Computer Networks 125 (2017) 160–171

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Distributed algorithms for content placement in hierarchical cache networks

Slađana Jošilo, Valentino Pacifici*, György Dán

Department of Communication Networks and ACCESS Linnaeus Center, School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden

ARTICLE INFO

Article history:
Received 2 September 2016
Revised 17 February 2017
Accepted 26 May 2017
Available online 27 May 2017

Keywords:
Content caching
Distributed placement
Hierarchical cache network
Mobile backhaul
Approximation ratio
Heuristics

ABSTRACT

The growing popularity of mobile multimedia content and the increase of wireless access bitrates are straining backhaul capacity in mobile networks. A cost-effective solution to reduce the strain, enabled by emerging all-IP 4G and 5G mobile backhaul architectures, could be in-network caching of popular content during times of peak demand. Motivated by the potential benefits of caching in mobile backhaul networks, in this paper we formulate the problem of content placement in a hierarchical cache network as a binary integer programming problem. We provide a polynomial time solution when the link costs are induced by a potential and we propose a 2-approximation algorithm for the general case. The 2-approximation requires full information about the network topology and the link costs, as well as about the content demands at the different caches, we thus propose two distributed algorithms that are based on limited information on the content demands. We show that the distributed algorithms terminate in a finite number of steps, and we provide analytical results on their approximation ratios. We use simulations to evaluate the proposed algorithms in terms of the achieved approximation ratio and computational complexity on hierarchical cache network topologies as a model of mobile backhaul networks.

1. Introduction

The penetration of high speed mobile access technologies, such as HSDPA and LTE, together with the proliferation of powerful handheld devices has stimulated a rapid increase of user demand for mobile multimedia content in recent years. The traffic growth is predicted to continue in coming years, with an estimated 10-fold increase in mobile data traffic in 5 years and an increasing peak-to-average traffic ratio, and puts significant strain on mobile backhaul capacity.

Recent measurement studies of mobile data traffic indicate that caching could be an effective means of decreasing the mobile backhaul bandwidth requirements: caching could reduce the bandwidth demand by up to 95% during peak hours and could at the same time reduce content delivery time by a factor of three as shown in [1]. Such a high cache efficiency is likely due to the concentration of content popularity to relatively few content items during peak hours, a phenomenon that has been observed for, e.g., multimedia content [2]. At the same time, mobile traffic is dominated by downloads; up to 75% of daily traffic load comes from download traffic, and the demand shows significant diurnal fluctuations with low loads during early morning hours [3].

Tunelling imposed by previous 3GPP standards made backhaul in-network caching technically challenging [4], allowing only caches at the network edge, in emerging all-IP mobile backhaul architectures the caches could be co-located with every switch and could implement cooperative caching policies throughout the backhaul. Since fairly accurate content popularity predictions can be obtained for Web and video content [5,6], the most popular contents could be downloaded into the caches of the mobile backhaul in the early morning hours when the load is relatively low, thereby alleviating the traffic demand during peak hours. A similar approach could be used in content distribution networks (CDNs) and edge caching architectures in wireline networks, as done in the Netflix Open Connect program [7].

Given predicted content popularities, a fundamental problem of in-network caching is to find efficient content placement algorithms that take into consideration the characteristics of the network topology and of the content workload. The algorithms should achieve close to optimal bandwidth cost savings and should have low computational complexity. Furthermore, they should require as little information as possible, e.g., about content popularities and network topology, in order to allow fully distributed operation and scaling to large topologies with small communication overhead.

* Corresponding author.
E-mail addresses: josilo@kth.se (S. Jošilo), pacifici@kth.se (V. Pacifici), gyuri@kth.se (G. Dán).

http://dx.doi.org/10.1016/j.comnet.2017.05.029
1389-1286/

While previous works proposed centralized and distributed content placement algorithms for two-level hierarchical topologies [8], general topologies with an ultrametric [9], and topologies in a metric space [10], efficient distributed algorithms based on limited topological information have received little attention.

Motivated by the observation that mobkile backhaul networks and edge caching architectures can often well be modeled by a cache hierarchy, in this paper we formulate the problem of content placement based on predicted demands in a hierarchical cache network with asymmetric link costs. We formulate the problem as a 0-1 integer programming problem, and show that under the potential induced link cost model the problem can be solved in polynomial time. For the general case we show that a 2-approximation to the problem can be obtained using a distributed greedy algorithm when global information is available, and propose two computationally simple distributed algorithms that do not require global information. We evaluate the algorithms through extensive simulations on various network topologies. Our results show that information about object demands at descendants is not sufficient for achieving good performance, but the proposed *h*-Push Down algorithm achieves consistently good performance based on a limited amount of information about object placements.

The rest of the paper is organized as follows. Section 2 describes the system model and provides the problem formulation. Section 4 describes the 2-approximation algorithm based on global information, and Section 5 describes the distributed algorithms based on limited information. Section 6 shows performance results based on simulations. Section 7 discusses related work and Section 8 concludes the paper.

2. System model and problem formulation

We consider a typical mobile backhaul, which is the primary motivating use case of our work, and model its active topology by a symmetric acyclic directed graph $\mathcal{G}(\mathcal{N}, E)$, where the vertices $\mathcal{N}$ are routers that connect cell sites and may aggregate traffic from other routers (and thus cell cites), and for every connected pair of nodes $i, j \in \mathcal{N}$ there exist edges $(i, j) \in E$ and $(j, i) \in E$. Observe that since $\mathcal{G}$ is connected and acyclic, $\mathcal{G}$ is a tree. The assumption that the active topology is a tree is realistic for the access part of the mobile backhaul in urban environments. We denote by $\mathcal{L}$ the set of leaf nodes in $\mathcal{G}$, by $\mathcal{I}$ the set of internal nodes and by n_0 the root node, i.e., $\mathcal{N} = \mathcal{L} \cup \mathcal{I} \cup n_0$. We denote the unique simple path from node i to node j by $P_{i,j} = \Big((i, v_1), (v_1, v_2), \ldots, (v_{|P_{i,j}|-1}, j)\Big)$, and we denote by $|P_{i,j}|$ the number of edges in path $P_{i,j}$. Observe that $|P_{i,j}| = |P_{j,i}|$. We define the level $l(i)$ of node $i \in \mathcal{N}$ in the tree $\mathcal{G}$ as the number of edges from node i to the tree's root node n_0 in the unique simple path from i to n_0, i.e., $l(i) = |P_{i,n_0}|$. We denote the children of node $i \in \mathcal{N}$ by $\mathcal{C}(i) \triangleq \{j | (i, j) \in E \wedge l(j) > l(i)\}$ and the parent of node i by $\mathcal{P}(i)$, where $\mathcal{P}(i) \in \mathcal{N}$ such that $i \in \mathcal{C}(\mathcal{P}(i))$. We denote by $\mathcal{P}^l(i)$ the l^{th}-ancestor of node i, e.g., $\mathcal{P}^2(i) = \mathcal{P}(\mathcal{P}(i))$. By definition $\mathcal{P}^0(i) = i$. We refer to an edge (i, j) as the downlink direction if $j \in \mathcal{C}(i)$ and as the uplink direction if $i \in \mathcal{C}(j)$.

We say that two nodes are siblings if they have the same parent, and define the sibling set $\mathcal{S}(i) \triangleq \{j | \mathcal{P}(j) = \mathcal{P}(i) \wedge i \neq j\}$. We denote the descendants of node i by $\mathcal{D}(i) \triangleq \{j | l(j) > l(i) \wedge \text{LCA}(i, j) = i\}$, where LCA$(i, j)$ denotes the lowest common ancestor of nodes i and j, furthermore we use the notation $\mathcal{G}_i(\mathcal{N}_i, E_i)$ for the subgraph induced by $\mathcal{N}_i = \{i\} \cup \mathcal{D}(i)$ rooted in i.

2.1. Objects, demand and storage

We denote the set of objects requested by mobile nodes by $\mathcal{O}$. We follow common practice and consider that every object has unit size [11,12], which is a reasonable simplification if content is divisible into unit-sized chunks. We denote the average request rate (demand) predicted for the peak hours for object $o \in \mathcal{O}$ at the cell site connected to node i by w_i^o.

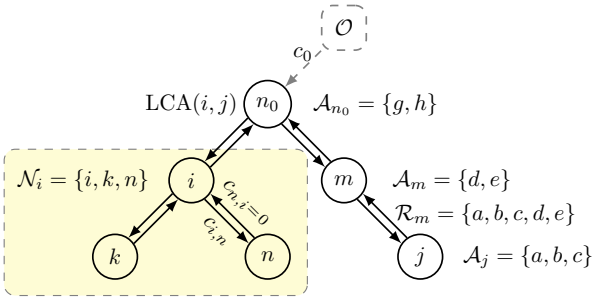

Fig. 1. Example hierarchical cache network with nodes in three levels, showing commonly used notation.

Every node $i \in \mathcal{N}$ is equipped with a cache, and we denote the size of the cache at node i by K_i. We denote the set of objects stored in the cache at node i by $\mathcal{A}_i \subset \mathcal{O}$, $|\mathcal{A}_i| \leq K_i$. We use the shorthand notation $\mathcal{A}_V \triangleq (\mathcal{A}_j)_{j \in V}$, where $V \subseteq \mathcal{N}$, and $\mathcal{A}_{-i} \triangleq (\mathcal{A}_j)_{j \in \mathcal{N} \setminus \{i\}}$. We denote by $\mathfrak{A}_i$ the set of object placements that satisfy the storage capacity constraint at node i, i.e. $\mathfrak{A}_i = \{\mathcal{A}_i \in 2^{\mathcal{O}} : |\mathcal{A}_i| \leq K_i\}$, where $2^{\mathcal{O}}$ is the powerset of $\mathcal{O}$. Finally, we denote the set of objects stored at node i and at its descendants by $\mathcal{R}_i(\mathcal{A}) = \mathcal{A}_i \bigcup_{j \in \mathcal{D}(i)} \mathcal{R}_j(\mathcal{A})$. Fig. 1 shows an example topology with a maximum level of 2, illustrating some of the commonly used notation.

2.2. Cost model

We denote the unit cost of using edge (i, j) by $c_{i,j}$. Since during peak hours most of the traffic in a mobile backhaul is flowing downlink (serving users' requests for content) [1,3], we consider that uplink edges have zero unit cost, i.e., $c_{i,\mathcal{P}(i)} = 0$. Without loss of generality, the cost of downlink edges is $c_{\mathcal{P}(i),i} > 0$. We consider that edge costs are additive, i.e., if a request for object o arrives at node i and is served from node j then the unit cost is $d_{i,j} = \sum_{(v,w) \in P_{j,i}} c_{v,w}$. We call $d_{i,j}$ the *distance* from node j to node i. Note that the terms $c_{v,w}$ are zero if they correspond to an uplink, i.e., if $w = \mathcal{P}(v)$. Furthermore, observe that in general $d_{j,i} \neq d_{i,j}$, thus distance is not symmetric (hence it is a hemimetric).

A request for object o generated by a mobile user connected to the cell site at node $i \in \mathcal{N}$ is served locally if $o \in \mathcal{A}_i$. Otherwise, if node i has a descendant $j \in \mathcal{D}(i)$ for which $o \in \mathcal{A}_j$, the node forwards the request to the closest such descendant. Otherwise, node i forwards the request to its parent $\mathcal{P}(i)$, which follows the same algorithm for serving the request. If an object o is not stored in any node (i.e., $o \notin \mathcal{R}_{n_0}$) then it needs to be retrieved through the Backbone via the root node n_0 at a unit cost of c_0.

