Distributed Approximation of k-Service Assignment

Magnús M. Halldórsson[*1], **Sven Köhler**[†2], **and Dror Rawitz**[2]

1 **Reykjavik University, Reykjavik, Iceland**
mmh@ru.is
2 **Bar-Ilan University, Ramat-Gan, Israel**
sven.kohler@biu.ac.il
3 **Bar-Ilan University, Ramat-Gan, Israel**
dror.rawitz@biu.ac.il

Abstract

We consider the k-Service Assignment problem (k-SA), defined as follows. The input consists of a network that contains servers and clients, and an integer k. Each server has a finite capacity, and each client is associated with a demand and a profit. A feasible solution is an assignment of clients to neighboring servers such that (i) the total demand assigned to a server is at most its capacity, and (ii) a client is assigned either to k servers or to none. The profit of an assignment is the total profit of clients that are assigned to k servers, and the goal is to find a maximum profit assignment. In the r-restricted version of k-SA, no client requires more than an r-fraction of the capacity of any adjacent server. The k-SA problem is motivated by backup placement in networks and by resource allocation in 4G cellular networks. It can also be viewed as machine scheduling on related machines with assignment restrictions.

We present a centralized polynomial time greedy $\frac{k+1-r}{1-r}$-approximation algorithm for r-restricted k-SA. We then show that a variant of this algorithm achieves an approximation ratio of $k+1$ using a resource augmentation factor of $1+r$. We use the latter to present a $(k+1)^2$-approximation algorithm for k-SA. In the distributed setting, we present: (i) a $(1+\varepsilon)\frac{k+1-r}{1-r}$-approximation algorithm for r-restricted k-SA, (ii) a $(1+\varepsilon)(k+1)$-approximation algorithm that uses a resource augmentation factor of $1+r$ for r-restricted k-SA, both for any constant $\varepsilon > 0$, and (iii) an $O(k^2)$-approximation algorithm for k-SA (in expectation). The three distributed algorithms compute a solution with high probability and terminate in $O(k^2 \cdot \log^3 n)$ rounds.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, distributed algorithms, related machines

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2015.11

1 Introduction

We consider the k-Service Assignment problem (abbreviated k-SA). A k-SA instance consists of a set of servers and a set of clients. Each server has a finite capacity, and each client has a demand and a profit. (The demand of a client does not depend on the identity of the server.) A feasible solution is a *k-service assignment* of clients to servers such that:

- A client is only assigned to neighboring servers.
- The total demand of clients that are assigned to a server does not exceed its capacity.
- Each client is assigned either to k servers or to none.

* Supported in part by Icelandic Research Fund (grants No. 120032011 and 152679-051).
† Supported in part by a grant from the Israeli Ministry of Science, Technology, and Space and by the Israel Science Foundation (grant No. 497/14).

A client that is assigned to k servers is said to be *satisfied*, and the profit of a service assignment is the total profit of satisfied clients. The goal in k-SA is to find a service assignment with maximum profit.

Given a constant $r \in (0, 1]$, an instance of k-SA is said to be r-restricted if no client requires more than an r-fraction of the capacity of any neighboring server. k-SA on r-restricted instances is referred to as r-restricted k-SA.

k-SA is NP-hard, since the special case with exactly k servers is equivalent to the Knapsack problem. Since Knapsack remains NP-hard even if the size of each item is at most an r-fraction of the knapsack size, this hardness result applies to r-restricted k-SA, for any $r \in (0, 1]$. (This was explicitly shown for 1-SA in [1].) This also means that the approximation ratio of the natural greedy algorithm is $\Omega(\frac{1}{1-r})$, even for r-restricted 1-SA.

The k-SA problem naturally arises in network applications where clients need service from (multiple) servers. Amzallag et al. [1] used 1-SA to model the problem of assigning clients to base stations in 4G cellular networks where services offered by providers (such as video streaming and web browsing) require high bit-rates, and client diversity is an issue. By using 1-SA they took into account both base stations diversity (using non-uniform capacities), as well as clients diversity (using different demands, profits, and potential set of base stations). Amzallag et al. [1] also considered the variant of 1-SA where a client c may be serviced by multiple servers as long as the total service it receives is $d(c)$. Such an assignment is called a *cover by many*, while a solution that assigns a single server to a client is called a *cover by one*. They presented a $\frac{2-r}{1-r}$-approximation algorithm that computes *covers by one* and a $\frac{1}{1-r}$-approximation algorithm that computes *covers by many*. In fact the former ratio is in comparison to an optimal *cover by many*. Both algorithms are based on the local ratio technique [5, 3, 4].

Patt-Shamir et al. [18] presented a distributed implementation of the first algorithm from [1] while paying a $(1 + \varepsilon)$ factor in the approximation ratio. That is, they presented a distributed $(1 + \varepsilon)\frac{2-r}{1-r}$-approximation algorithm, for any $\varepsilon > 0$, for r-restricted 1-SA, for any $r \in (0, 1)$. The algorithm requires a polylogarithmic number of rounds in the Congest model. The above result is based on two assumptions: (i) the *cost-effectiveness* of clients is polynomially bounded, and (ii) each server knows the demands and profits of adjacent clients. (The definition of cost-effectiveness and a more detailed description of these assumptions are given in the next section.)

Recently, Halldórsson at al. [13] considered the Backup Location problem in which each client has a file whose backup should be stored in k neighbors to increase fault tolerance. They mainly focused on the dual problem, where an instance is similar to a k-SA instance and the goal is to satisfy all clients while minimizing the maximum load. They also observed that k-SA is APX-hard, for $k \geq 3$, and showed a lower bound of $\Omega(\frac{k}{\log k})$ for the approximation ratio based on a reduction from k-Dimensional Matching [14].

Our Results. We generalize the $\frac{2-r}{1-r}$-approximation algorithm from [1] by presenting a $\frac{k+1-r}{1-r}$-approximation algorithm for r-restricted k-SA, for any $r \in (0, 1)$. We provide a simplified analysis that does not rely on the local ratio technique. We show that a variant of the above algorithm achieves an approximation ratio of $k + 1$ for r-restricted k-SA, for any $r \in (0, 1]$, using a resource augmentation factor of $1 + r$. Then, by showing that the clients that receive service in the resource augmented solution can be $(k + 1)$-colored, such that each color induces a feasible solution, we obtain a $(k + 1)^2$-approximation algorithm for k-SA. The algorithm outperforms the $\frac{k+1-r}{1-r}$-approximation algorithm, when $r > \frac{k+1}{k+2}$.

Based on the approach taken in [18], we design a distributed version of the former algorithm that, for any constant $\varepsilon > 0$, computes $(1 + \varepsilon)\frac{k+1-r}{1-r}$-approximate solutions for

r-restricted k-SA with high probability and whose running time is $O(k^2 \cdot \log^3 n)$ rounds in the Congest model. While the algorithm for 1-SA from [18] is based on computing a maximal matching, our algorithm is based on computing a maximal packing of stars, where each star consists of a client and k adjacent servers. As in the centralized setting we provide an algorithm that achieves a factor of $(1+\varepsilon)(k+1)$ using a resource augmentation factor $1+r$. We use distributed random selection instead of coloring to design a distributed algorithm for k-SA that computes solutions whose expected profit is a $\Omega(k^{-2})$-fraction of the optimum, using $O(k^2 \cdot \log^3 n)$ rounds. When $k = O(1)$, this amounts to an $O(1)$-approximation algorithm that terminates in $O(\log^3 n)$ rounds.

