S$. The objective is to identify $V[N,S]$ and the selection of objects that leads to it. First define the initial conditions:

$$V[j,0]=0,\quad 1\leqslant j\leqslant N,$$

$$V[1,s]=\begin{cases} G_{1,s}, & \text{if } 1\leqslant s\leqslant S',\\ G_{1,S'}, & \text{if } S'<s\leqslant S.\end{cases}$$

Now define the recursive relationship for the cases $2\leqslant j\leqslant N,\ 1\leqslant s\leqslant S$:

$$V[j,s] = \max\Big\{V[j-1,s],\ \max_{1\leqslant k\leqslant \min\{s,S'\}}\big\{V[j-1,s-k]+G_{jk}\big\}\Big\}. \tag{11}$$

Proposition 1. *The dynamic programming algorithm defined by the recursive equation* (11) *solves optimally the packing problem defined by Eqs.* (8)–(10).

The proof is a straightforward generalization of the proof of the correctness of the dynamic programming solution for the standard 0/1 knapsack problem and is, thus, omitted.

Since there are NS subproblems $V[j,s]$ to be solved and each subproblem requires at most S' comparisons to identify whether to add to the knapsack one of the objects (j,k) of box j, the time complexity of the packing problem is $\mathrm{O}(NSS')$. In general such an algorithm is called pseudopolynomial as its running time depends not only on the size of the input (here N,n) but on the magnitude of some of the parameters as well (here on S). In the context of the stated CAP problem, however, the parameter S is upper bounded by $S \leqslant nN$, capturing the fact that CAP becomes meaningless when the available storage capacity is larger than the minimum required to store all N objects on all n nodes of the graph. The upper bound guarantees that the dynamic programming algorithm will terminate in polynomial time. Using $S \leqslant nN$ and $S' \leqslant n$ the complexity of the packing becomes $\mathrm{O}(n^2N^2)$. Notice that this complexity does not depend upon the type of graph G.

4.2.2. *Exact greedy algorithm for the packing problem under concave G_{jk}'s*

It is possible to reduce the complexity of the packing problem to $\mathrm{O}(nN\log N)$, for an interesting family of CAP instances involving G_{jk}'s that are concave with respect to k, for all $o_j \in \mathcal{O}$. This can be achieved by replacing the dynamic programming algorithm with a faster greedy algorithm that, however, when operating on concave G_{jk}'s is guaranteed to be leading to an optimal solution for the packing problem.

The concaveness of G_{jk}'s implies that the "value" of adding more capacity is progressively declining. This is the case under many topologies and request patterns. The test for concaveness may be implemented at no additional cost while computing the G_{jk}'s at the first step. When the test succeeds, the following sketched greedy algorithm may be employed for solving the packing problem.

The algorithm assigns the total capacity in exactly S iterations, with each iteration assigning an additional capacity unit. The first unit goes to the object that has the largest G_{j1}, $1 \leqslant j \leqslant N$. Then each additional unit goes to an object of type j^*, if and only if the *incremental gain* for j^*, $G_{j^*k+1} - G_{j^*k}$, is larger than any other available incremental gain for any other type of object. Referring back to Fig. 1, this amounts to selecting object $(j^*, k+1)$ instead of object (j^*, k), so as to maximize $G_{jk+1} - G_{jk}$, $1 \leqslant j \leqslant N$, where (j,k) is the object of maximum weight that has already been chosen from box j at a prior iteration.

The greedy mode of operation combined with the concaveness of G_{jk}'s guarantee the identification of the optimal solution. By using a max-heap data structure to store the largest available incremental gain for an unselected unit of each of the N distinct objects, the algorithm can be implemented at a complexity of $\mathrm{O}(nN\log N)$ ($\mathrm{O}(N)$ for the initial creation of the max-heap plus $S \leqslant nN$ iterations with each iteration requiring $\mathrm{O}(\log N)$ complexity to re-organize the max-heap). Notice that the greedy solution for the fractional knapsack problem is not applicable here because the current packing problem involves objects with a non-constant per-unit value (the normalized per-unit value of (j,k) decreases with k).