Given a placement $\mathcal{A} = (\mathcal{A}_j)_{j \in \mathcal{N}}$ we can define the unit cost to serve a request for object o at node i as

$$d_i(o, \mathcal{A}) = \begin{cases} \min\limits_{\{j \in \mathcal{N} | o \in \mathcal{A}_j\}} d_{i,j} & \text{if } o \in \mathcal{R}_{n_0} \\ d_{i,n_0} + c_0 & \text{if } o \notin \mathcal{R}_{n_0}, \end{cases}$$

which together with the demand w_i^o determines the cost incurred by node i as

$$C_i(\mathcal{A}) = \sum_{o \in \mathcal{O}} C_i^o(\mathcal{A}) = \sum_{o \in \mathcal{O}} w_i^o d_i(o, \mathcal{A}). \quad (1)$$

Finally, we define the total cost $C(\mathcal{A}) = \sum_{i \in \mathcal{N}} C_i(\mathcal{A})$.

2.3. Problem formulation

Motivated by minimizing the congestion in the mobile backhaul during peak hours, our objective is to find a placement that mini-

mizes the total cost $C(\mathcal{A})$. We refer to this as the mobile backhaul content placement problem (MBCP), which can be formulated as finding $\bar{\mathcal{A}} = \text{argmin}_{\mathcal{A}\in\times_{i\in\mathcal{N}}\mathfrak{A}_i} C(\mathcal{A})$.

It is easy to see that the MBCP problem can be formulated as the following $0-1$ integer linear program

$$\min \sum_{i\in\mathcal{N}}\sum_{o\in\mathcal{O}} w_i^o \left(\sum_{j\in\mathcal{N}, j\neq i} d_{i,j} x_{i,j,o} + (d_{i,n_0} + c_0) x_{i,-1,o} \right) \quad \text{s.t.}$$
$$\sum_{o\in\mathcal{O}} x_{i,o} \leq K_i, \qquad \forall i \in \mathcal{N} \tag{2}$$

$$x_{i,j,o} \leq x_{j,o}, \qquad \forall i, j \in \mathcal{N}, o \in \mathcal{O} \tag{3}$$

$$\sum_{j\in\mathcal{N}} x_{i,j,o} + x_{i,-1,o} \geq 1, \qquad \forall i \in \mathcal{N}, o \in \mathcal{O} \tag{4}$$

$$x_{i,o}, x_{i,j,o}, x_{i,-1,o} \in \{0,1\}, \tag{5}$$

where $x_{i,o}$ indicates whether object o is in the storage of node i (i.e. $x_{i,o} = 1 \Leftrightarrow o \in \mathcal{A}_i$), $x_{i,j,o}$ indicates whether a request for object o at node i is served from node j, and $x_{i,-1,o}$ indicates whether object o is retrieved from the Backbone, i.e., the *level* of the Backbone is indicated with -1.

3. Centralized algorithms

We start with considering centralized algorithms for finding an optimal solution for the MBCP. First we show that, in the general case, the MBCP problem cannot be solved in polynomial time through linear relaxation, as the constraint matrix (2)–(4) is not totally unimodular.

3.1. Arbitrary link costs

We start with the following definition.

Definition 1. Let A be a matrix with entries 0, $+1$, or -1. A is totally unimodular if and only if each square submatrix of A has determinant 0, $+1$, or -1.

We now recall a result from Schrijver [13] on the relation between total unimodularity and integer linear programming.

Theorem 1 (Hoffman and Kruskal's Theorem [13])**.** *Let A be an integral matrix. A is totally unimodular if and only if for each integral vector b the polyhedron $\{x | x \geq 0; Ax \leq b\}$ is integral.*

It follows from Theorem 1 that if the constraint matrix A of the MBCP problem is totally unimodular, then the linear relaxation of the MBCP problem would have integral solutions that would correspond to the solutions of the $0-1$ integer linear program. In what follows we show that the constraint matrix A is not totally unimodular for non-trivial instances of the MBCP problem, as it is shown by the following Lemma.

Lemma 1. *The constraint matrix for the MBCP problem is not totally unimodular for $|\mathcal{N}| \geq 4$.*

Proof. Consider a MBCP problem defined for a system of $|\mathcal{N}| \geq 4$ nodes and an arbitrary number of objects. The constraint matrix A has $|\mathcal{N}||\mathcal{O}| + |\mathcal{N}|^2|\mathcal{O}| + |\mathcal{N}||\mathcal{O}|$ rows and $|\mathcal{N}||\mathcal{O}| + 2\cdot|\mathcal{N}|^2|\mathcal{O}| + (|\mathcal{N}|+1)\cdot|\mathcal{N}||\mathcal{O}|$ non-zero entries. In the following we show that there exists a 9×9 square sub-matrix A' of the constraint matrix A that has 18 non-zero entries and is such that $\det(A') = 2$.

We construct the sub-matrix A' for one arbitrary object o, by initially selecting a subset of 6 rows of A corresponding to constraint (3), such that $(i, j) \in \{(1,2), (2,2), (1,3), (3,3), (2,4), (3,4)\}$. Each selected row has exactly two non-zero entries, we select the columns of A' corresponding to the non-zero entries in the selected rows. Finally, we add to A' the 3 rows of A, corresponding to constraint (4), for nodes $i \in \{1,2,3\}$ and object o. The resulting matrix A' is illustrated in Table 1.

It is easy to verify that $\det(A') = 2$, which proves the result. □

A consequence of Lemma 1 is that solving the MBCP problem might be computationally infeasible already for moderate sized instances of the problem. We are thus interested in finding computationally feasible, scalable distributed algorithms to approximate the solution of the MBCP problem in the general case.

3.2. Potential induced downlink costs

In what follows, we consider a special case of the MBCP problem where the downlink costs are *potential induced*. We start with the following definition.

Definition 2. A function $\Psi : \mathcal{N} \mapsto \mathbb{R}^+$ is a *potential* if it assigns one label to each of the nodes in $\mathcal{N}$, such that each node's label is greater than any label of its children, i.e., $\Psi(\mathcal{P}(i)) > \Psi(i),\ \forall i \in \mathcal{I}$.

The downlink cost $c_{\mathcal{P}(i),i}$ is *induced* by the potential function Ψ if it tantamounts the potential difference between the nodes, i.e., $c_{\mathcal{P}(i),i} = \Psi(\mathcal{P}(i)) - \Psi(i)$. Observe that, for the root node n_0, $\Psi(\mathcal{P}(n_0)) = \Psi(n_0) + c_0$. If the downlink costs are induced by Ψ, the distance $d_{i,j}$ between two arbitrary nodes $i, j \in \mathcal{N}$ is $d_{i,j} = \sum_{(v,w)\in P_{j,i}} c_{v,w} = \Psi(\text{LCA}(i,j)) - \Psi(i)$.

Theorem 2. *There exists a centralized algorithm that solves the MBCP problem with potential induced downlink costs in polynomial time.*

Proof. We prove the theorem by showing that, if the downlink costs are potential induced then the MBCP problem can be reduced to a minimum-cost flow problem. We assume that graph $\mathcal{G}$ is non-singleton, i.e. $|\mathcal{N}| > 1$, and we construct a directed graph $\mathcal{G}' = (\mathcal{N}', E')$ as follows. We define the set $R' \triangleq \{\langle o_k, n_0\rangle | k = 1, 2, \ldots |\mathcal{O}|\}$ of $|\mathcal{O}|$ nodes and the set $\mathcal{I}' \triangleq \mathcal{O}\times\mathcal{I}$ of $|\mathcal{O}||\mathcal{I}|$ nodes . The set of nodes in graph $\mathcal{G}'$ is defined as $\mathcal{N}' \triangleq \{S\} \cup R' \cup \mathcal{I}' \cup \mathcal{N} \cup \{T\}$, where S and T are source and sink nodes, respectively. The edge set E' consists of five types of edges: (1) for each $\langle o_k, n_0\rangle \in R'$ there are two parallel edges $(S, \langle o_k, n_0\rangle)$ with capacities 1 and ∞, and costs $-\sum_{j\in\mathcal{N}} w_j^{o_k} c_0$ and 0, respectively; (2) for each $\langle o_k, i\rangle \in \mathcal{I}'$ there are two parallel edges $(\langle o_k, \mathcal{P}(i)\rangle, \langle o_k, i\rangle)$, with capacities 1 and ∞, and costs $-\sum_{j\in\mathcal{N}_i} w_j^{o_k}[\Psi(\mathcal{P}(i)) - \Psi(i)]$ and 0, respectively; (3) for each $i \in \mathcal{I}\cup\{n_0\}$ and $o_k \in \mathcal{O}$ there is one edge $(\langle o_k, i\rangle, i)$ with capacity 1 and cost 0; (4) for each $i \in \mathcal{L}$ and $o_k \in \mathcal{O}$ there is one edge $(\langle o_k, \mathcal{P}(i)\rangle, i)$ with capacity 1 and cost $-w_i^{o_k}[\Psi(\mathcal{P}(i)) - \Psi(i)]$; (5) for each $i \in \mathcal{N} = \mathcal{L}\cup\mathcal{I}$ there is one edge (i, T) with capacity K_i and cost 0. We reduce the MBCP problem to the minimum-cost flow problem on graph $\mathcal{G}'$ by proving the following two lemmas.

Lemma 2. *Given a placement $\mathcal{A}$ there exists an integral flow $f^{\mathcal{A}}$ in $\mathcal{G}'$ with cost equal to $C(f^{\mathcal{A}}) = C(\mathcal{A}) - C(\varnothing)$, where $C(\varnothing)$ is the cost of an empty placement, i.e., $C(\varnothing) = \sum_{o\in\mathcal{O}} C^o(\varnothing) = \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} w_i^o[\Psi(n_0) - \Psi(i) + c_0]$.*

Proof. Let $n_i(o, \mathcal{A})$ be the number of occurrences of item o in placement $(\mathcal{A}_j)_{j\in\mathcal{N}_i}$ at node i and its descendants $\mathcal{D}(i)$, i.e., $n_i(o,\mathcal{A}) = |\{j \in \{i\}\cup\mathcal{D}(i) | o \in \mathcal{A}_j\}|$. We construct the flow $f^{\mathcal{A}}$ of value $\sum_{i\in\mathcal{N}} |\mathcal{A}_i|$ as follows. For each edge (i, T) of type 5, we set the flow to $|\mathcal{A}_i|$. For each edge $(\langle o_k, j\rangle, i)$ of type 4 and type 3, we set the flow to 1 if $o_k \in \mathcal{A}_i$, and to 0 otherwise. For each edge incident to $\langle o_k, i\rangle \in R'\cup\mathcal{I}'$, of type 2 and type 1 with capacity 1, we set the flow to 1 if $n_i(o_k, \mathcal{A}) > 0$, and to 0 otherwise. For each

Table 1
9×9 square sub-matrix A' of the constraint matrix A with $\det(A') = 2$, for instances of the MBCP problem with at least 4 nodes, i.e., $|\mathcal{N}| \geq 4$.