The results of this paper can be extended to a natural variant of k-SA in which each client $c \in C$ requires service from $k(c) \in \mathbb{N}$ servers. Then $k_{\max} \triangleq \max_{c \in C} k(c)$ replaces k in the approximation ratios and time complexities. We will address this variant in the full version of the paper.

Related Work. 1-SA is equivalent to Multiple Knapsack with Assignment Restrictions (MKAR), where the input consists of a set of bins and a set of items. Each bin has a capacity, and each item j has a size, a profit, and a subset of bins in which it can be placed. A feasible solution is an assignment of items to bins such that each item is assigned to one of the bins in its subset and the total size of items assigned to each bin is at most its capacity. The goal is to find a solution of maximum profit. A special case of MKAR, where the size and profit of an item are the same, was considered by Dawande et al. [8]. They presented an LP-rounding 2-approximation algorithm, a $(2+\varepsilon)$-approximation algorithm that uses an FPTAS for solving a single knapsack problem, and a greedy 3-approximation algorithm.

Fleischer et al. [11] studied the Separable Assignment Problem (SAP). In this problem the input consists of a set of bins and a set of items, and a profit f_{ij} for assigning item j to bin i. There is also a separate packing constraint for each bin, i.e., a collection $\mathcal{I}_i$ of subsets of items that fit in bin i. The goal is to maximize the total profit. Given an α-approximation algorithm for the single machine version of SAP, they presented an LP-rounding based $\frac{\alpha e}{e-1}$-approximation algorithm and a local search $(\frac{\alpha+1}{\alpha}+\varepsilon)$-approximation algorithm, for any $\varepsilon > 0$. If the single machine version admits a PTAS (FPTAS), then the ratios are $\frac{e}{e-1}+\varepsilon$ $(\frac{e}{e-1})$ and $2+\varepsilon$.

In the Generalized Assignment Problem (GAP) the input consists of a set of bins and a set of items. Each bin has a capacity, and each item j has a size and a profit for each bin i. A feasible solution is an assignment of items to bins such that the total size of items that are assigned to a bin is at most its capacity. GAP is a special case of SAP where the simple knapsack version admits an FPTAS, and thus it has a $\frac{e}{e-1}$-approximation algorithm. MKAR (and hence 1-SA) is a special case of GAP. Chekuri and Khanna [7] gave a PTAS for Multiple Knapsack (without assignment restrictions) and showed that GAP is APX-hard. In addition they observed that an LP-rounding 2-approximation algorithm for the minimization version of GAP by Shmoys and Tardos [22] implies a 2-approximation algorithm for GAP. This result applies to 1-SA.

Amzallag et al. [1] showed that the version of 1-SA that allows *cover by many* cannot be approximated to within a factor which is better than $|J|^{1-\varepsilon}$, for any $\varepsilon > 0$, unless NP=ZPP.

k-SA is a special case of the Packing Integer Programs problem (PIP). In this problem we are given a set of items and a collection of knapsack constraints over these items. The goal is to maximize the profit of packed items. In k-SA each item appears in k constraints with the same coefficient. The single constraint (or server) case is the Knapsack problem which has an FPTAS [21, 15], and the constant number of constraints

case is the Multi-dimensional Knapsack problem that has a PTAS [12], while obtaining an FPTAS is NP-hard [17]. Raghavan and Thompson [20] used randomized LP-rounding to obtain an approximation ratio of $O(m^r)$ for PIP, where m is the number of constraints, k is the maximum number of constraints per item, and r is the maximum item coefficient per constraint RHS. Srinivasan [23] improved this ratio to $O(m^{r/(r+1)})$. In k-SA this translates to an $O(|S|^{r/(r+1)})$ ratio, where S is the set of servers. Chekuri and Khanna [6] proved that the above ratio is almost tight by showing that, for every fixed integer α and fixed $\varepsilon > 0$, the special case of PIP where all constraints are composed of binary coefficients and RHS α cannot be approximated within a factor of $m^{1/(\alpha+1)-\varepsilon}$, unless NP=ZPP. They also showed that PIP with uniform RHS α cannot be approximated within a factor of $m^{1/(\alpha+1)-\varepsilon}$, unless NP=ZPP, even with a resource augmentation factor α. Note that this does not contradict our results, since we assume that each item appears in at most k constraints.

Paper Organization. We formally define the problem and the execution model in Section 2. This section also contains definitions and notation that is used in the paper. The centralized algorithms are given and Section 3 and the distributed algorithms are presented in Section 4.

2 Preliminaries

This section contains a formal problem statement, several definitions and notation that we use throughout the paper, and the execution model.

Problem. We consider the k-Service Assignment (k-SA) problem. A k-SA instance consists of a bipartite graph $G = (C, S, E)$, where C is a set of clients and S is a set of servers. Each server $s \in S$ has a positive capacity $cap(s)$, and each client $c \in C$ has a demand $d(c)$ and a profit $p(c)$. We define $n \triangleq |C| + |S|$. A feasible solution is a *k-service assignment* (or simply a *service assignment*) of clients to servers, i.e., it is a function $x : C \times S \to \{0, 1\}$ such that:

- A client is only assigned to neighboring servers, namely $x(c, s) = 1$ implies $(c, s) \in E$.
- The total demand of clients assigned to a server is not larger than its capacity, i.e., $\sum_{c \in C} x(c, s) \cdot d(c) \leq cap(s)$, for every server $s \in S$.
- Each client is assigned either to k servers or to none. That is, $\sum_{s \in S} x(c, s) \in \{0, k\}$, for every client $c \in C$.

Given a k-service assignment x, a client is *satisfied* if $\sum_{s \in S} x(c, s) = k$. The set of satisfied clients is denoted by C_x, that is $C_x \triangleq \left\{c \in C : \sum_{s \in S} x(c, s) = k\right\}$. The profit of a service assignment x is the total profit of satisfied clients, or $p(C_x) \triangleq \sum_{c \in C_x} p(c)$, and the goal in k-SA is to find a service assignment with maximum profit.

Given a constant $r \in (0, 1]$, a k-SA instance is said to be r-restricted if no client requires more than an r-fraction of the capacity of any neighboring server, namely if $d(c) \leq r \cdot cap(s)$, for every $(c, s) \in E$. k-SA on r-restricted instances is referred to as r-restricted k-SA.

Definitions, Notation, and Assumptions. We use standard graph theoretic notation. The neighborhood of a vertex v is denoted by $N(v)$, and the degree of v is denoted by $\deg(v)$.

If a function is applied to a finite set, then this yields the sum of function values for all elements of the set, e.g., $d(C) \triangleq \sum_{c \in C} d(c)$. Also, given a function f with a finite domain, let $f_{\min}$ and $f_{\max}$ denote the minimum and maximum value of f in its domain. For example, $p_{\max} \triangleq \max_{c \in C} p(c)$.

