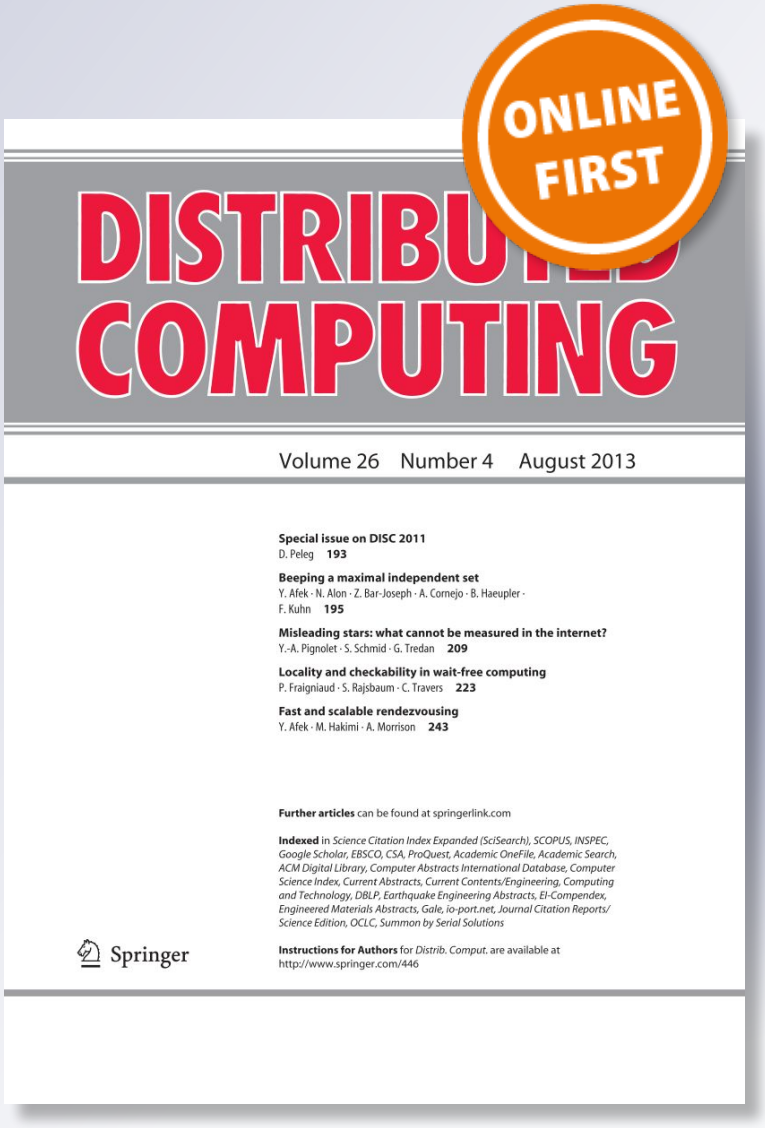
Distributed approximation of k-service assignment

Magnús M. Halldórsson, Sven Köhler & Dror Rawitz

Distributed Computing

ISSN 0178-2770

Distrib. Comput.
DOI 10.1007/s00446-017-0321-3



Springer

Distributed Computing
https://doi.org/10.1007/s00446-017-0321-3

Distributed approximation of *k*-service assignment

Magnús M. Halldórsson[1] · Sven Köhler[2] · Dror Rawitz[3]

Received: 10 November 2016 / Accepted: 7 December 2017

Abstract
We consider the k-SERVICE ASSIGNMENT problem (k-SA). The input consists of a network that contains servers and clients. Associated with each client is a demand and a profit. In addition, each client c has a *service requirement* $\kappa(c) \leq k$, where $\kappa(c)$ is a positive integer. A client c is *satisfied* only if its demand is handled by exactly $\kappa(c)$ neighboring servers. The objective is to maximize the total profit of satisfied clients, while obeying the given capacity limits of the servers. We focus here on the more challenging case of hard constraints, where no profit is granted for partially satisfied clients. This models, e.g., when a client wants, for reasons of fault tolerance, a file to be stored at $\kappa(c)$ or more nearby servers. Other motivations from the literature include resource allocation in 4G cellular networks and machine scheduling on related machines with assignment restrictions. In the r-restricted version of k-SA, no client requires more than an r-fraction of the capacity of any adjacent server. We present a (centralized) polynomial-time $\frac{k+1-r}{1-r}$-approximation algorithm for r-restricted k-SA. A variant of this algorithm achieves an approximation ratio of $k+1$ when given a resource augmentation factor of $1+r$. We use the latter result 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 run in $O(k^2\varepsilon^{-2}\log^3 n)$ synchronous rounds (with high probability). In particular, this yields the first distributed $O(1)$-approximation of 1-SA.

1 Introduction

We consider the k-SERVICE ASSIGNMENT problem (abbreviated k-SA). A k-SA instance consists of a network containing servers and clients. Associated with each client c is a demand $d(c)$, a profit $p(c)$, and a *service requirement* $\kappa(c) \leq k$, where $\kappa(c)$ is a positive integer. Each server s has a capacity $cap(s)$.