$x_{j,o}$			$x_{i,j,o}$						
$j=2$	$j=3$	$j=4$	$i=1$		$i=2$		$i=3$		
-1	0	0	1	0	0	0	0	0	$-x_{2,o} + x_{1,2,o} \leq 0$
-1	0	0	0	0	1	0	0	0	$-x_{2,o} + x_{2,2,o} \leq 0$
0	-1	0	0	1	0	0	0	0	$-x_{3,o} + x_{1,3,o} \leq 0$
0	-1	0	0	0	0	0	1	0	$-x_{3,o} + x_{3,3,o} \leq 0$
0	0	-1	0	0	0	1	0	0	$-x_{4,o} + x_{2,4,o} \leq 0$
0	0	-1	0	0	0	0	0	1	$-x_{4,o} + x_{3,4,o} \leq 0$
0	0	0	-1	-1	0	0	0	0	$-x_{1,2,o} - x_{1,3,o} - .. \leq -1$
0	0	0	0	0	-1	-1	0	0	$-x_{2,2,o} - x_{2,4,o} - .. \leq -1$
0	0	0	0	0	0	0	-1	-1	$-x_{3,4,o} - x_{3,4,o} - .. \leq -1$

edge incident to $\langle o_k, i\rangle \in R' \cup \mathcal{I}'$, of type 2 and type 1 with capacity ∞, we set the flow to $\max\{0, n_i(o_k, \mathcal{A}) - 1\}$. Flow $f^{\mathcal{A}}$ is feasible as each edge (i, T) of type 5 has assigned flow $|\mathcal{A}_i|$ and for all inbound edges $(\langle o_k, j\rangle, i)$ to node $i \in \mathcal{N}$, flow $f^{\mathcal{A}}$ is set to 1 only if $o_k \in \mathcal{A}_i$. In addition, for each node $\langle o_k, i\rangle \in R' \cup \mathcal{I}'$ the aggregate inbound flow is equal to $n_i(o_k, \mathcal{A})$. In the following we calculate the cost of flow $f^{\mathcal{A}}$ to prove the lemma. Observe that only the flow on edges of type 1, 2 and 4 affects the cost $C(f^{\mathcal{A}})$. It is possible to compute the cost $C(f^{\mathcal{A}})$ as

$$C(f^{\mathcal{A}}) = \sum_{o\in\mathcal{O}} C^o(f^{\mathcal{A}}) = \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} \mathbb{1}_{\mathcal{R}_i(\mathcal{A})}(o)\left(-\sum_{j\in\mathcal{N}_i} w_j^o[\Psi(\mathcal{P}(i)) - \Psi(i)]\right), \quad (6)$$

where $\mathbb{1}_{\mathcal{R}_i(\mathcal{A})}(o)$ is the indicator function of set $\mathcal{R}_i(\mathcal{A})$, and it is such that $\mathbb{1}_{\mathcal{R}_i(\mathcal{A})}(o) = 1 \Leftrightarrow n_i(o, \mathcal{A}) > 0$. Assuming potential induced downlink costs, the cost $C(\mathcal{A})$ can be rewritten as

$$\begin{aligned}
C(\mathcal{A}) &= \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} w_i^o\left[\sum_{l=0}^{l(i)}(1 - \mathbb{1}_{\mathcal{R}_{\mathcal{P}^l(i)}(\mathcal{A})}(o))c_{\mathcal{P}^{l+1}(i),\mathcal{P}^l(i)}\right] \\
&= \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} w_i^o\Bigg[\Psi(\mathcal{P}(n_0)) - \Psi(i) \\
&\quad + \sum_{l=0}^{l(i)} \mathbb{1}_{\mathcal{R}_{\mathcal{P}^l(i)}(\mathcal{A})}(o)\left[\Psi(\mathcal{P}^l(i)) - \Psi(\mathcal{P}^{l+1}(i))\right]\Bigg] \\
&= \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} w_i^o[\Psi(n_0) + c_0 - \Psi(i)] \\
&\quad + \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} w_i^o\left[\sum_{l=0}^{l(i)} \mathbb{1}_{\mathcal{R}_{\mathcal{P}^l(i)}(\mathcal{A})}(o)\left[\Psi(\mathcal{P}^l(i)) - \Psi(\mathcal{P}^{l+1}(i))\right]\right] \\
&= C(\varnothing) + \sum_{o\in\mathcal{O}}\sum_{i\in\mathcal{N}} \mathbb{1}_{\mathcal{R}_i(\mathcal{A})}(o)\sum_{j\in\mathcal{N}_i} w_j^o[\Psi(i) - \Psi(\mathcal{P}(i))] \\
&= C(\varnothing) + C(f^{\mathcal{A}}). \quad (7)
\end{aligned}$$

□

Lemma 3. *For every integral minimum-cost flow f with cost $C(f)$ in $\mathcal{G}'$, there exists a placement $\mathcal{A}^f$ such that $C(\mathcal{A}^f) = C(\varnothing) + C(f)$.*

Proof. Given an integral minimum-cost flow f in $\mathcal{G}'$, we define a corresponding placement $\mathcal{A}^f$ as follows. For each $i \in \mathcal{N}$ and each $o \in \mathcal{O}$, $o \in \mathcal{A}_i^f \Leftrightarrow f(\langle o, j\rangle, i) = 1$, where $(\langle o, j\rangle, i)$ is an edge of type 3 or 4. Observe that $\mathcal{A}^f$ is feasible, as $\Sigma_o f(\langle o, j\rangle, i) \leq K_i$. In the following we prove that $C(\mathcal{A}^f) = C(\varnothing) + C(f)$. Recall that for each $\langle o_k, i\rangle \in R' \cup \mathcal{I}'$ there are two parallel edges $(u, \langle o_k, i\rangle)$ with capacity 1 and ∞ and costs $-\sum_{j\in\mathcal{N}_i} w_j^{o_k} c_0$ and 0, respectively. Let us refer to such edges as $e_1^{\langle o_k,i\rangle}$ and as $e_\infty^{\langle o_k,i\rangle}$, respectively. We now show that $f(e_\infty^{\langle o_k,i\rangle}) > 0 \Rightarrow f(e_1^{\langle o_k,i\rangle}) = 1$ holds by contradiction. Assume that $f(e_\infty^{\langle o_k,i\rangle}) > 0$ and $f(e_1^{\langle o_k,i\rangle}) = 0$. Then there exists a flow f' defined as $f'(e_1^{\langle o_k,i\rangle}) = 1$, $f'(e_\infty^{\langle o_k,i\rangle}) = f(e_\infty^{\langle o_k,i\rangle}) - 1$ on edges $e_1^{\langle o_k,i\rangle}$ and $e_\infty^{\langle o_k,i\rangle}$, and $f'(e) = f(e)$ on any other edge $e \in E' \setminus \{e_1^{\langle o_k,i\rangle}, e_\infty^{\langle o_k,i\rangle}\}$. As edge $e_1^{\langle o_k,i\rangle}$ has negative cost, it follows that $C(f') < C(f)$. Thus, in any integral minimum cost flow $f(e_\infty^{\langle o_k,i\rangle}) > 0$ implies $f(e_1^{\langle o_k,i\rangle}) = 1$.

From the flow conservation constraint it follows that, when $o_k \in \mathcal{A}_j^f$ for some $j \in \mathcal{N}_i$, then $f(e_\infty^{\langle o_k,i\rangle}) + f(e_1^{\langle o_k,i\rangle}) > 0$, and consequently $f(e_1^{\langle o_k,i\rangle}) = 1$. By summing the flow on non-zero cost edges (i.e., edges of type 1, 2 and 4) we can compute the cost $C(f)$ of flow f as

$$\begin{aligned}
C(f) &= \sum_{\langle o_k,i\rangle\in R'\cup\mathcal{I}'}\left(-\sum_{j\in\mathcal{N}_i} w_j^{o_k}[\Psi(\mathcal{P}(i)) - \Psi(i)]\right)\mathbb{1}_{\mathcal{R}_i(\mathcal{A})}(o_k) \\
&\quad + \sum_{i\in\mathcal{L}}\sum_{o_k\in\mathcal{O}}\left(-w_i^{o_k}[\Psi(\mathcal{P}(i)) - \Psi(i)]\right)\mathbb{1}_{\mathcal{A}_i}(o_k) \\
&= \sum_{o_k\in\mathcal{O}}\sum_{i\in\mathcal{N}} \mathbb{1}_{\mathcal{R}_i(\mathcal{A})}(o_k)\sum_{j\in\mathcal{N}_i} w_j^{o_k}[\Psi(i) - \Psi(\mathcal{P}(i))]. \quad (8)
\end{aligned}$$

From (7) and (8) it follows that $C(\mathcal{A}^f) = C(\varnothing) + C(f)$, which proves the Lemma. □

Lemmas 2 and 3 imply that solving the minimum-cost flow problem in $\mathcal{G}'(\mathcal{N}', E')$ leads to finding an optimal object placement in $\mathcal{G}(\mathcal{N}, E)$, which proves Theorem 2. □

4. Distributed 2-approximation algorithm based on global information

In what follows we show that if global information is available about the object demands and placements at every node of the network, then it is possible to obtain a 2-approximation to the optimal solution using the *Depth First Greedy* (*DFG*) algorithm. The *DFG* algorithm is based on a depth-first traversal of the graph $\mathcal{G}$, i.e., an ordering $i_1, \ldots, i_{|\mathcal{N}|}$ of the vertices in $\mathcal{N}$, and can be executed by the nodes in an iterative (distributed) manner. The algorithm starts with an empty allocation ($\mathcal{A}_i = \emptyset$); at iteration $1 \leq k \leq |\mathcal{N}|$ node i_k populates its cache with K_{i_k} objects, one at a time, that provide the highest global cost saving. The *DFG* algorithm is shown in Fig. 2.

Theorem 3. *The* DFG *algorithm is a 2-approximation algorithm for the MBCP problem in terms of cost saving, i.e.,* $\frac{C(\emptyset) - C(\bar{A})}{C(\emptyset) - C(\mathcal{A}^{\text{DFG}})} \leq 2$.

As such, the *DFG* algorithm has a better approximation ratio for cache hierarchies with more than 2 levels than existing algorithms [14]. Before we prove the theorem we introduce some definitions and previous results.

DFG Algorithm

INPUT: DF Traversal $(i_1, \ldots, i_{|\mathcal{N}|})$
$k \leftarrow 1$
$\mathcal{A}_i \leftarrow \emptyset$, for all $i \in \mathcal{N}$
for $k = 1 \ldots |\mathcal{N}|$ **do**
  while $|\mathcal{A}_{i_k}| < K_{i_k}$ **do**
    $o^* \leftarrow \arg\max_{o \in \mathcal{O}} (C(\mathcal{A}_{-i_k}, \mathcal{A}_{i_k}) - C(\mathcal{A}_{-i_k}, \mathcal{A}_{i_k} \cup \{o\}))$
    $\mathcal{A}_{i_k} \leftarrow \mathcal{A}_{i_k} \cup \{o^*\}$
  end while
end for

Fig. 2. Pseudo-code of the *DFG* algorithm.

Definition 3. Let E be a finite set and let $\mathcal{F}$ be a collection of subsets of E. The pair $(E, \mathcal{F})$ is a *partition matroid* if $E = \bigcup_{i=1}^{k} E_i$ is the disjoint union of k sets, $l_1, \ldots, l_k$ are positive integers and $\mathcal{F} = \{F | F = \bigcup_{i=1}^{k} F_i, F_i \subseteq E_i, |F_i| \leq l_i, i = 1, \ldots, k\}$.

Definition 4. Let E be a finite set, and $f : 2^E \rightarrow \mathbb{R}$ a real valued function on subsets of E. Then f is submodular if for every $A, B \in E$ we have

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B).$$

Let us now recall a fundamental result about the maximization of submodular functions over partition matroids.