4.3. *Correctness of the two-step algorithm*

In this section it is established that the solution obtained from the aforementioned two-step algorithm is an optimal solution to the CAP problem defined in Section 3. Informally, a description of the correctness proof goes as follows. First it is shown that the objective function of CAP is separable with regard to the different objects. Then it is shown that an optimal S-placement for CAP must contain only (j,k)-placements that are optimal solutions to k-median problems (for different objects o_j). The last argument allows to limit the search for an optimal S-placement to S-placements that contain only optimal k-median (j,k)-placements. This reduces the original CAP to the packing problem solved in the composition step.

Observation 1. *The optimal (j,k)-placement, $\mathcal{P}_j^{(k)} \in \mathbb{P}_j^{(k)}$, depends only on j for a given k and graph G.*

This can be verified by looking at Eq. (7) that defines the gain under $\mathcal{P}_j^{(k)}$ to be the maximum of $\psi_j(P_j^{(k)})$, $P_j^{(k)} \in \mathbb{P}_j^{(k)}$. Under given k and graph G the function $\psi_j(P_j^{(k)})$ depends only on j through r_{ij}'s, $1 \leqslant i \leqslant n$ (see (5) for the definition of $\psi_j(P_j^{(k)})$).

Proposition 2. *Let the optimal S-placement be $\mathcal{P}^{(S)} = \{\dot{P}_{j_1}^{(k_{j_1})}, \ldots, \dot{P}_{j_R}^{(k_{j_R})}\}$. Then $\dot{P}_{j_h}^{(k_{j_h})} = \mathcal{P}_{j_h}^{(k_{j_h})}$ for $1 \leqslant h \leqslant R$.*

S0020-0190(03)00537-4/SCO AID:3014 Vol.●●●(●●●) ipl3014 P.7(1-7)
ELSGMLTM(IPL):m3 v 1.181 Prn:12/01/2004; 14:33 by:PS p. 7

Proof. This can be shown via contradiction. Since $\mathcal{P}^{(S)}$ has been assumed to be optimal it must be that $g(\mathcal{P}^{(S)}) \geqslant g(P^{(S)})$ for all $P^{(S)} \in \mathbb{P}^{(S)}$. Suppose that there exists h': $1 \leqslant h' \leqslant R$ such that $\dot{P}_{j_{h'}}^{(k_{j_{h'}})} \neq \mathcal{P}_{j_{h'}}^{(k_{j_{h'}})}$. A new solution $\mathfrak{P}^{(S)}$ may then be constructed by setting $\mathfrak{P}_{j_h}^{(k_{j_h})} = \dot{P}_{j_h}^{(k_{j_h})}$ for $1 \leqslant h \leqslant h'-1$ and $h'+1 \leqslant h \leqslant R$ and setting $\mathfrak{P}_{j_{h'}}^{(k_{j_{h'}})} = \mathcal{P}_{j_{h'}}^{(k_{j_{h'}})}$. Since $\psi_{j_{h'}}(\mathfrak{P}_{j_{h'}}^{(k_{j_{h'}})}) = \psi_{j_{h'}}(\mathcal{P}_{j_{h'}}^{(k_{j_{h'}})}) > \psi_{j_{h'}}(\dot{P}_{j_{h'}}^{(k_{j_{h'}})})$ it turns that $g(\mathfrak{P}^{(S)}) > g(\mathcal{P}^{(S)})$ which is a contradiction as $\mathcal{P}^{(S)}$ has been assumed to be optimal. Thus it must be that $\dot{P}_{j_{h'}}^{(k_{j_{h'}})} = \mathcal{P}_{j_{h'}}^{(k_{j_{h'}})}$ for all $1 \leqslant h' \leqslant R$. □