Given a k-SA instance and a k-service assignment x, the set of clients assigned to a server s is denoted by $C_x(s) = \{c \in C : x(c,s) = 1\}$, and note that $C_x = \cup_{s \in S} C_x(s)$. We call $d(C_x(s)) = \sum_{c \in C} x(c,s)d(c)$ the *load* of server s. In this paper we sometimes consider non-feasible k-service assignments that violate the server capacity constraints, in which case it is possible that $d(C_x(s)) > cap(s)$, for a server $s \in S$. Such a server is called *overloaded*. Given $\alpha \in [0,1]$, a server s is called α-*saturated* if $d(C_x(s)) \geq \alpha \cdot cap(s)$. A service assignment x is called α-*maximal*, if no unsatisfied client is adjacent to k non-α-saturated servers.

Given a k-SA instance, the *cost effectiveness* of a client $c \in C$ is denoted by $\rho(c) \triangleq \frac{p(c)}{d(c)}$. Cost-effectiveness is assumed to be polynomially bounded, i.e., $\rho(c) \geq \rho_{\min} \in n^{-O(1)}$ as well as $\rho(c) \leq \rho_{\max} \in n^{O(1)}$. The bounds $\rho_{\min}$ and $\rho_{\max}$ are assumed to be known to each node.

Following [18] we assume that each server s is aware of the demands and profits of adjacent clients, namely each server knows $d(c)$ and $p(c)$, for every $c \in N(s)$. Observe that even if the numbers are large, it may be the case that their encoding is somewhat small (i.e., of size $O(\log n)$). An actual implementation may use a floating-point encoding, so it may be possible to efficiently send the demands and profits of clients to the adjacent servers. We also consider an alternative assumption that all nodes know the maximum profit $p_{\max}$. Notice that while the latter assumption requires global knowledge, the former requires only local knowledge.

Execution Model. We use the classic CONGEST model [19], which is a network model with small messages. Briefly, in this model nodes are processors with unique IDs, connected by links that can carry $O(\log n)$-bit messages in a time unit, or *round*. Processors are not restricted computationally (all computations required by our algorithms are polynomial, though). As usual, for our upper bounds, we implicitly assume that the α-synchronizer [2] is employed in the system, so that the algorithms operate in a synchronous manner in the following sense. Execution proceeds in global *rounds*, where in each round each processor: (i) eceives messages sent by its neighbors in the previous round, (ii) performs a local computation, and (iii) sends (possibly distinct) messages to its neighbors.

3 Centralized Greedy Algorithm

In this section we present an algorithm that computes α-maximal k-service assignments. This algorithm is used to obtain three results: (i) a $\frac{k+1-r}{1-r}$-approximation algorithm for r-restricted k-SA, for any $r \in (0,1)$, (ii) a $(k+1)$-approximation algorithm for r-restricted k-SA, for any $r \in (0,1]$, using a resource augmentation factor of $1+r$, and (iii) a $(k+1)^2$-approximation algorithm for k-SA. The first algorithm extends the $\frac{2-r}{1-r}$-approximation algorithm for 1-SA from [18]. However, we provide a simplified analysis that does not use the local ratio technique. Also, note that while the $\frac{k+1-r}{1-r}$-approximation algorithm requires knowledge of r, the other two algorithms do not.

Algorithm **α-Greedy** (Algorithm 1) sorts the clients in a non-increasing order by cost effectiveness and then tries to service the clients in order. It assigns each client to some k adjacent servers that are not yet α-saturated, if possible; otherwise, the client is dismissed. We note that if $\alpha > 1 - r$, the computed solution x may be non-feasible.

▶ **Observation 1.** *Algorithm **α-Greedy** computes α-maximal service assignments.*

Proof. Assume that the computed solution x is not α-maximal. Then, there exists a client $c_i \in C$ that is adjacent to k non-α-saturated servers. It follows that when c_i is considered by **α-Greedy** these k servers are non-α-saturated, which means that c_i would have received service. A contradiction. ◀

Algorithm 1 : $\boldsymbol{\alpha}$**-Greedy**(C, S, E, d, p)

Let $\langle c_1, c_2, c_3, \ldots \rangle$ be a sequence of all clients sorted in non-increasing order of ρ
$x \leftarrow 0$
for $i = 1, 2, 3, \ldots$ **do**
  if there exist k non-α-saturated servers in $N(c_i)$ **then**
    Let $s_1, \ldots, s_k \in N(c_i)$ be k non-α-saturated servers
    $x(c_i, s_j) \leftarrow 1$, for every j
  end if
end for

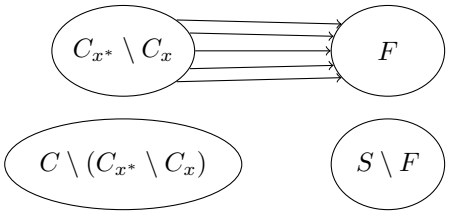

Figure 1 The arrows represent the mapping f.

We will in later sections be using rounding and therefore state our analysis of $\boldsymbol{\alpha}$**-Greedy** more generally than will be used in this section.

Let $\pi, \delta \geq 1$. Given a k-SA instance, let p' be a profit vector such that $p'(c) \in [p(c), \pi \cdot p(c)]$ and let d' be a demand vector such that $d'(c) \in [d(c), \delta \cdot d(c)]$. Define $\rho'(c) \triangleq \frac{p'(c)}{d'(c)}$.

▶ **Lemma 2.** *Given a k-SA instance, let x be the solution computed by $\boldsymbol{\alpha}$-Greedy using p' and d', and let x^* be an optimal solution with respect p and d. Then, we have that $p(C_x) \geq \frac{\alpha}{\delta\pi k + \alpha} p(C_{x^*})$.*

Proof. Let F be the set of servers that are α-saturated with respect to the $\boldsymbol{\alpha}$**-Greedy** solution x. Consider a client $c_i \in C_{x^*} \setminus C_x$ satisfied by the optimal solution x^* but not by $\boldsymbol{\alpha}$**-Greedy**. Since $\boldsymbol{\alpha}$**-Greedy** does not satisfy c_i and due to Observation 1, c_i must be connected to fewer than k non-α-saturated servers (in $S \setminus F$). Therefore, there exists an α-saturated server $s \in F$ that is assigned to c_i by the optimal solution, that is such that $x^*(c_i, s) = 1$. Let f be a mapping which maps each client $c_i \in C_{x^*} \setminus C_x$ to a server $s \in F$ such that $x^*(c_i, s) = 1$. This is depicted in Figure 1.

Observe that the load of an α-saturated server s is by definition at least

$$d'(C_x(s)) = \sum_c x(c, s) d'(c) \geq \alpha \cdot cap(s) \ .$$

Let $f^{-1}(s) = \{c \in C_{x^*} \setminus C_x : f(c) = s\}$ be the set of clients that are mapped to a α-saturated server $s \in F$. Since each such client is assigned to s in the optimal solution x^*,

$$d(f^{-1}(s)) \leq \sum_c x^*(c, s) d(c) \leq cap(s) \leq \frac{d'(C_x(s))}{\alpha} \ . \tag{1}$$

Consider a client $c_i \in f^{-1}(s)$ and a client $c_j \in C_x(s)$. Since x does not satisfy c_i, the server s must have been α-saturated when $\boldsymbol{\alpha}$**-Greedy** tried to assign c_i. Thus, c_j must have been considered by $\boldsymbol{\alpha}$**-Greedy** prior to c_i, and the cost-effectiveness of c_j is then at least as high as that of c_i, i.e., $\rho'(c_j) \geq \rho'(c_i)$. It follows that $\rho'(c) \leq \rho'(c')$, for every $c \in f^{-1}(s)$ and $c' \in C_x(s)$. This implies that $\rho'(c) \leq \frac{p'(C_x(s))}{d'(C_x(s))}$, for every $c \in f^{-1}(s)$.