Lemma 4. *[15] Let $\mathcal{F}$ be a partition matroid over a set E, and $f : \mathcal{F} \rightarrow \mathbb{R}$ be a non-decreasing submodular function with $f(\emptyset) = 0$. Then the* DFG *algorithm achieves a 2-approximation of* $\max_{F \in \mathcal{F}} f(F)$.

In what follows we show that MBCP can be formulated as the maximization of a non-decreasing submodular function over a partition matroid. Let us define for every object $o \in \mathcal{O}$ one fictitious object (o, i) per node $i \in \mathcal{N}$, i.e., $(o, i) \in \mathcal{O} \times \mathcal{N}$. The set of fictitious objects that can be assigned to node i is then $\mathcal{E}_i = \{(o, i) | o \in \mathcal{O}\}$ and we define the set $\mathcal{E} = \bigcup_{i \in \mathcal{N}} \mathcal{E}_i$. We denote by $\mathfrak{A}$ the family of subsets of $\mathcal{E}$, defined as $\mathfrak{A} = \times_{i \in \mathcal{N}} \mathfrak{A}_i$, where $\mathfrak{A}_i \subseteq \mathcal{E}_i$, $|\mathfrak{A}_i| \leq K_i$ is the set of object placements that satisfy the storage capacity constraint at node i, as defined in Section 2.1.

Proposition 4. *The pair $(\mathcal{E}, \mathfrak{A})$ is a partition matroid.*

Proof. Consider an allocation $\mathcal{A} \in \mathfrak{A}$ and a fictitious object $(o, i) \in \mathcal{A}_i$. If we remove (o, i) from $\mathcal{A}_i$, i.e. $\mathcal{A}'_i = \mathcal{A}_i \setminus \{(o, i)\}$, then $\mathcal{A}'_i \subseteq \mathcal{E}_i$ will still hold as well as $\mathcal{A}_j \subseteq \mathcal{E}_j$, for $j \in \mathcal{N} \setminus \{i\}$, which implies that $(\mathcal{E}, \mathfrak{A})$ is an independence system.

Consider now two allocations $\mathcal{A}, \mathcal{A}' \in \mathfrak{A}$. If $|\mathcal{A}| < |\mathcal{A}'|$ then $\exists \mathcal{E}_i$ such that $|\mathcal{A}' \cap \mathcal{E}_i| > |\mathcal{A} \cap \mathcal{E}_i|$, which implies that there is a node $i \in \mathcal{N}$ with at least one free space in its cache, i.e. $|\mathcal{A}_i| < K_i$. Therefore, there is an $(o, i) \in (\mathcal{A}' \setminus \mathcal{A}) \cap \mathcal{E}_i$ such that $\mathcal{A} \cup \{(o, i)\} \in \mathfrak{A}$. □

Proof of Theorem 3.. We prove the theorem by showing that the function $\bar{C}(\mathcal{A}) = -C(\mathcal{A})$ is a nondecreasing submodular function on $\mathcal{E}$. Let us define the change of the global cost after inserting an object o in the cache of node i as $\Delta C(\mathcal{A}) = \bar{C}(\mathcal{A} \cup \{(o, i)\}) - \bar{C}(\mathcal{A})$, where $\mathcal{A} \in \mathfrak{A}$ and $\exists i \in \mathcal{N}$ for which $|\mathcal{A}_i| < K_i$. We show that $\bar{C}(\mathcal{A} \cup (o, i)\}) - \bar{C}(\mathcal{A}) \geq \bar{C}(\mathcal{A}' \cup (o, i)\}) - \bar{C}(\mathcal{A}')$ for all $\mathcal{A} \subseteq \mathcal{A}' \in \mathfrak{A}$ and $(o, i) \in \mathcal{E}_i \setminus \mathcal{A}'_i$. We now distinguish between two cases. If $\exists j$ such that $(o, j) \in \mathcal{A}'_j \setminus \mathcal{A}_j$ then the difference $\Delta C(\mathcal{A})$ is

$$\Delta C(\mathcal{A}) = c_0 \sum_{k \in \{\mathcal{N} | \mathsf{LCA}(k,i) = n_0\}} w_k^o + (c_0 + d_{i,n_0}) \sum_{k \in \mathcal{N}_i} w_k^o + \sum_{t=1}^{l(i)-1} (c_0 + d_{\mathcal{P}^t(i), n_0}) \sum_{k \in \{\mathcal{N}_{\mathcal{P}^t(i)} \setminus \mathcal{N}_{\mathcal{P}^{t-1}(i)}\}} w_k^o,$$

and the difference $\Delta C(\mathcal{A}')$ is

$$\Delta C(\mathcal{A}') = (c_0 + d_{i,\mathsf{LCA}(j,i)}) \sum_{k \in \mathcal{N}_i} w_k^o + \sum_{t=1}^{l(i)-l(\mathsf{LCA}(j,i))-1} (c_0 + d_{\mathcal{P}^t(i), \mathsf{LCA}(j,i)}) \sum_{k \in \{\mathcal{N}_{\mathcal{P}^t(i)} \setminus \mathcal{N}_{\mathcal{P}^{t-1}(i)}\}} w_k^o.$$

Since $l(\mathsf{LCA}(j, i)) \geq 0$, it holds that $\Delta C(\mathcal{A}) > \Delta C(\mathcal{A}')$. Otherwise, if $\exists j$ such that $(o, j) \in \mathcal{A}_j$ or if $\nexists j$ such that $(o, j) \in \mathcal{A}'_j$ then $\Delta C(\mathcal{A}) = \Delta C(\mathcal{A}')$. The result then follows by applying Lemma 4 to $C(\emptyset) - C(\mathcal{A})$. □

Observe that the approximation ratio is bounded for arbitrary traversals of the graph. Nonetheless, a pre-order depth-first traversal allows for a distributed implementation of *DFG* with a communication overhead of $\sum_{k=1}^{|\mathcal{N}|} (|\mathcal{N}| - k) K_{i_k}$. This communication overhead is comparable to that of a centralized implementation, as it requires collection of all demands and needs to communicate the placement decisions, but allows for the distribution of the computations in the network.

It is important to note that *DFG* differs from the distributed global greedy (*DGG*) algorithm used in [8,16]. *DGG* chooses in every iteration the fictitious item (i, o) that maximizes the cost saving, and thus has computational complexity $O(|\mathcal{N}|^2 \max_i K_i |\mathcal{O}| \log(|\mathcal{N}||\mathcal{O}|))$. In contrast, *DFG* populates the caches of the nodes one-by-one, and thus has computational complexity $O(|\mathcal{N}| \max_{i_k} K_{i_k} |\mathcal{O}| \log(|\mathcal{O}|))$. Unfortunately, *DFG* requires global information at every node of the network, which may cause significant communication overhead. We therefore turn to distributed approximation algorithms based on limited information.

5. Distributed algorithms under limited information

In what follows we propose two distributed algorithms that do not need global information about the demands and the network topology.

5.1. Local Greedy Swapping (LGS) Algorithm

The first algorithm, called Local Greedy Swapping (*LGS*), allows nodes to swap objects with their parents based on the aggregate demands and the object placements in their *descendants only*. Denoting the placement at node i at iteration k by $\mathcal{A}_i(k)$, the *LGS* algorithm starts with an arbitrary initial object placement $\mathcal{A}(0) = (\mathcal{A}_i(0))_{i \in \mathcal{N}}$ in which each node $i \in \mathcal{N}$ stores K_i objects. At iteration k the algorithm computes the set of beneficial swaps $T(\mathcal{A}(k)) \subset \mathcal{N} \times \mathcal{O}^2$. A triplet $(i, o, p) \in T(\mathcal{A}(k))$ corresponds to that node i can swap object $p \in \mathcal{A}_i(k)$ with object $o \in \mathcal{A}_{\mathcal{P}(i)}(k)$ at its parent node $\mathcal{P}(i)$. For $i = n_0$, i.e., $(n_0, o, p) \in T(\mathcal{A}(k))$ the root node n_0 can evict object p and can fetch object o through the Backbone. The set of implemented swaps $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$ is then chosen to increase the local cost saving greedily.

To define the set of beneficial swaps $T(\mathcal{A})$, let us introduce the function $I(i, o, p)$ to indicate whether the aggregate demand at node i and its descendants $\mathcal{D}(i)$ is higher for object o than for object p,

$$I(i, o, p) = \begin{cases} 1, & \text{if} \sum_{j \in \mathcal{N}_i} (w_j^o - w_j^p) > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Given a placement $\mathcal{A}$, node i might be interested in swapping object $p \in \mathcal{A}_i$ with object $o \in \mathcal{A}_{\mathcal{P}(i)}$ at its parent if $I(i, o, p) = 1$ or if p is available in the cache of node i's descendants $\mathcal{D}(i)$, i.e., $p \in \mathcal{R}_i \setminus \mathcal{A}_i$, as in this case node i can retrieve object p at no cost

LGS Algorithm

$k \leftarrow 0$
while $|T(\mathcal{A}(k))| > 0$ **do**
  $\mathcal{A}(k+1) \leftarrow \mathcal{A}(k)$
  for each $(i, o, p) \in S(\mathcal{A}(k))$ **do**
    $\mathcal{A}_i(k+1) \leftarrow (\mathcal{A}_i(k) \cup \{o\} \setminus \{p\})$
    if $p \notin \mathcal{A}_{\mathcal{P}(i)}(k)$ **then**
      $\mathcal{A}_{\mathcal{P}(i)}(k+1) \leftarrow (\mathcal{A}_{\mathcal{P}(i)}(k) \cup \{p\} \setminus \{o\})$
    end if
  end for
  $k \leftarrow k+1$
end while

Fig. 3. Pseudo-code of the *LGS* algorithm.

even if $p \notin \mathcal{A}_i$. We use this observation to define the set of node-object triplets that would be beneficial for swapping at placement $\mathcal{A}$,

$$T(\mathcal{A}) = \{(i, o, p) | i \in \mathcal{N}, o \in \mathcal{A}_{\mathcal{P}(i)} \setminus \mathcal{R}_i, p \in \mathcal{A}_i, \big((p \in \mathcal{R}_i \setminus \mathcal{A}_i) \vee (p \notin \mathcal{R}_i \setminus \mathcal{A}_i \wedge I(i, o, p) = 1)\big)\}.$$

The algorithm terminates at iteration k if the set $T(\mathcal{A}(k))$ is empty. Note that the complexity of the algorithm is low at each iteration, as the minimization is performed over a small subset of objects held by a node and its children. The pseudo-code of *LGS* is shown in Fig. 3.

To complete the definition of the algorithm, we now describe a greedy algorithm to choose the set $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$ at iteration k. Given $T(\mathcal{A}(k))$, we choose a node i_k with a child that would like to swap (i.e., $\exists j \in \mathcal{C}(i_k)$ and $(j, o, p) \in T(\mathcal{A}(k))$). Given i_k we select the best swap (j_k, o_k, p_k) of its children, i.e., the one that maximizes the local cost saving in the subtree $\mathcal{N}_{i_k}$ (swap with parent), and we then allow every child node $j \in \mathcal{C}(i_k)$ to insert into its cache objects $o \in \mathcal{A}_{i_k}(k) \cup \{p_k\}$, if doing so would increase the local cost saving (copy from parent). The algorithm is shown in Algorithm 1.

Algorithm 1 $S(\mathcal{A}(k)) = \text{populateS}(\mathcal{A}(k), i_k)$.