Having established that an optimal S-placement $\mathcal{P}^{(S)}$ contains only optimal (j,k)-placements $\mathcal{P}_j^{(k)}$ it becomes possible to limit the search for $\mathcal{P}^{(S)}$ in $\widetilde{\mathbb{P}}^{(S)} \subset \mathbb{P}^{(S)}$ where $\widetilde{\mathbb{P}}^{(S)}$ denotes the set of S-placements that contain only optimal (j,k)-placements. Thus for the identification of $\mathcal{P}^{(S)}$ it suffices to look for the S-placement that maximizes the following re-written expression for the $g(P^{(S)})$ of (6):

$$g(P^{(S)}) = \sum_{h=1}^{R} G_{j_h k_{j_h}}, \quad P^{(S)} \in \widetilde{\mathbb{P}}^{(S)}. \tag{12}$$

Proposition 3. *Let the optimal S-placement be $\mathcal{P}^{(S)} = \{\mathcal{P}_{j_1}^{(k_{j_1})}, \ldots, \mathcal{P}_{j_R}^{(k_{j_R})}\}$. Then,*

$$\mathcal{P}^{(S)} = \left\{\mathcal{P}_{j_1}^{(k_{j_1})}, \ldots, \mathcal{P}_{j_R}^{(k_{j_R})}\right\} \quad \Longleftrightarrow \quad Y_{j_h k_{j_h}} = 1, \quad 1 \leqslant h \leqslant R.$$

Proof. The equivalence is a direct consequence of the following two facts: (1) The objective function of CAP given in Eq. (12) is equivalent to the objective function of the composition step given in Eq. (8) (by way of constraint (9)); (2) the optimal solution of CAP is known to be included in $\widetilde{\mathbb{P}}^{(S)}$ which matches exactly the feasible region defined by constraints (9), (10) of the composition step. □

5. Concluding remarks

The two-step algorithm finds an optimal solution to CAP in polynomial time, in the case of a tree graph, while it obtains a constant factor approximation in polynomial time, in the case of an arbitrary graph. The last column of Table 1 summarizes the exact overall complexity for the various cases. Notice that the final complexity is dominated by the complexity of the most complex step, which in turn depends on the relative magnitude of n, N for specific applications. In problems having $N \gg n$ (e.g., web caching, content distributions, see Section 1) it will be the packing problem that dominates the overall complexity. In such cases the greedy solution to the packing problem, when applicable, will offer a valuable speedup.

References

[1] V. Arya, N. Garg, R. Khandekar, K. Munagala, V. Pandit, Local search heuristic for k-median and facility location problems, in: Proc. 33rd Annual Symp. on Theory of Computing (ACM STOC), ACM Press, New York, 2001, pp. 21–29.

[2] I.D. Baev, R. Rajaraman, Approximation algorithms for data placement in arbitrary networks, in: Proc. 12th Annual Symp. on Discrete Algorithms (ACM-SIAM SODA), January 2001, pp. 661–670.

[3] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems, II: p-medians, SIAM J. Appl. Math. 37 (1979) 539–560.

[4] M.R. Korupolu, C.G. Plaxton, R. Rajaraman, Analysis of a local search heuristic for facility location problems, in: Proc. 9th Annual Symp. on Discrete Algorithms (ACM-SIAM SODA), 1998, pp. 1–10.

[5] M.R. Korupolu, C.G. Plaxton, R. Rajaraman, Placement algorithms for hierarchical cooperative caching, in: Proc. 10th Annual Symp. on Discrete Algorithms (ACM-SIAM SODA), 1999, pp. 586–595.

[6] N. Laoutaris, V. Zissimopoulos, I. Stavrakakis, Storage capacity allocation algorithms for hierarchical content distribution, submitted work.

[7] J.-H. Lin, J.S. Vitter, Approximation algorithms for geometric median problems, Inform. Process. Lett. 44 (1992) 245–249.

[8] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Dover Publications, New York, 1998.

[9] M. Syslo, N. Deo, J.S. Kowalik, Discrete Optimization Algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[10] A. Tamir, An O(pn^2) algorithm for p-median and related problems on tree graphs, Oper. Res. Lett. 19 (1996) 59–64.

[11] A. Vigneron, L. Gao, M.J. Golin, G.F. Italiano, B. Li, An algorithm for finding a k-median in a directed tree, Inform. Process. Lett. 74 (2000) 81–88.