For the total profit of all clients that f maps to s we then have that

$$\begin{aligned} p(f^{-1}(s)) \;\leq\; p'(f^{-1}(s)) \;&=\; \sum_{c\in f^{-1}(s)} d'(c)\cdot\rho'(c) \\ &\leq\; \sum_{c\in f^{-1}(s)} d'(c)\cdot\frac{p'(C_x(s))}{d'(C_x(s))} \\ &=\; d'(f^{-1}(s))\cdot\frac{p'(C_x(s))}{d'(C_x(s))} \\ &\leq\; \delta\cdot d(f^{-1}(s))\cdot\frac{p'(C_x(s))}{d'(C_x(s))} \;\leq\; \frac{\delta}{\alpha}p'(C_x(s)) \;\leq\; \frac{\pi\delta}{\alpha}p(C_x(s)) \;, \end{aligned}$$

where the third inequality is due to (1).

It remains to bound the approximation ratio:

$$\begin{aligned} p(C_{x^*}) \;&=\; \sum_{c\in C_{x^*}\cap C_x} p(c) \;+\; \sum_{c\in C_{x^*}\setminus C_x} p(c) \\ &\leq\; \sum_{c\in C_x} p(c) \;+\; \sum_{s\in F} p(f^{-1}(s)) \\ &\leq\; p(C_x) \;+\; \sum_{s\in F}\frac{\pi\delta}{\alpha}p(C_x(s)) \quad\leq\; p(C_x) \;+\; k\cdot\frac{\pi\delta}{\alpha}p(C_x) \;=\; \frac{\alpha+\pi\delta k}{\alpha}\cdot p(C_x) \;, \end{aligned}$$

where the last inequality holds because each client is assigned to k servers. ◀

We note that a similar proof can be given with comparison to an optimal fractional solution as was done in [1] for the case of $k = 1$.

Furthermore, we show that the analysis of **α-Greedy** is almost tight. Consider the following k-SA instance for the case where $1-\alpha = \frac{1}{q}$, for $q \in \mathbb{N}$. Let $C = \{c_1, c_2, \ldots\}$ be a set of $q(k+1)-1$ clients and $S = \{s_1, s_2, \ldots\}$ be a set of $2k-1$ servers. For $i \leq q-1$, let $d(c_i) = q$, $p(c_i) = q^t + 1$, and $N(c_i) = \{s_1, \ldots, s_k\}$, while for $i \geq q$, let $d(c_i) = q$, $p(c_i) = q^t$, and $N(c_i) = S$. As for server capacities, $cap(s_i) = q^2$, for $i \leq k$, and $cap(s_i) = kq^2$, for $i > k$. If we run **α-Greedy** (assuming $\delta = \pi = 1$), it will consider clients $c_1, \ldots, c_{q-1}$ first and assigns all of them. This renders servers $s_1, \ldots, s_k$ α-saturated, so that no other clients will receive service. Thus, **α-Greedy** obtains a profit of $(q-1)(q^t+1)$, while an optimal solution services clients $c_q, \ldots, c_{q(k+1)-1}$ for a profit of $kq \cdot q^t = kq^{t+1}$. Hence, the approximation ratio of **α-Greedy** is at least $\frac{kq^{t+1}}{(q-1)(q^t+1)}$, which goes to $\frac{k}{\alpha}$ as t goes to infinity.

We get our first result by assigning $\alpha = 1 - r$ and $\delta = \pi = 1$.

▶ **Corollary 3.** *If $\delta = \pi = 1$, the approximation ratio of $(1-r)$-**Greedy** is at most $\frac{k+1-r}{1-r}$.*

Our next result is obtained by assigning $\alpha = 1$ and $\delta = \pi = 1$. Notice that in this case the server capacity constraints may be violated, but not by much.

▶ **Lemma 4.** *Given an r-restricted k-SA instance, let x be a k-service assignment computed by 1-**Greedy** with $\delta = \pi = 1$. Then, the load on any server s is less than $(1+r)\cdot cap(s)$. Moreover, if we remove the last client assigned to each overloaded server we obtain a feasible k-service assignment.*

Proof. By the algorithm design, an overloaded server s was non-1-saturated when the last client was assigned to it. The load of a non-1-saturated server is less than its capacity, while the last client assigned to s has a demand of at most $r \cdot cap(s)$. ◀

From Lemma 4 we get that 1-**Greedy** obtains an approximation ratio of $k+1$ with a resource augmentation factor $(1+r)$.

▶ **Corollary 5.** *If $\delta = \pi = 1$, then 1-**Greedy** is a $(k+1)$-approximation algorithm for r-restricted k-SA that uses $(1+r)$ times the capacity of each server.*

In the next lemma we show that the non-feasible solution that is computed by 1-**Greedy** can be partitioned into $k+1$ feasible solutions.

▶ **Lemma 6.** *Given a k-SA instance, let x be a k-service assignment computed by 1-**Greedy** with $\delta = \pi = 1$. Then, x can be partitioned into $k+1$ feasible k-service assignments.*

Proof. Consider the directed conflict graph $G' = (C_x, E')$, where E' contains an arc (c, c') if and only if 1-**Greedy** assigned both c and c' to a server s and c was the last client assigned to s. The maximum in-degree of G' is at most k for the simple reason that x assigns at most k servers to each client. Furthermore, the graph G' is a DAG, since an edge (c, c') always points from a client c to a client c' that was considered by 1-**Greedy** prior to c. It follows that the underlying graph of G' is k-*degenerate* (or k-*inductive*), and therefore can be $(k+1)$-colored [10]. For completeness, we provide the following simple recursive algorithm that $(k+1)$-colors G':

- If the graph is empty, return an empty coloring.
- Find a node v with out-degree zero. Such a node always exists as the graph is a DAG.
- Remove v from the graph and color the remaining graph recursively.
- Color node v with the smallest available color. Since only k neighbors of v have already received a color – only the in-neighbors – at least one of the first $k+1$ colors is free.
- Return the coloring.

The coloring of G' is a partition of C_x into $k+1$ independent sets. We show that an independent set induces a feasible solution. Let I be an independent set and let x^I be the solution restricted to I, that is $x^I(c,s) = x(c,s)$, if $c \in I$, and $x^I(c,s) = 0$, otherwise. If I contains the last client assigned to a server s, then I does not contain any other clients assigned to s. It follows that x^I is feasible. ◀

This leads to the last result of the section.

▶ **Corollary 7.** *There exists a $(k+1)^2$-approximation algorithm for k-SA.*

Proof. First, 1-**Greedy**, with $\delta = \pi = 1$, computes a possibly non-feasible $(k+1)$-approximate service assignment due to Corollary 5. Lemma 6 implies that x can be partitioned into $k+1$ feasible solutions $x^1, \ldots, x^{k+1}$. Since $p(C_x) = \sum_{i=1}^{k+1} p(C_{x^i})$, there exists i such that $p(C_{x^i}) \geq \frac{1}{k+1} p(C_x)$. ◀

4 Distributed Greedy Algorithm

In this section, we present distributed approximation algorithms for k-SA by providing a distributed implementation of Algorithm $\boldsymbol{\alpha}$-**Greedy**. More specifically, we present (i) a $\frac{k+1-r}{1-r}(1+\gamma)$-approximation algorithm for r-restricted k-SA, (ii) a $(k+1)(1+\gamma)$-approximation algorithm that uses a resource augmentation factor $1+r$ for r-restricted k-SA, and (iii) a $O(k^2)$-approximation algorithm for k-SA, all for any constant $\gamma > 0$. The three algorithms terminate in $O(k^2\gamma^{-2}\operatorname{polylog}(n))$ rounds.