1: Select the best swapping opportunity at the childrenof i_k,

$$(j_k, o_k, p_k) \leftarrow \underset{\{(j,o,p) \in T(\mathcal{A}(k)) | j \in \mathcal{C}(i_k)\}}{\arg\max} \sum_{n \in \mathcal{N}_j} c_{i,j}(w^o_n - w^p_n)$$

$$S(\mathcal{A}(k)) \leftarrow (j_k, o_k, p_k)$$

2: Further decrease the cost function through allowing nodes in$\mathcal{C}(i_k)$ to insert objects available at$\{\mathcal{A}_{i_k}(k) \cup \{p_k\}\}$.

$\text{PE}_j \leftarrow (\mathcal{A}_{i_k}(k) \cup \{p_k\}) \cap \mathcal{A}_j(k)$
$\text{PO}_j \leftarrow (\mathcal{A}_{i_k}(k) \cup \{p_k\}) \setminus \mathcal{R}_j(k)$
while $\exists (j, o, p)$ s.t.$o \in \text{PO}_j$ and$p \in \text{PE}_j$ and

$$(p \in \mathcal{R}_j \setminus \mathcal{A}_j(k)) \vee (p \notin \{\mathcal{R}_j \setminus \mathcal{A}_j(k)\} \wedge I(j, o, p) = 1)) \quad (10)$$

do
  $S(\mathcal{A}(k)) \leftarrow S(\mathcal{A}(k)) \cup \{(j, o, p)\}$
  $\text{PE}_j \leftarrow \text{PE}_j \setminus \{p\}$
  $\text{PO}_j \leftarrow \text{PO}_j \setminus \{o\}$
end while

Lemma 5. *The global cost C decreases strictly at every swap.*

Proof. Consider $(i, o, p) \in S(\mathcal{A}(k))$ at iteration k. For every node $j \in \mathcal{N} \setminus \mathcal{N}_i$ it holds $d_{j,i} = d_{j,\mathcal{P}(i)} + c_{i,\mathcal{P}(i)} = d_{j,\mathcal{P}(i)}$, hence $d_j(o, \mathcal{A}(k+1)) = d_j(o, \mathcal{A}(k))$ and $d_j(p, \mathcal{A}(k+1)) = d_j(p, \mathcal{A}(k))$. Consequently, $C_j(\mathcal{A}(k+1)) = C_j(\mathcal{A}(k))$ for all $j \in \mathcal{N} \setminus \mathcal{N}_i$.

Consider now node $j \in \mathcal{N}_i$. Since $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$, it follows that $o \notin \mathcal{R}_i(k)$ and $o \in \mathcal{A}_{\mathcal{P}(i)}(k)$. Hence $d_j(o, \mathcal{A}(k)) = d_{j,i} + c_{\mathcal{P}(i),i}$, $d_j(o, \mathcal{A}(k+1)) = d_{j,i}$, and the difference in the cost $\Delta C(k+1)$ before and after the swap is

$$\begin{aligned}\Delta C(k+1) &= \sum_{j \in \mathcal{N}_i} \big[C_j(\mathcal{A}(k+1)) - C_j(\mathcal{A}(k))\big] \\ &= \sum_{j \in \mathcal{N}_i} \big[w^o_j d_{j,i} - w^o_j (d_{j,i} + c_{\mathcal{P}(i),i}) + w^p_j d_j(p, \mathcal{A}(k+1)) \\ &\qquad - w^p_j d_j(p, \mathcal{A}(k))\big] \\ &= \sum_{j \in \mathcal{N}_i} \big[-w^o_j c_{\mathcal{P}(i),i} + w^p_j \big(d_j(p, \mathcal{A}(k+1)) - d_j(p, \mathcal{A}(k))\big)\big].\end{aligned}$$

Similarly, $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$ implies that $p \in \mathcal{A}_i(k)$, hence $d_j(p, \mathcal{A}(k)) \leq d_{j,i}$. We now distinguish between two cases. If $d_j(p, \mathcal{A}(k)) < d_{j,i}$, then $d_j(p, \mathcal{A}(k+1)) = d_j(p, \mathcal{A}(k))$, which implies that $\Delta C(k+1) < 0$. Otherwise, if $d_j(p, \mathcal{A}(k)) = d_{j,i}$, then $d_j(p, \mathcal{A}(k+1)) = d_{j,i} + c_{\mathcal{P}(i),i}$. Since $I(i, o, p) = 1$, then $\Delta C(k+1) = c_{\mathcal{P}(i),i} \sum_{j \in \mathcal{N}_i} (w^p_j - w^o_j) < 0$. This proves the lemma. □

We can use this result to show that the algorithm terminates after a finite number of iterations.

Theorem 5. *The* LGS *algorithm terminates after a finite number of iterations.*

Proof. Consider iteration k of the *LGS* algorithm. Call $s(\mathcal{A})$ the object placement that results from applying swap $s = (j, o, p)$ to placement $\mathcal{A}$. It follows from the proof of Lemma 5 that for any swap $s = (j, o, p) \in S(\mathcal{A}(k))$ and every node $l \in \mathcal{N} \setminus \mathcal{N}_j$, it holds $\{\mathcal{R}_j(\mathcal{A}(k)) \cup \mathcal{A}_{i_k}(k)\} = \{\mathcal{R}_j(s(\mathcal{A}(k))) \cup s(\mathcal{A}_{i_k}(k))\}$ and hence $C_l(s(\mathcal{A}(k))) = C_l(\mathcal{A}(k))$. Since for every $j, l \in \mathcal{C}(i_k)$, $j \neq l$ it holds $l \notin \mathcal{N}_j$, we can consider each node $j \in \mathcal{C}(i_k)$ separately.

Consider swap $s = (j, o, p) \in S(\mathcal{A}(k))$. It follows from (9) that either $p \in \mathcal{R}_j \setminus \mathcal{A}_j(k)$ or $I(j, o, p) = 1$. Therefore, from the proof of Lemma 5, it follows that $C_l(s(\mathcal{A}(k))) \leq C_l(\mathcal{A}(k))$ for all $l \in \mathcal{N}_j$. In particular, for swap $s_k = (j_k, o_k, p_k) \in T(\mathcal{A}(k))$, it holds that $I(j_k, o_k, p_k) = 1$, which implies $C_{j_k}(s_k(\mathcal{A}(k))) < C_{j_k}(\mathcal{A}(k))$.

Since $\times_{i \in \mathcal{N}} \mathfrak{A}_i$ is a finite set, $C(\mathcal{A}(k))$ can not decrease indefinitely and the *LGS* algorithm terminates after a finite number of iterations. □

Besides being guaranteed to converge starting from an arbitrary initial placement, a nice property of *LGS* is that if started from an optimal placement, the algorithm is stable in the sense that it does not make any changes, as we show next.

Corollary 1. *An optimal content placement $\bar{\mathcal{A}}$ is stable under the* LGS *algorithm.*

Proof. From Lemma 5 and Theorem 5 it follows that $C(\mathcal{A}(k+1)) < C(\mathcal{A}(k))$ for any swap $s \in S(\mathcal{A}(k))$. By definition $\nexists \mathcal{A}' \in \times_{i \in \mathcal{N}} \mathfrak{A}_i$ s.t. $C(\mathcal{A}') < C(\bar{\mathcal{A}})$, hence the result. □

For simplicity, we restricted ourselves to a single i_k per iteration when defining $S(\mathcal{A}(k))$, but the above results hold for any set of nodes that are not each others' descendants, hence the algorithm can be executed in parallel.

5.2. h-Push Down algorithm

In the *LGS* algorithm, every node i swaps objects based on the information about the object placement and the aggregate demand for objects at its descendants $\mathcal{D}(i)$. In the following we provide a distributed algorithm that allows node i to leverage additional information on placements and on aggregate demands for objects. In the *h-Push Down* algorithm, every node i has information about the placement $\mathcal{A}_{\mathcal{N}_j}$ and about the object demands w^o_k, $k \in \mathcal{N}_j$, for

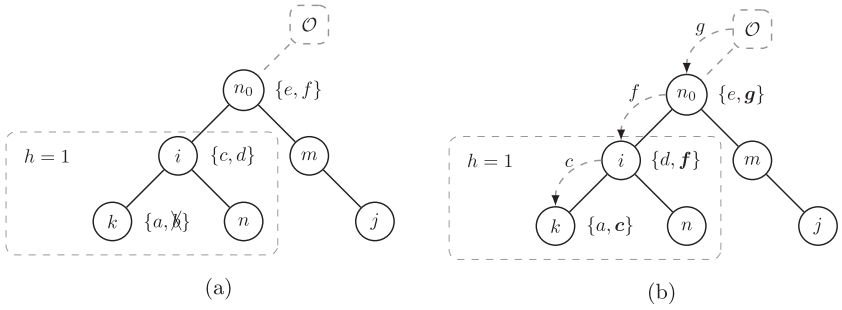

Fig. 4. Illustration of first 4(a) and second 4(b) steps of the *h-Push Down* algorithm. Object b is evicted from $\mathcal{A}_k$ during the first step. A *PushDown* move is performed during the second step.

every ancestor j that lies within its information horizon h, i.e., for $j = \mathcal{P}^l(i)$ for $0 \le l \le h$.

The algorithm starts with an object placement $(\mathcal{A}_i(0))_{i\in\mathcal{N}}$ in which each node $i \in \mathcal{N}$ stores K_i objects that have the highest aggregated demands in the subnetwork $\mathcal{N}_i$ and that are not available in the cache of node i's descendants $\mathcal{D}(i)$. An iteration of the algorithm consists of two steps. The first step is an eviction operation at some node i. The second step is a *PushDown* move, a sequence of placement updates such that at each update one object $o \in \mathcal{A}_{\mathcal{P}^l(i)}$ is moved from $\mathcal{P}^l(i)$ to $\mathcal{P}^{l-1}(i)$, for $l = 1, 2, \ldots, k$, where $\mathcal{P}^{k-1}(i) = n_0$. In the last update of the *PushDown* move, i.e., $l = k$, one object is retrieved through the Backbone and stored at the root node $\mathcal{P}^{k-1}(i) = n_0$. The two steps of the *h-Push Down* algorithm are illustrated in Fig. 4. The pseudo-code of the *PushDown* move of the *h-Push Down* algorithm is shown in Algorithm 2. Note

Algorithm 2 $\mathcal{A}' = \text{PushDown}(i, \mathcal{A})$.

$t \leftarrow 0$
$\mathcal{A}^0 \leftarrow \mathcal{A}$
do
  $n \leftarrow \mathcal{P}^t(i)$
  $o^t \leftarrow \arg\min_{o\in\mathcal{A}^t_{\mathcal{P}(n)}} C(\mathcal{A}^t_n \cup \{o\}, \mathcal{A}^t_{-n})$
  $\mathcal{A}^{t+1}_n \leftarrow \mathcal{A}^t_n \cup \{o^t\}$
  $\mathcal{A}^{t+1}_{\mathcal{P}(n)} \leftarrow \mathcal{A}^t_{\mathcal{P}(n)} \setminus \{o^t\}$
  $t \leftarrow t + 1$
while $n \ne n_0$
return $\mathcal{A}'$

that the complexity of the algorithm at each iteration is limited by the number of objects held in the nodes and the number of levels in the cache hierarchy.

Central to the algorithm is the LCA of node i and the node from which node i would retrieve object o in the placement $(\varnothing, \mathcal{A}_{-i})$, i.e., if it had no objects cached,

$$P_i^o(\mathcal{A}_{-i}) \triangleq \text{LCA}\Big(i, \operatorname*{argmin}_{\{j\in\mathcal{N}\setminus\{i\}|o\in\mathcal{A}_j\}} d_{i,j}\Big). \tag{10}$$

Similarly, we define $P_i^o(\mathcal{A})$ for placement $\mathcal{A}$, i.e., $P_i^o(\mathcal{A}) = i$ if $o \in \mathcal{A}_i$, otherwise $P_i^o(\mathcal{A}) = P_i^o(\mathcal{A}_{-i})$.

The following lemma shows an important property of the *PushDown* move.

Lemma 6. *A move* $\mathcal{A}' = \text{PushDown}(i, \mathcal{A})$ *always decreases the global cost by*

$$\Delta C_{\text{PD}}(i, \mathcal{A}) \triangleq C(\mathcal{A}) - C(\mathcal{A}') = \sum_{t=0}^{l(i)} c_{\mathcal{P}^{t+1}(i),\mathcal{P}^t(i)} \sum_{j\in T(t)} w_j^{o^t},$$

where $T(t) = \{j \in \mathcal{N}_{\mathcal{P}^t(i)} | P_j^{o^t}(\mathcal{A}) = \mathcal{P}^{t+1}(i)\}$.

h-Push Down Algorithm

$k \leftarrow 0$
$\text{Z}^0 \leftarrow \{i \in \mathcal{N}$ such that $|\text{Z}_i(\mathcal{A}(0))| > 0\}$
$\mathcal{A} \leftarrow \mathcal{A}(0)$
while $|\text{Z}^k| > 0$ **do**
  Pick $i_k \in \text{Z}^k$
  Compute the least cost eviction: $o^k \leftarrow \arg\min_{o\in\text{Z}_{i_k}} |\Delta C_{\text{EV}}(i_k, o, \mathcal{A})|$
  Compute $\Delta C^h_{\text{PD}}(i_k, \mathcal{A}) \triangleq \sum_{t=0}^{\min(h,l(i_k))} c_{\mathcal{P}^{t+1}(i_k),\mathcal{P}^t(i_k)} \sum_{j\in T(t)} w_j^{o^t}$,
  if $\Delta C^h_{\text{PD}}(i_k, \mathcal{A}) + \Delta C_{\text{EV}}(i_k, o^k, \mathcal{A}) > 0$ **then**
    $\mathcal{A}(k+1) \leftarrow \text{PushDown}(i, (\mathcal{A}_{i_k} \setminus \{o^k\}, \mathcal{A}_{-i_k}))$
    $k \leftarrow k + 1$
    $\mathcal{A} \leftarrow \mathcal{A}(k)$
    $\text{Z}^k \leftarrow \{i \in \mathcal{N}$ such that $|\text{Z}_i(\mathcal{A}(k))| > 0\}$
  else
    $\text{Z}^k \leftarrow \text{Z}^k \setminus \{i_k\}$
  end if
end while

Fig. 5. Pseudo code of the *h-Push Down* algorithm.

Proof. Consider iteration t of move $\mathcal{A}' = \text{PushDown}(i, \mathcal{A})$. Since $c_{n,\mathcal{P}(n)}$=0, for all $j\in\mathcal{N}\setminus\mathcal{N}_n$ it holds that $d_j(o^t, \mathcal{A}^t)$=$d_j(o^t, \mathcal{A}^{t+1})$. For nodes $j \in \mathcal{N}_n$ we need to distinguish between two cases. If $P_j^{o^t}(\mathcal{A}^t) \ne \mathcal{P}(n)$, then $P_j^{o^{t+1}}(\mathcal{A})$=$P_j^{o^t}(\mathcal{A}^t)$ and $d_j(o^t, \mathcal{A}^t)$=$d_j(o^t, \mathcal{A}^{t+1})$. It follows that, if $j \notin T(t)$, then $C_j^o(\mathcal{A}^t) - C_j^o(\mathcal{A}^{t+1}) = 0$. Otherwise, $P_j^{o^t}(\mathcal{A}^t)$=$\mathcal{P}(n)$ implies $P_j^{o^t}(\mathcal{A}^{t+1})$=$n$, and hence $C_j^o(\mathcal{A}^t) - C_j^o(\mathcal{A}^{t+1}) = w_j^{o^t} c_{\mathcal{P}(n),n}$. By summing over all the $l(i)$ iterations of the *PushDown* move, we prove the lemma. □

In the *h-Push Down* algorithm, a node i can only initiate a move, and therefore evict one object o, if o is cached at node i's descendants or if $P_i^o(\mathcal{A}_{-i})$ lies within node i's information horizon, i.e., $P_i^o(\mathcal{A}_{-i}) = \mathcal{P}^l(i)$ for some $0 < l \le h$. We use $\text{Z}_i(\mathcal{A})$ to denote the set of objects that are candidate for eviction at node i under placement $\mathcal{A}$, i.e.,

$$\text{Z}_i(\mathcal{A}) = \{o \in \mathcal{A}_i | P_i^o(-\mathcal{A}) \in \textstyle\bigcup_{l=0}^h \mathcal{P}^l(i) \vee o \in \bigcup_{j\in\mathcal{D}(i)} \mathcal{A}_j\}.$$

We use $\Delta C_{\text{EV}}(i, o, \mathcal{A}) \triangleq C(\mathcal{A}) - C(\mathcal{A}_i \setminus \{o\}, \mathcal{A}_{-i})$ to denote the change in the global cost caused by the eviction of object o at node i. Observe that $\Delta C_{\text{EV}}(i, o, \mathcal{A}) \le 0$.

The pseudo-code of the *h-Push Down* algorithm is shown in Fig. 5. We start with showing that the algorithm terminates in a finite number of iterations.

Theorem 6. *The h-Push Down algorithm terminates after a finite number of iterations.*

Proof. We prove the theorem by showing that the global cost $C(\mathcal{A})$ decreases at every iteration of the h-Push Down algorithm. From Lemma 6 it follows that

$$\Delta C_{\text{PD}}(i_k, (\mathcal{A}(k)_{i_k} \setminus \{o^k\}, \mathcal{A}(k)_{-i_k})) \ge \Delta C^h_{\text{PD}}(i_k, \mathcal{A}(k)). \tag{11}$$

By definition, the variation of the global cost at iteration k can be written as the sum of the variation due

to the eviction and the variation due to *PushDown* move, i.e., $\Delta C_{\text{EV}}(i_k, o^k, \mathcal{A}(k)) + \Delta C_{\text{PD}}(i_k, (\mathcal{A}(k)_{i_k} \setminus \{o^k\}, \mathcal{A}(k)_{-i_k})) = C(\mathcal{A}(k)) - C(\mathcal{A}(k+1))$. The proof of the theorem follows from (11). □

Furthermore, similar to *LGS*, the algorithm does not make any changes to an optimal placement, as shown next.

Corollary 2. *The optimal content placement $\bar{\mathcal{A}}$ is stable with respect to the h-Push Down algorithm.*

Proof. The proof is analogous to the proof of Corollary 1. □

Observe that the computation of $\Delta C^h_{\text{PD}}(i_k, \mathcal{A}(k))$ depends only on the object demands and the placements at the nodes in the set $\mathcal{N}_{\mathcal{P}^h(i_k)}$. Furthermore, in order to compute $\Delta C_{\text{EV}}(i_k, o^k, \mathcal{A}(k))$, node i_k only requires information about placements and demands in the subnetwork $\mathcal{N}_{P^{o^k}_i(\mathcal{A}_{-i}(k))}$, which lies within node i_k's information horizon h.

6. Numerical results

We use simulations to evaluate the approximation ratio and the convergence rate of the proposed algorithms. To generate backhaul topologies, we use a *Random grid* model, in which $|\mathcal{N}|$ nodes are randomly placed on a $|\mathcal{N}| \times |\mathcal{N}|$ regular grid. The random placement of nodes on a grid captures the potentially uneven spatial distribution of base stations in urban mobile deployments. Given the node placement, we build a weighted complete graph by setting the weight on edge (i, j) equal to the Euclidean distance between nodes i and j, computed based on their coordinates. We then run *Kruskal*'s algorithm [17] on the resulting weighted complete graph to compute a minimum spanning tree to obtain the topology $\mathcal{G}$. We consider two different cost models. In the *distance* cost model the edge costs $c_{\mathcal{P}(i),i}$ are equal to the weights used for generating the tree. In the *descendants* cost model the edge costs $c_{\mathcal{P}(i),i}$ are proportional to the size of the subtree $\mathcal{N}_i$, as larger subnetworks likely lead to higher peak loads and less available bandwidth on the links serving them.

The object demands w^o_i at the nodes follow Zipf's law. For the ranking of the object demands at the nodes we consider two models. In the case of *homogeneous demands*, the object demands have the same rank at all nodes. In the case of *heterogeneous demands*, the demand w^o_i for object o at node i is ranked as in the case of *homogeneous demands* with probability 0.5. With probability 0.5, the rank of w^o_i is picked uniformly at random. The results shown are the averages of 500 simulations, and the error bars show 95% confidence intervals.

As a baseline for comparison, we use a selfish distributed algorithm called *Distributed Local-Greedy* (DLG), which is based on global information about the object demands and placements at every node of the network. Following the *DLG* algorithm, starting from a randomly chosen allocation, at iteration k node i_k optimizes its placement of objects $\mathcal{A}_{i_k}(k)$ so as to minimize the cost for serving the requests from the local cell site, given the placement of objects $\mathcal{A}_{-i_k}(k)$ at the other nodes in the network [18–20]. As there is no guarantee that the *DLG* algorithm terminates [20], we run it for $|\mathcal{N}|$ iterations and we set $i_k = k$. Note that although *DLG* is seemingly similar to *DFG*, *DFG* minimizes the global cost based on global information, while *DLG* minimizes the local cost based on global information, hence it is algorithmically simpler.

6.1. Performance of distributed algorithms

In order to compare the performance of the proposed algorithms, as well as to evaluate the tightness of the analytical

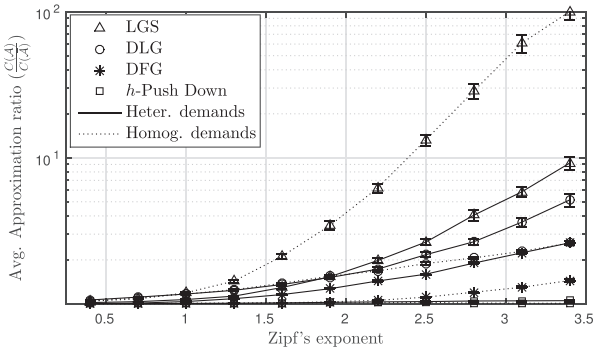

Fig. 6. Average approximation ratio vs. Zipf exponent for the *LGS*, *DLG*, *DFG*, and *h-Push Down* algorithms. *Heterogeneous* and *homogeneous demands*, $|\mathcal{O}| = 100$, $|\mathcal{N}| = 20$, $K_i = 2$.