We first give a distributed algorithm that relies on the assumption that all nodes know $p_{\max}$. We then give a modification that does not need this assumption, but relies on the assumption that each server s knows the demands and profits of the clients in $N(s)$. We start the section by classifying the clients.

4.1 Client Classification

The basic idea of our distributed algorithm is to mimic the sequential **α-Greedy**. The challenge is to parallelize the computation of the assignment as dealing with clients one-by-one would yield linear running time. The key is to efficiently compute the assignment of multiple clients with equal profit and equal demand. To enlarge the number of clients with equal profit and demand, we apply an implication of Lemma 2: we may round profits and demands up to the closest power of $1+\varepsilon$, for some $\varepsilon > 0$, increasing the approximation ratio by at most a factor of $(1+\varepsilon)^2$.

We first classify all clients by demand and profit. Define

$$C_\ell^i \triangleq \left\{c \in C : d(c) \in ((1+\varepsilon)^{i-1}, (1+\varepsilon)^i] \ \wedge \ p(c) \in ((1+\varepsilon)^{i+\ell-1}, (1+\varepsilon)^{i+\ell}]\right\} ,$$

and $C_\ell \triangleq \bigcup_i C_\ell^i$. Also, define the *rounded* demand and profit for all clients $c \in C_\ell^i$ as

$$d'(c) \triangleq (1+\varepsilon)^i$$
$$p'(c) \triangleq (1+\varepsilon)^{i+\ell} .$$

Note that all clients in C_ℓ have equal cost-effectiveness with respect to the rounded profits and demands, namely $\rho'(c) = \frac{p'(c)}{d'(c)} = (1+\varepsilon)^\ell$. That means that the clients in C_ℓ can be considered by Algorithm **α-Greedy** in any order. Also note that

$$d'(c) \in [d(c), (1+\varepsilon)d(c))$$
$$p'(c) \in [p(c), (1+\varepsilon)p(c))$$
$$\rho'(c) \in ((1+\varepsilon)^{-1}\rho(c), (1+\varepsilon)\rho(c)) .$$

For the remainder of the section, we mostly consider rounded profits and demands.

4.2 Distributed Implementation of α-Greedy

We are ready to describe a distributed implementation of Algorithm **α-Greedy** that relies on the assumption that all nodes know $p_{\max}$. The algorithm is described in a top-down manner.

By assumption, the cost-effectiveness is polynomially bounded in n. Given the values $\rho_{\min}$ and $\rho_{\max}$, we can find an interval $[W, W']$ such that $C_\ell \neq \emptyset$ only if $\ell \in [W, W']$. This is the case for $W = \lfloor \log_{1+\varepsilon}(\rho_{\min}) \rfloor$ and $W' = \lceil \log_{1+\varepsilon}(\rho_{\max}) \rceil$. Note that $W' - W \in O(\log_{1+\varepsilon} n) \subseteq O(\varepsilon^{-1} \log n)$.

Define $C_{\geq z} \triangleq \bigcup_{\ell \geq z} C_\ell$ and assume that there is an algorithm called **Augment** that augments a k-service assignment for $C_{\geq \ell+1}$ into a k-service assignment for $C_{\geq \ell}$. Algorithm **Dist-α-Greedy** (Algorithm 2) uses **Augment** iteratively to construct a k-service assignment. Clearly **Dist-α-Greedy** runs for $O(T_A \varepsilon^{-1} \log_n)$ rounds, where T_A is the running time of **Augment**.

Algorithm 2 : **Dist-α-Greedy**(C, S, E, d, p, cap)

1: $x \leftarrow 0$
2: **for** $\ell = W'$ **downto** W **do**
3: $\quad x \leftarrow$ **Augment**(C, S, E, cap, ℓ, x)
4: **end for**

As shown in the sequel, Algorithm **Augment** considers the subclasses of C_ℓ one by one and augments the given k-service assignment with a k-service assignment for each C_ℓ^i. In

order to keep the running time of our algorithm poly-logarithmic, we use the next result showing that only considering $O(\log_{1+\varepsilon} n)$ subclasses per class C_ℓ does not increase the approximation ratio by much.

A client $c \in C$ is called *heavy* if $p'(c) > \frac{p'_{\max}}{n^3}$. Otherwise, it is called *light*. Recall that $p'_{\max} = \max_{c \in C} p'(c)$. Define $C_{\text{heavy}} \triangleq \{c \in C : p'(c) > \frac{p'_{\max}}{n^3}\}$. The next lemma explains why we can simply ignore light clients.

▶ **Lemma 8.** *Let x be an optimal k-service assignment and let y be an optimal k-service assignment for the same instance but restricted to a set $\tilde{C}$, where $\tilde{C} \supseteq C_{heavy}$. Then $p(C_x) \leq (1 + \frac{1}{n^2}) p(C_y)$.*

Proof. Observe that each client $c \notin \tilde{C}$ satisfies $p(c) \leq \frac{p_{\max}}{n^3}$. Clearly, $p(C_x) \geq p(C_y) \geq p_{\max}$, and thus we have that

$$p(C_x) = p(C_x \cap \tilde{C}) + p(C_x \setminus \tilde{C}) \leq p(C_y) + n \cdot \frac{p_{\max}}{n^3} = p(C_y) + \frac{p_{\max}}{n^2} \leq \left(1 + \frac{1}{n^2}\right) p(C_y)$$

which concludes the proof. ◀

Following the above result, Algorithm **Augment** (Algorithm 3) considers only the subclasses C_ℓ^i which contain heavy clients. The heavy clients are contained in at most $\lceil 3 \log_{1+\varepsilon} n \rceil$ subclasses of C_ℓ. For each subclass, **Augment** uses Algorithm **Uniform-Augment** which augments the current k-service assignment with an assignment for the specified subclass C_ℓ^i. Recall that all clients in C_ℓ^i have the same profit and demand (with respect to p' and d').

Algorithm 3 : **Augment**(C, S, E, cap, ℓ, x)

```
1: i_ℓ^max ← log_{1+ε} p'_max − ℓ
2: for i = i_ℓ^max downto i_ℓ^max − ⌈3 log_{1+ε} n⌉ + 1 do
3:     x ← Uniform-Augment(C, S, E, cap, i, ℓ, x)
4: end for
5: return x
```

Clearly, if Algorithm **Uniform-Augment** requires T_U rounds, then **Augment** terminates after $O(T_U \log_{1+\varepsilon} n) \subseteq O(T_U \varepsilon^{-1} \log n)$ rounds. It follows that Algorithm **Dist-α-Greedy** requires $O(T_U \varepsilon^{-2} \log^2 n)$ rounds.

As mentioned before, Algorithm **Uniform-Augment** is used to compute a k-service assignment for all clients in a given subclass C_ℓ^i that augments a given k-service assignment x. Recall that clients have *uniform* demands, i.e., $d'(c) = (1+\varepsilon)^i$ for each $c \in C_\ell^i$. Hence, given a solution x and a server s, an upper bound $m_\ell^i(x, s)$ on the number of clients from C_ℓ^i that can be assigned to s while it is not α-saturated can be computed as follows:

$$m_\ell^i(x, s) = \min\left\{ \max\left\{ 0, \left\lceil \frac{\alpha \cdot cap(s) - d'(C_x(s))}{(1+\varepsilon)^i} \right\rceil \right\}, \deg(s) \right\} .$$

A *star* centered at a client $c \in C$ is a subgraph of G that contains c and k servers adjacent to c. We call the servers the leaves of the star. Per server $s \in S$ we introduce $m_\ell^i(x, s)$ copies denoted s_1, s_2, and so forth. An *incarnation* of a star replaces each leaf s with a copy s_q, where $1 \leq q \leq m_\ell^i(x, s)$. Note that incarnations never have two leaves which are copies of the same server. Also note that some stars have no incarnations, namely if $m_\ell^i(x, s) = 0$ for some leaf s. We define the graph $H(i, \ell, x)$. The vertex set of $H(i, \ell, x)$

contains all possible incarnations of stars centered at a client $c \in C^i_\ell$. There is an edge between two nodes of $H(i,\ell,x)$, namely between two incarnations, if and only if the two are either centered at the same client or share a common leaf (copy of a server). Given i, ℓ, and x, Algorithm **Uniform-Augment** (Algorithm 4) constructs $H(i,\ell,x)$ and computes a *maximal independent set* (MIS) in $H(i,\ell,x)$.

Algorithm 4 : **Uniform-Augment**$(C, S, E, cap, i, \ell, x)$

1: $MIS \leftarrow$ Maximal Independent Set of $H(i,\ell,x)$
2: Augment x with k-service assignment corresponding to MIS
3: **return** x

The computation of the MIS is based on Luby's algorithm [16]. In fact we rely on the analysis of Wattenhofer [24] that shows that the MIS algorithm terminates with high probability after $O(\log N)$ rounds, where N is the number of nodes in the graph. The next lemma shows how to implement the algorithm such that the number of rounds is $O(k^2 \log n)$.

▶ **Lemma 9.** *Algorithm* ***Uniform-Augment*** *computes an α-maximal service assignment w.h.p. in $O(k^2 \log n)$ rounds.*

Proof. Consider the graph $H(i,\ell,x) = (V(i,\ell,x), E(i,\ell,x))$. For a client $c \in C$, there are $\binom{\deg(c)}{k}$ stars centered at c. For each star there are at most n^k different incarnations, as there are at most $\deg(s) \leq n$ copies of each server s. It follows that, per client c of G, the vertex set $V(i,\ell,x)$ contains at most $\binom{\deg(c)}{k} n^k \leq n^{2k}$ vertices. So in total, $V(i,\ell,x)$ contains

$$n(i,\ell,x) \triangleq |V(i,\ell,x)| \leq n^{2k+1}$$

vertices (incarnations of stars of G).

We would like to execute Luby's algorithm [16] to compute a maximal independent set in $H(i,\ell,x)$. Let $M = \emptyset$ and $\overline{M} = V(i,\ell,x)$. Luby's algorithm repeatedly executes the following procedure:

- Each incarnation in $\overline{M}$ is assigned a random priority with $O(k \log n)$ bits.
- Let U be the set of incarnations with a priority higher than any adjacent incarnation.
- All incarnations in U are added to M and removed from $\overline{M}$. Also, all incarnations adjacent to incarnations in U are removed from $\overline{M}$.

With high probability, M is a maximal independent set after the above procedure has been executed $O(k \log n)$ times [24].

We now describe how the above procedure can be simulated on the graph G:

1. A client $c \in C^i_\ell$ draws a random priority for each incarnation in $\overline{M}$ centered at c. Per client, only the incarnation with the highest priority is relevant to Luby's algorithm. Thus for each leaf s_q of such an incarnation, clients send the priority and the index q to s. Note that each client sends at most one message to each adjacent server.
2. Per copy, each server determines the highest priority received. The server sends an ACK message to the clients that sent the winning (highest) priorities and a NACK message to clients that sent the losing priorities.
3. If a client c receives k ACK messages, then the incarnation with the highest priority centered at c joins the independent set and all other incarnations centered at c are removed from $\overline{M}$. Per leaf s_q of an incarnation joining the MIS, the clients inform server s that the copy with index q has been taken.
4. Servers keep track which copies have been taken and inform the clients which copies are no longer available. Clients remove all incarnations with unavailable leaves from $\overline{M}$.

We examine the messages exchanged during this procedure and their sizes in bits. In the first step each client sends a priority and server copy index to k servers, therefore the message size is $O(k \log n + \log n) \subseteq O(k \log n)$. In the second step each server sends a single ACK/NACK message to clients whose size is $O(1)$. Winning clients send a single server copy index to k servers. The message size is $O(\log n)$. Finally, servers need to update clients on which copy cannot be used anymore. The naive solution is a message that may require $\Omega(n)$ bits (a bit vector with one bit per server copy).

Recall that for each star there are $O(n^k)$ different incarnations. Since these incarnations are interchangeable, server copies may be relabeled after each iteration such that the available copies have the smallest possible indexes. It follows that the number of available copies per server, and not the actual server copy indexes, is important. In conclusion, it suffices to inform the clients only about the number of available copies per server. This can be done using a message of size $O(\log n)$ bits.

The above procedure takes $O(1)$ rounds when assuming messages of size $O(k \log n)$ or $O(k)$ rounds using messages of size $O(\log n)$ bits. As the procedure needs to be executed $O(k \log n)$ times, we have that the total number of rounds is $O(k^2 \log n)$.

Finally, the computed solution is α-maximal, since otherwise the independent set in $H(i, \ell, x)$ is not maximal. ◀

We note that a client c need not draw one random priority for each incarnation in $\overline{M}$ centered at c. As c is aware of the number of available copies per adjacent server, it is easy to count the number of incarnations in $\overline{M}$ centered at c. Let z be this number. It then suffices to choose one of the z incarnations uniformly at random and to draw its priority from the distribution of the maximum over z random priorities (see, e.g., [9]).

We bound the running time of Algorithm **Dist-α-Greedy**.

▶ **Lemma 10.** *Algorithm **Dist-α-Greedy** terminates w.h.p. in $O(k^2 \varepsilon^{-2} \cdot \log^3 n)$ rounds.*

Proof. Algorithm **Dist-α-Greedy** consists of $O(\varepsilon^{-1} \log n)$ invocations of **Augment**, which in turn consists of $O(\varepsilon^{-1} \log n)$ invocations of **Uniform-Augment**. The lemma follows since **Uniform-Augment** requires $O(k^2 \log n)$ rounds according to Lemma 9. ◀

Next, we analyze the computed solution. In preparation for Section 4.3, the next result is slightly more general than necessary.