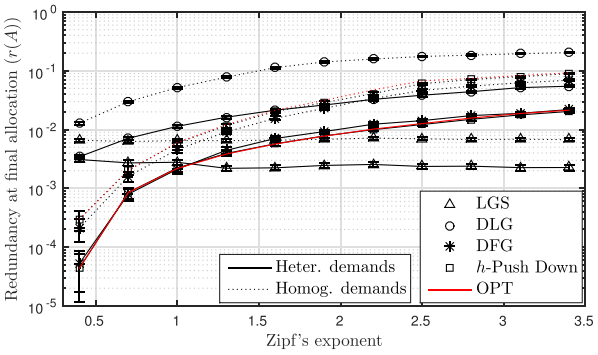

Fig. 7. Redundancy $r(\mathcal{A})$ vs. Zipf exponent for *LGS*, *DLG*, *DFG*, and *h-Push Down* and for the optimal placement. *Heterogeneous* and *homogeneous demands*, $|\mathcal{O}| = 100$, $|\mathcal{N}| = 20$, $K_i = 2$.

results, we computed the optimal placement $\bar{\mathcal{A}}$ and the cost-approximation ratio $C(\mathcal{A})/C(\bar{\mathcal{A}})$ for each algorithm. To make the computation of the optimal placement feasible, we considered a relatively small scenario with $|\mathcal{N}| = 20$, $|\mathcal{O}| = 100$ and $K_i = 2$ for all $i \in \mathcal{N}$. Fig. 6 shows the cost-approximation ratio as a function of the Zipf exponent of the object demand distribution for *LGS*, *DLG*, *DFG* and for the *h-Push Down* algorithm with global information, i.e., for $h = \max_{i \in \mathcal{N}} l(i)$, for the *descendants* cost model.

The most salient feature of the figure is that the approximation ratio of the *LGS* algorithm increases exponentially with the Zipf exponent at a fairly high rate. The reason for the poor performance in the case of homogeneous demands is that the *LGS* algorithm populates the set $S(\mathcal{A}(k))$ exclusively based on the rankings of the object demands and not based on their values. As the Zipf exponent increases, the demand of the most popular content increases and the optimal solution might differ significantly from the allocation reached by the *LGS* algorithm. In order to validate this hypothesis, we computed the redundancy of a placement $\mathcal{A}$ using the index

$$r(\mathcal{A}) = \frac{\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N} \setminus \{i\}} \left(1 - \frac{\min(K_i, K_j) - |\mathcal{A}_i \cap \mathcal{A}_j|}{\min(K_i, K_j)}\right)}{|\mathcal{N}|(|\mathcal{N}| - 1)}. \tag{12}$$

Intuitively, $r(\mathcal{A})$ is the average ratio of objects common between all pairs of placements $\mathcal{A}_i$ and $\mathcal{A}_j$. In Fig. 7 we plot the average $r(\mathcal{A})$ index of the final placements reached by the algorithms, for the same scenario as Fig. 6. The figure confirms that as the Zipf exponent increases, the *LGS* algorithm fails to introduce redundancy, which explains its poor performance.

Comparing the performance of *h-Push Down* to that of *DFG* we observe that *h-Push Down* (with global information) performs better than *DFG*, which is also reflected by the redundancy index, which is very close to the optimal (cf. Fig. 6). Finally, it is note-

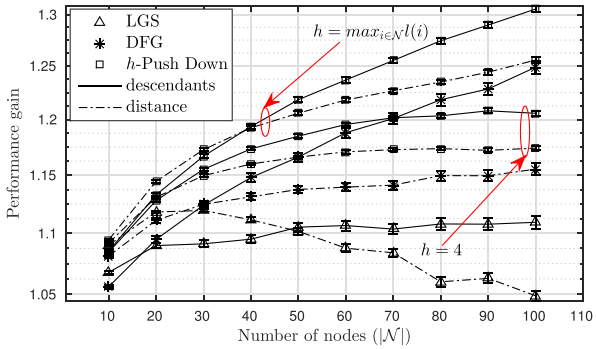

Fig. 8. Performance gain vs number of nodes $|\mathcal{N}|$ for the *h-Push Down*, *LGS* and *DFG* algorithms on the Random grid model with *descendants* and *distance* cost model, $|\mathcal{O}| = 5000$, $K_i = 20$.

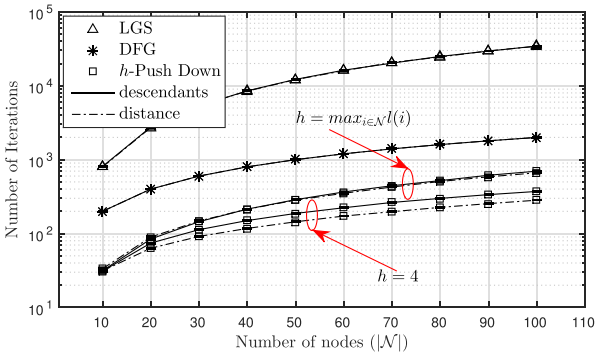

Fig. 9. Number of iterations vs number of nodes $|\mathcal{N}|$ for the *h-Push Down*, *LGS* and *DFG* algorithms on the Random grid model with *descendants* and *distance* cost model, $|\mathcal{O}| = 5000$, $K_i = 20$.

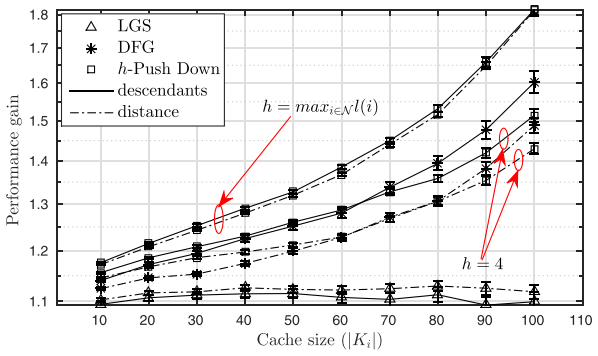

Fig. 10. Performance gain vs. cache size K_i for *h-Push Down*, *LGS* and *DFG* on the Random grid model with *descendants* and *distance* cost model. Results for $|\mathcal{O}| = 5000$, $|\mathcal{N}| = 50$.

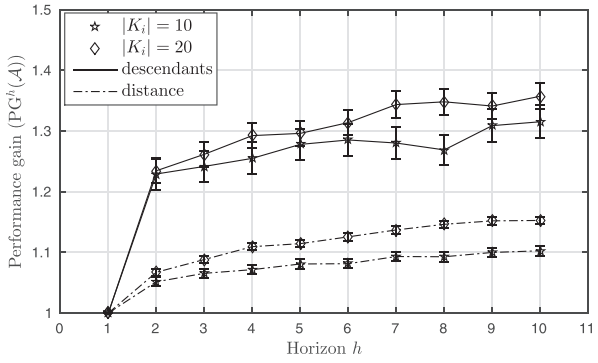

Fig. 11. Performance gain vs. horizon h for cache sizes $K_i \in \{10, 20\}$ on the Random grid model with *descendants* and *distance* cost models. Results for $|\mathcal{O}| = 5000$ and $|\mathcal{N}| = 100$.

worthy that the *DLG* algorithm, which corresponds to selfish local optimization, fails to achieve performance close to the optimal, despite the availability of global information.

In order to evaluate the performance of the algorithms for larger scenarios, in the following we use the *DLG* algorithm as a baseline for comparison, as it is prohibitive to compute the optimal placement. Recall that the *DLG* algorithm optimizes the placement of objects in order to minimize the local cost, which would make it a reasonable simple choice in absence of more elaborate distributed algorithms.

To capture the performance of the algorithms relative to *DLG* we define the performance gain of an algorithm as the ratio between the cost of the placement reached by the *DLG* algorithm and the cost of the placement reached by the algorithm. It follows from (1) that the performance gain is also a measure of the increased hit rate achieved by the algorithm relative to *DLG*. Fig. 8 shows the performance gain for the *LGS*, *DFG* and *h-Push Down* (for two values of the information horizon h) algorithms, as a function of the number of nodes for $K_i = 20$. The results are shown for *heterogeneous demands* using a Zipf exponent of 1, for the two cost models. We observe that the performance gain for the *DFG* and the *h-Push Down* algorithms increases with the number of nodes. Furthermore, the figure shows that *h-Push Down* outperforms *DFG* (i.e., it is close to optimal) for both values of the horizon h. The figure also shows that *LGS* performs just slightly better than *DLG*, with a decreasing gain as the network size increases.

Fig. 9 shows the number of iterations needed to compute the final object placement corresponding to the results shown in Fig. 8. Recall that the *DFG* algorithm starts with an empty allocation and terminates in $\sum_{i\in\mathcal{N}} K_i$ iterations, and can thus be used a baseline in terms of convergence. The results show that *LGS* performs worst, while *h-Push Down* for $h = 4$ requires almost an order of magnitude less iterations to terminate than *DFG*.

Fig. 10 shows the performance gain as a function of the cache sizes for $|\mathcal{N}| = 50$. The figure shows that for higher cache sizes the performance gain of the *DFG* and *h-Push Down* algorithms over the *DLG* algorithm increases faster than exponentially. In the case of global information, the *h-Push Down* algorithm outperforms the *DFG* algorithm, while in the case of non-global information, i.e., for $h = 4$, it achieves performance close to the *DFG* algorithm. Furthermore, the performance gap between the *h-Push Down* algorithm with global and non-global information increases for higher cache sizes. The figure also confirms that the *LGS* and *DLG* algorithms achieve a comparable total cost.

6.2. Impact of the information horizon (h)

Finally, we evaluate the impact of the information horizon h on the performance of *h*-Push Down. We define the performance gain $PG^h(\mathcal{A})$ for horizon h as the ratio between the cost of the placement $\mathcal{A}^1$ reached by the *h*-Push Down algorithm with $h = 1$ and the cost of the placement $\mathcal{A}^h$ reached with horizon h, i.e. $PG^h(\mathcal{A}) = \frac{C(\mathcal{A}^1)}{C(\mathcal{A}^h)}$.

Figs. 11 and 12 show the performance gain $PG^h(\mathcal{A})$ and the number of iterations, respectively, for the *h-Push Down* algorithm as a function of the information horizon h for $|\mathcal{N}| = 100$ and two different cache sizes K_i. We plot the performance gain $PG^h(\mathcal{A})$ for the same cost and object demands models as in Figs. 8 and 10. We observe that the performance gain increases with a decreasing marginal gain in h, making the algorithm perform fairly well with limited available information (low h). Furthermore, the same observation holds for the convergence time, hence a moderate value

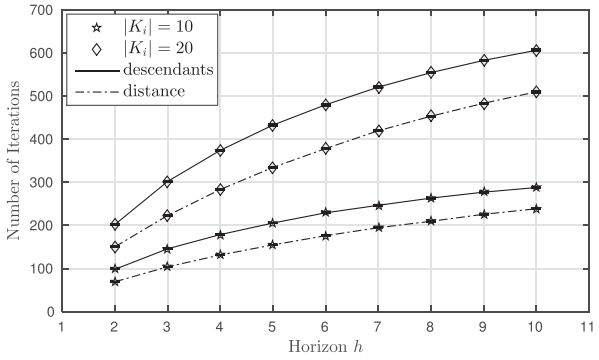

Fig. 12. Number of iterations vs horizon h for two values of cache sizes $K_i \in \{10, 20\}$ on the Manhattan graph with *descendants* and *distance* cost models. Results for $|\mathcal{O}| = 5000$ and $|\mathcal{N}| = 100$.

of h provides a good trade-off between performance and convergence time. Fig. 11 also shows that as the horizon h increases, the performance gain increases more in the case of the *descendants* cost model than in the case of the *distance* cost model. The reason is that as the horizon h increases, the nodes have access to the cost of edges between nodes at lower levels of the tree (i.e., closer to the root), which in the case of the *descendants* cost model are the edges with highest cost, and thus they have a higher impact on the total cost.

7. Related work

Closest to ours are recent works on content placement in networks [8,10,14,16,21–24]. The authors in [21] compared cache architectures in terms of latency, bandwidth usage and cache load, and evaluated cache placement policies. The authors in [22] provide an algorithm for computing the optimal placement in a hierarchical network by reducing the content placement problem to a minimum-cost flow problem. Motivated by the computational complexity of the problem, they design a distributed amortizing algorithm that achieves a constant factor approximation. The model considered in [22] is based on the ultrametric cost model introduced in [9], which differs from our model on the assumption of symmetric costs between nodes. The authors in [8] give insights in the structure of the optimal placement in a regular two level hierarchical network, and they develop a greedy distributed 2-approximation algorithm. The authors in [16] consider a hybrid network with in-network caching and they propose a $(1 - 1/e)$-approximation greedy algorithm. A more generic cost model was considered in [10], where the authors develop a 10-approximation algorithm by rounding the optimal solution of the LP-relaxation of the problem. Poularakis et al. [24] proposed a set of centralized, polynomial time algorithms with approximation guarantees, for the joint problem of request routing and content replication under strict bandwidth constraints at the storage sites. Poularakis and Tassiulas [14] considered a hierarchical cache network in which requests originate in leaf nodes only, and are served from upstream caches. They provided a $1/(1 - 1/e)$ approximation for a 2-level hierarchy and an approximation to hierarchies with more than 2 levels with an approximation ratio bound exceeding 2, which is worse than ours. Unlike works that provide approximations with bounded approximation ratio, Gkatzikis et al. [23] proposed to cluster contents and to solve the placement problem for clusters of contents. The authors proposed clustering schemes and evaluated the efficiency loss due to clustering. In contrast to [8,10,14,16,22–24], in our work we proposed a 2-approximation algorithm that can be executed in a decentralized manner, and we developed two distributed algorithms for computing a content placement based on limited information on the content demands and on the network topology, which can be used to solve large problem instances with prohibitive space complexity.

Related to ours are recent works on game theoretical analyses of distributed selfish replication on graphs [18–20,25–28], as they can serve as a basis for distributed content placement algorithms. Equilibrium existence when the access costs are homogeneous and nodes form a complete graph were provided in [18], and results on the approximation ratio (referred to as the price of anarchy) were provided in [25,26] for homogeneous costs and a complete graph. Non-complete graphs were considered in [19,27,28], and results on the approximation ratio of a distributed greedy algorithm were given for the case of unit storage capacity and an infinite number of objects in [27]. Dán [28] considered a variant of the problem where nodes can replicate a fraction of objects, and showed the existence of equilibria, while convergence results were provided for the integer problem in [19] in the case of homogeneous neighbor costs. The case of heterogeneous neighbor costs, for which the non-convergence of distributed greedy replication was shown in [20] is a generalization of our model, and thus the negative result provided in [20] may not apply to our case. Different from these works, in this paper we consider caches managed by a single entity, and thus we consider the minimization of the total cost as opposed to the selfish minimization of the cost of the individual nodes. Our objective of minimizing the total cost also sets this work apart from recent work on cache networks in the context of content centric networks, e.g., [29].

8. Conclusion

Motivated by mobile backhaul networks, we considered the problem of minimizing the bandwidth demand in a hierarchical cache network through cooperative caching, and formulated it as a 0-1 integer linear program. We showed that a polynomial time solution exists for special instances of the problem, and we proposed a 2-approximation algorithm that is based on global information for the general case. Furthermore, we proposed a low complexity distributed algorithm based on information about object demands at descendants, and an algorithm with an adjustable level of available information. We proved convergence and stability of the algorithms. We used extensive simulations to evaluate the performance of the proposed algorithms. Our results show that information about object demands at descendants is insufficient for good cooperative caching performance, but the proposed h-Push Down algorithm achieves consistently good performance despite limited information availability, consistently better than greedy optimization based on global information.

References

[1] G. Carofiglio, M. Gallo, L. Muscariello, D. Perino, Scalable Mobile Backhauling via Information-Centric Networking, in: Proc. of IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN), 2015, pp. 1–6, doi:10.1109/LANMAN.2015.7114719.
[2] G. Dán, N. Carlsson, Dynamic Content Allocation for Cloud-assisted Service of Periodic Workloads, in: Proc. of IEEE INFOCOM, 2014, pp. 1–9.
[3] U. Paul, A.P. Subramanian, M.M. Buddhikot, S.R. Das, Understanding Traffic Dynamics in Cellular Data networks, in: Proc. of IEEE INFOCOM, 2011, pp. 882–890, doi:10.1109/INFCOM.2011.5935313.
[4] M. Rodrigues, G. Dán, M. Gallo, Enabling Transparent Caching in LTE Mobile Backhaul Networks with SDN, in: Proc. of IEEE Infocom Workshops (SWFAN), 2016.
[5] H. Pinto, J.M. Almeida, M.A. Gonçalves, Using Early View Patterns to Predict the Popularity of Youtube Videos, in: Proc. of ACM Intl. Conf. on Web Search and Data Mining (WSDM), 2013, pp. 365–374, doi:10.1145/2433396.2433443.
[6] A. Tatar, M.D. de Amorim, S. Fdida, P. Antoniadis, A survey on predicting the popularity of web content, J. Internet Serv. Appl. 5 (1) (2014) 8, doi:10.1186/s13174-014-0008-y.
[7] Netflix, Open connect program.
[8] S. Borst, V. Gupta, A. Walid, Distributed Caching Algorithms for Content Distribution Networks, in: Proc. of IEEE INFOCOM, 2010, pp. 1478–1486.

[9] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy, Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, in: Proc. of ACM Symposium on Theory of Computing (STOC), 1997, pp. 654–663, doi:10.1145/258533.258660.
[10] I.D. Baev, R. Rajaraman, Approximation Algorithms for Data Placement in Arbitrary Networks, in: Proc. of ACM SODA, 2001.
[11] E.J. Rosensweig, J. Kurose, D. Towsley, Approximate Models for General Cache Networks, in: Proc. of IEEE INFOCOM, 2010, pp. 1–9, doi:10.1109/INFCOM.2010.5461936.
[12] C. Fricker, P. Robert, J. Roberts, A Versatile and Accurate Approximation for LRU Cache Performance, in: Proc. of the 24th International Teletraffic Congress (ITC), 2012, pp. 1–8.
[13] A. Schrijver, Theory of Linear and Integer Programming, 1, John Wiley & Sons, Inc., 1986, doi:10.1017/CBO9781107415324.004.
[14] K. Poularakis, L. Tassiulas, On the complexity of optimal content placement in hierarchical caching networks, IEEE Trans. Commun. 64 (5) (2016) 2092–2103, doi:10.1109/TCOMM.2016.2545655.
[15] M.L. Fisher, G.L. Nemhauser, L.A. Wolsey, An analysis of approximations for maximizing submodular set functions - II, Math. Program. Stud. 8 (1978) 73–78, doi:10.1007/BF01588971.
[16] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, R. Sitaraman, On the Complexity of Optimal Routing and Content Caching in Heterogeneous Networks, in: Proc. of IEEE INFOCOM, 2015, pp. 936–944.
[17] J.B. Kruskal, On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, in: Proceedings of the American Mathematical Society, 7, 1956, doi:10.1090/S0002-9939-1956-0078686-7. 1
[18] N. Laoutaris, O. Telelis, V. Zissimopoulos, I. Stavrakakis, Distributed selfish replication, IEEE Trans. Parallel Distrib. Syst. 17 (12) (2006) 1401–1413.
[19] V. Pacifici, G. Dán, Convergence in player-specific graphical resource allocation games, IEEE J. Sel. Areas Commun. 30 (11) (2012) 2190–2199.
[20] V. Pacifici, G. Dán, Distributed Algorithms for Content Allocation in Interconnected Content Distribution Networks, in: Proc. of IEEE INFOCOM, 2015, pp. 2362–2370.
[21] P. Rodriguez, C. Spanner, E.W. Biersack, Analysis of web caching architectures: hierarchical and distributed caching, IEEE/ACM Trans. Netw. 9 (4) (2001) 404–418, doi:10.1109/90.944339.
[22] M.R. Korupolu, M. Dahlin, Coordinated placement and replacement for large-scale distributed caches, IEEE Trans. Knowl. Data Eng. 14 (6) (2002) 1317–1329.
[23] L. Gkatzikis, V. Sourlas, C. Fischione, I. Koutsopoulos, G. Dán, Clustered Content Replication for Hierarchical Content Delivery Networks, in: Proc. of IEEE ICC, 2015.
[24] K. Poularakis, G. Iosifidis, L. Tassiulas, Approximation algorithms for mobile data caching in small cell networks, IEEE Trans. Commun. 62 (10) (2014) 3665–3677, doi:10.1109/TCOMM.2014.2351796.
[25] G. Pollatos, O. Telelis, V. Zissimopoulos, On the Social Cost of Distributed Selfish Content Replication, in: Proc. of IFIP Networking, 2008, pp. 195–206.
[26] E. Jaho, M. Karaliopoulos, I. Stavrakakis, Social similarity favors cooperation: the distributed content replication case, IEEE Trans. Parallel Distrib. Syst. 24 (3) (2013) 601–613.
[27] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C.H. Papadimitriou, J. Kubiatowicz, Selfish Caching in Distributed Systems: a Game-Theoretic Analysis, in: Proc. of ACM Symposium on Principles of Distributed Computing (PODC), 2004, pp. 21–30, doi:10.1145/1011767.1011771.
[28] G. Dán, Cache-to-cache: could ISPs cooperate to decrease peer-to-peer content distribution costs? IEEE Trans. Parallel Distrib. Syst. 22 (9) (2011) 1469–1482.
[29] N. Laoutaris, H. Che, I. Stavrakakis, The LCD interconnection of LRU caches and its analysis, Perform. Eval. 63 (7) (2006) 609–634, doi:10.1016/j.peva.2005.05.003.

Slađana Jošilo is a Ph.D. student at the Laboratory for Communication Networks in KTH, Royal Institute of Technology. She received her M.Sc. degree in electrical engineering from the University of Novi Sad, Serbia in 2011. She worked as a research engineer at the Department of Power, Electronics and Communication Engineering, University of Novi Sad from 2013 to 2014. Her research interests are design and analysis of distributed algorithms for exploiting resources available at the network edge using game theoretical tools.

Valentino Pacifici is a postdoctoral researcher at the Laboratory for Communication Networks in KTH Royal Institute of Technology, Stockholm, Sweden. He studied computer engineering at Politecnico di Milano in Milan, Italy. In October 2010, he completed a joint M.Sc. degree in computer engineering, between Politecnico di Milano and KTH Royal Institute of Technology. He received the Ph.D. in Telecommunications from KTH in 2016. His research interests include game theoretical and stochastic modeling, design and analysis of content management systems.

György Dán is an associate professor at KTH Royal Institute of Technology, Stockholm, Sweden. He received the M.Sc. in computer engineering from the Budapest University of Technology and Economics, Hungary in 1999, the M.Sc. in business administration from the Corvinus University of Budapest, Hungary in 2003, and the Ph.D. in Telecommunications from KTH in 2006. He worked as a consultant in the field of access networks, streaming media and videoconferencing 1999–2001. He was a visiting researcher at the Swedish Institute of Computer Science in 2008, a Fulbright research scholar at University of Illinois at Urbana-Champaign in 2012–2013, and an invited professor at EPFL in 2014–2015. His research interests include the design and analysis of content management and computing systems, game theoretical models of networked systems, and cyber-physical system security in power systems.