▶ **Lemma 11.** *Given a k-SA instance, Algorithm **Dist-α-Greedy** mimics **α-Greedy** on a set $\tilde{C} \supseteq C_{heavy}$ using the rounded profits p' and the rounded demands d'.*

Proof. Notice that Algorithm **Dist-α-Greedy** considers $C_{W'}, \ldots, C_W$ in decreasing order of ℓ. Since all clients in C_ℓ have the same cost-effectiveness, $(1+\varepsilon)^\ell$, it follows that the algorithm augments x according to a non-increasing order of client cost-effectiveness with respect to p' and d'. For each ℓ, Algorithm **Augment** considers subclasses C_ℓ^i in a decreasing order of i, that is in a decreasing order of both profit and demand. For each i, **Uniform-Augment** computes an α-maximal solution, as shown in Lemma 9, by adding clients with the same profit, demand, and cost-effectiveness in an order that is induced by the random choices of the maximal independent set computation. Hence, Algorithm **Augment** can be seen as trying to service clients with the same cost-effectiveness in an arbitrary order. It follows that **Dist-α-Greedy** is a specific implementation of **α-Greedy**.

It remains to show that **Augment** considers all heavy clients in each class C_ℓ. Let $c \in C_\ell$ be a client not considered by **Augment**. Then $c \in C_\ell^i$ with $i \leq i_\ell^{\max} - \lceil 3 \log_{1+\varepsilon} n \rceil$ and we

have that

$$p'(c) = (1+\varepsilon)^{i+\ell} \leq (1+\varepsilon)^{i_\ell^{\max}+\ell-\lceil 3\log_{1+\varepsilon} n\rceil} = (1+\varepsilon)^{\log_{1+\varepsilon} p'_{\max}-\lceil 3\log_{1+\varepsilon} n\rceil} \leq \frac{p'_{\max}}{n^3} ,$$

and thus c is a light client. ◀

The previous lemma allows us to find a lower bound on the profit of the solution that is computed by **Dist-α-Greedy**.

▶ **Lemma 12.** *Given a k-SA instance, let x be the solution computed by **Dist-α-Greedy** using p' and d', and let x^* be an optimal solution with respect p and d. Then, we have that $p(C_x) \geq \frac{1}{1+1/n^2} \cdot \frac{1}{(1+\varepsilon)^2} \cdot \frac{\alpha}{k+\alpha} p(C_{x^*})$.*

Proof. Let y^* be an optimal solution with respect to p, d, and $\tilde{C} \supseteq C_{\text{heavy}}$. We have that $p(C_{x*}) \leq (1+\frac{1}{n^2})p(C_{y^*})$ by Lemma 8. Furthermore, $p(C_x) \geq \frac{1}{(1+\varepsilon)^2} \cdot \frac{\alpha}{k+\alpha} p(C_{y^*})$ due to Lemmas 2 and 11 and the definition of p' and d'. The claim follows. ◀

▶ **Lemma 13.** *Let $\gamma > 0$ be a constant. There exists distributed $((1+\gamma)\frac{k+\alpha}{\alpha})$-approximation algorithm for k-SA that terminates w.h.p. in $O(k^2\gamma^{-2} \cdot \log^3 n)$ rounds.*

Proof. If $\gamma < \frac{4}{n^2}$, then $n \leq 2/\sqrt{\gamma}$ which means that $n = O(1)$. In this case, an optimal solution can be computed in $O(1)$ rounds as follows: each node sends its input to the node with highest id, which computes an optimal solution and broadcasts it to all nodes.

If $\gamma \geq \frac{4}{n^2}$, then set $\varepsilon = \gamma/4$ and run **Dist-α-Greedy**. In this case we have that

$$(1+\frac{1}{n^2}) \cdot (1+\varepsilon)^2 \leq \left(1+\frac{\gamma}{4}\right)^3 = \left(1+\frac{3\gamma}{4}+\frac{3\gamma^2}{16}+\frac{\gamma^3}{64}\right) < 1+\gamma .$$

The rest follows from Lemmas 10 and 12. ◀

By setting $\alpha = 1-r$, Lemma 13 leads to the following result:

▶ **Corollary 14.** *There exists a distributed $((1+\gamma)\frac{k+1-r}{1-r})$-approximation algorithm for r-restricted k-SA that terminates w.h.p. in $O(k^2\gamma^{-2} \cdot \log^3 n)$ rounds, for every $\gamma > 0$.*

We can obtain a better ratio using resource augmentation, i.e., by setting $\alpha = 1$.

▶ **Corollary 15.** *There exists a distributed $(1+\gamma)(k+1)$-approximation algorithm for r-restricted k-SA that uses at most $(1+r)$ times the capacity of each server and terminates w.h.p. in $O(k^2\gamma^{-2} \cdot \log^3 n)$ rounds, for every $\gamma > 0$.*

As in the centralized case (Lemma 6) we use the resource augmentation algorithm in order to obtain a feasible service assignment. However, in the distributed setting we use random selection instead of using coloring.

▶ **Theorem 16.** *There exists a distributed algorithm for k-SA that terminates w.h.p. in $O(k^2\gamma^{-2} \cdot \log^3 n)$ rounds and computes solutions whose expected profit is at least $p(C_{x^*})/((1+\gamma) \cdot 4k(k+1))$, for any $\gamma > 0$, where x^* is an optimal solution.*

Proof. We present a distributed randomized algorithm that computes a service assignment whose expected profit is at least the optimum divided by $(1+\gamma) \cdot 4k(k+1)$, for any constant $\gamma > 0$.

The first phase of the algorithm is to compute a $(1+\gamma)(k+1)$-approximate solution x for k-SA that uses at most $(1+r)$ times the capacity of each server. By Corollary 15 this

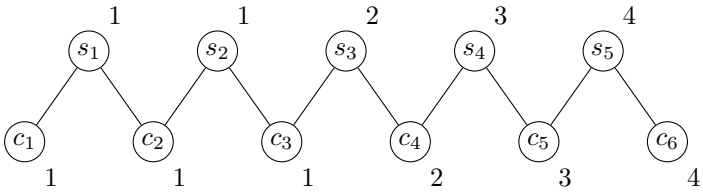

Figure 2 Each client is labeled with the index of its class, each server is labeled with $i_\ell^{\max}(s)$.

takes $O(k^2\gamma^{-2} \cdot \log^3 n)$ rounds. The solution is either already feasible or was computed by **Dist-1-Greedy** (see Lemma 13). In the latter case, consider the set of clients that were last assigned to s by an invocation of **Uniform-Augment** and choose as $c(s)$ the client with the largest identifier.

Recall the definition of the conflict graph $G' = (C_x, E')$ from Lemma 6. The set E' contains an edge between two clients (c, c') if c and c' are both assigned to a server s and $c = c(s)$. The second phase is to compute an independent set I of G'. As shown in the proof of Lemma 6, restricting x to the clients in I yields a feasible solution.

We exploit that G' is a DAG with in-degree at most k. Let $\beta > 1$ and $U = \emptyset$. Add each client $c \in C_x$ to U independently with probability $\frac{1}{\beta k}$. Then let $I \subseteq U$ be the set of clients with no in-neighbor in U. Clearly I is an independent set of G'. By the Union Bound, a node has an in-neighbor in U with probability at most $k \cdot \frac{1}{\beta k} = \frac{1}{\beta}$. Thus, a node of C_x is in I with probability at least $\frac{1}{\beta k}(1 - \frac{1}{\beta}) = \frac{\beta-1}{\beta^2 k}$. We choose $\beta = 2$ to maximize $\frac{\beta-1}{\beta^2}$, so a node of C_x is in I with probability $\frac{1}{4k}$. Thus $\mathbb{E}[p(I)] \geq \frac{1}{4k} p(C_x)$.

The set I can be easily constructed by the following distributed algorithm. Every client in $c \in C_x$ informs each server s with $x(c, s) = 1$ whether $c \in U$. A server s responds to a client $c \in U$ with a NACK message if $c \neq c(s)$ and $c(s) \in U$, and with an ACK message otherwise. If a client $c \in U$ receives no NACK message, then $c \in I$. Otherwise, c informs its servers that $c \notin I$. ◀

4.3 Modification

In the remainder of this section, we describe a modified version of **Augment**, called **Modified-Augment**, that does not assume knowledge of $p_{\max}$. Instead, we assume that each server s knows the demand $d(c)$ and the profit $p(c)$ of each adjacent client $c \in N(s)$, as explained in Section 2.

Without knowledge of $p_{\max}$, intuitively, we would like to start with the non-empty subclass C_ℓ^i with maximum index i. A naive approach, such as determining the maximum index i and making it known to all nodes, would require time proportional to the network diameter. Our algorithm avoids this issue by using the index

$$i_\ell^{\max}(s) = \max\{i : C_\ell^i \cap N(s) \neq \emptyset\} \ ,$$

for each server $s \in S$. See Figure 2 for an example. As the demands and profits are known, a server s can easily determine $i_\ell^{\max}(s)$.

Algorithm **Modified-Augment** works as follows. In each iteration of a loop starting at $i = i_\ell^{\max}(s)$ and counting downwards, each server s sends a START message to each adjacent client in class C_ℓ^i. It then runs Algorithm **Uniform-Augment** for index i. The execution of **Uniform-Augment** for index i is restricted to the graph $G(i, \ell) = (C_\ell^i, S, E \cap (C_\ell^i \times S))$. Thus, a client $c \in C_\ell^i$ may only receive messages due to an execution of **Uniform-Augment** for index i.

A client $c \in C_\ell^i$ doesn't run Algorithm **Uniform-Augment** for index i straightaway. Instead, it waits until all adjacent servers have sent a START message. While delaying the execution of **Uniform-Augment**, incoming messages for **Uniform-Augment** are saved by c and delivered later when its execution starts.

As messages are delayed and since **Uniform-Augment** was written for the synchronous model, we use an α-synchronizer to execute **Uniform-Augment**. Also, we assume that the synchronizer counts how many synchronous rounds **Uniform-Augment** has been executed for. This serves as a means of termination detection. Let T_U be the worst-case running time of **Uniform-Augment** in synchronous rounds. After starting an execution of **Uniform-Augment** for a particular index, servers and clients wait until the synchronizer has completed T_U synchronous rounds of **Uniform-Augment**. Once this has happened, clients update the k-service assignment and servers continue with the next iteration of the loop over i.

Consider the graph shown in Figure 2. In the first round, servers s_1 and s_2 send a START message to c_1, c_2, and c_3. So servers s_1 and s_2 as well as clients c_1 and c_2 start executing **Uniform-Augment** for class C_ℓ^1. However, client c_3 will locally delay the execution of **Uniform-Augment** for class C_ℓ^1 until it received a START message from server s_3. This will only happen after s_3 and c_4 have finished executing **Uniform-Augment** for class C_ℓ^2. This in turn will be delayed until s_4 and c_5 have finished executing **Uniform-Augment** for class C_ℓ^3. We observe that the execution of **Uniform-Augment** for some class C_ℓ^i is delayed for at most $(i_\ell - i) \cdot O(T_U)$ rounds, where $i_\ell = \max_{s \in S} i_\ell^{\max}(s)$.

As we do not need to consider all subclasses of C_ℓ but mainly subclasses with heavy clients, we simply stop the execution of **Modified-Augment** after $O(T_U \varepsilon^{-1} \log n)$ rounds and take the k-service assignment computed by then. Abruptly stopping the execution may render the local view of the computed k-service assignment by clients and servers inconsistent. This can be fixed within one round by letting each client c send the value $x(c, s)$ to each adjacent server s. As the following result shows, the given time bound suffices to let **Modified-Augment** consider all heavy clients.

▶ **Lemma 17.** *With high probability, Algorithm* ***Modified-Augment*** *requires* $O(k^2 \varepsilon^{-1} \log^2 n)$ *rounds to run* ***Uniform-Augment*** *for all subclasses of* C_ℓ *that contain heavy clients.*

Due to space constraints, the listing of Algorithm **Modified-Augment** and the proof of Lemma 17 have been omitted. Algorithm **Modified-Augment** is a drop-in replacement for Algorithm **Augment** in the sense that it preserves all significant properties of **Augment**: (i) All subclasses of C_ℓ with heavy clients are considered, (ii) **Modified-Augment** has the same asymptotic runtime as Algorithm **Augment**, and (iii) the subclasses C_ℓ^i are (locally) considered in decreasing order of index i. We conclude that in particular Lemmas 10 and 11 remain true if **Augment** is replaced with **Modified-Augment** (details are omitted). As all subsequent results in Section 4.2 are mainly derived from these two lemmas, they also remain true.

Acknowledgements. We thank Boaz Patt-Shamir for helpful discussions.

References

1 David Amzallag, Reuven Bar-Yehuda, Danny Raz, and Gabriel Scalosub. Cell selection in 4G cellular networks. *IEEE Trans. Mobile Comput.*, 12(7):1443–1455, 2013.

2 Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.

3 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baurch Schieber. A unified approach to approximating resource allocation and schedualing. *J. ACM*, 48(5):1069–1090, 2001.

4 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms. *ACM Comput. Surv.*, 36(4):422–463, 2004.

5 Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

6 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004.

7 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005.

8 Milind Dawande, Jayant Kalagnanam, Pinar Keskinocak, F. Sibel Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, 4(2):171–186, 2000.

9 Yuval Emek, Magnús M. Halldórsson, Yishay Mansour, Boaz Patt-Shamir, Jaikumar Radhakrishnan, and Dror Rawitz. Online set packing. *SIAM J. Comput.*, 41(4):728–746, 2012.

10 Paul Erdös and András Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica*, 17(1–2):61–99, 1966.

11 Lisa Fleischer, Michel X. Goemans, Vahab S. Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *17th SODA*, pages 611–620, 2006.

12 A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional $0-1$ knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, 15:100–109, 1984.

13 Magnús M. Halldórsson, Sven Köhler, Boaz Patt-Shamir, and Dror Rawitz. Distributed backup placement in networks. In *27th ACM SPAA*, pages 274–283, 2015.

14 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006.

15 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

16 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

17 Michael J. Magazine and Maw-Sheng Chern. A note on approximation schemes for multidimensional knapsack problems. *Mathematics of Operations Research*, 9(2):244–247, 1984.

18 Boaz Patt-Shamir, Dror Rawitz, and Gabriel Scalosub. Distributed approximation of cellular coverage. *J. Parallel Distrib. Comput.*, 72(3):402–408, 2012.

19 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. SIAM, 2000.

20 Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

21 Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *J. ACM*, 22(1):115–124, 1975.

22 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

23 Aravind Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29(2):648–670, 1999.

24 Roger Wattenhofer. Principles of distributed computing: Maximal independent set. `http://www.dcg.ethz.ch/lectures/fs15/podc/lecture/chapter7.pdf`. Accessed 2015-08-27.