A client c is *satisfied* only if its demand is assigned to exactly $\kappa(c)$ neighboring servers. Note that the demand cannot be split up between servers, i.e., the full demand $d(c)$ must be assigned individually to each of the $\kappa(c)$ servers. The objective is to maximize the total profit of satisfied clients, while obeying that the total demand assigned to a server s must not exceed $cap(s)$. We focus here on the more challenging case of hard constraints, where no profit is granted for partially satisfied clients. This models, e.g., when a client wants to store $\kappa(c)$ backups of a file of size $d(c)$ at nearby servers for reasons of fault tolerance. Other motivations from the literature include resource allocation in 4G cellular net-

A preliminary version was presented at the 19th International Conference on Principles of Distributed Systems (OPODIS) 2015.
M. M. Halldórsson supported in part by the Icelandic Research Fund (Grant nos. 120032011 and 152679-051).
S. Köhler supported in part by the Sustainability Center Freiburg, a cooperation of the Fraunhofer Society and the University of Freiburg, supported by grants from the Baden-Württemberg Ministry of Economics and the Baden-Württemberg Ministry of Science, Research and the Arts.
D. Rawitz supported in part by a grant from the Israeli Ministry of Science, Technology, and Space (Grant no. 3-10996) and by the Israel Science Foundation (Grant no. 497/14).

✉ Sven Köhler
koehlers@informatik.uni-freiburg.de

Magnús M. Halldórsson
mmh@ru.is

Dror Rawitz
dror.rawitz@biu.ac.il

[1] ICE-TCS, School of Computer Science, Reykjavik University, Reykjavik, Iceland

[2] Faculty of Engineering, University of Freiburg, Freiburg im Breisgau, Germany

[3] Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

Published online: 30 December 2017

works and machine scheduling on related machines with assignment restrictions.

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 s servers and $\kappa(c) = s$, for each client c, 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 (that tries to add clients in a non-increasing ordering of profit per demand) 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 greedy $\frac{2-r}{1-r}$-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 were analyzed using the local ratio technique [3–5], and the second algorithm relies on an algorithm for computing MAXIMUM FLOW.

Patt-Shamir et al. [23] 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. [17] considered the BACKUP LOCATION problem in which each client has an object whose backup should be stored in $k \in \mathbb{N}$ 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 [19].

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. We note that the latter outperforms the $\frac{k+1-r}{1-r}$-approximation algorithm, when $r > \frac{k+1}{k+2}$.

Based on the approach taken in [23], 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 [23] 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 c and $\kappa(c)$ 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 $k \in O(1)$, this amounts to an $O(1)$-approximation algorithm that terminates in $O(\log^3 n)$ rounds. In particular, this approach yields the first constant approximation of 1-SA in the distributed setting.

We note that our distributed algorithms do not require nodes to have knowledge of k or r.

Related work 1-SA is equivalent to an extension of the KNAPSACK problem called 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 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 each item are the same, was considered by Dawande et al. [10]. 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.

MKAR (and hence 1-SA) is a special case of the GENERALIZED ASSIGNMENT PROBLEM (GAP). In GAP the input consists of a set of bins and a set of items, where 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. Bar-Noy et al. [3] implicitly gave a local ratio $(2+\varepsilon)$-approximation algorithm for GAP. Cohen et al. [9] showed that any α-approximation algorithm for KNAPSACK can be transformed into a local ratio $(1+\alpha)$-approximation algorithm for GAP. This resulted in a $(2+\varepsilon)$-approximation algorithm for GAP with a better running time than implied by [3]. Cohen and Grebla [8] extended this result by allowing multiple choice, thus combining GAP and MULTIPLE CHOICE KNAPSACK PROBLEM (MCKP).

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 [28] implies a 2-approximation algorithm for GAP. This result applies to 1-SA. Fleischer et al. [14] studied the SEPARABLE ASSIGNMENT PROBLEM (SAP), where 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. Assuming the existence of an α-approximation algorithm for the single bin 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 $2+\varepsilon$, respectively. 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. The best known result for GAP is an LP-rounding $(\frac{e}{e-1}-\varepsilon)$-approximation algorithm, for some constant $\varepsilon > 0$, due to Feige and Vondrák [13].

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. This was done using a reduction from INDEPENDENT SET and the hardness result from [18]. Gurewitz et al. [16] studied an extension of 1-SA, in which every pair of a job $j \in J$ and a machine $i \in I$ is associated with a service rate $R(i,j) \in (0,1]$ and a *cover by many* assignment is allowed. In this setting a client can be assigned to multiple servers and a job is considered satisfied if $\sum_i x(i,j)R(i,j) \geq d(j)$, where $d(j)$ is the demand of job j. Under the assumption that there exists some $\delta > 0$ such that if $R(i,j) > 0$ then $R(i,j) > \delta$, they presented a $\frac{1}{(1-r)\delta}$-approximation algorithm for $r \in (0,1)$. The algorithm adopt the approach taken in [1]. However, as opposed to [1], it uses solutions to linear programs in the process of adding a job. Rawitz and Voloshin [26] also considered an extension of 1-SA that allows *cover by many*, but also allows *flexible* demands. They presented approximation algorithms whose approximation ratios are $\frac{1}{1-r}$ and 2.

The special case of k-SA in which each client c is adjacent to exactly $\kappa(c)$ servers is a special case of the PACKING INTEGER PROGRAMS problem (PIP). In PIP 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 at most k constraints with the same coefficient. The single constraint (or server) case is the KNAPSACK problem which has an FPTAS [20,27], and the constant number of constraints case is the MULTIDIMENSIONAL KNAPSACK problem that has a PTAS [15], while obtaining an FPTAS is NP-hard [22]. Raghavan and Thompson [25] 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 [29] 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 Sect. 2. This section also contains definitions and notation that is used in the paper. The centralized algorithms are given and Sect. 3 and the distributed algorithms are presented in Sect. 4.

2 Preliminaries

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

2.1 Problem definition

In the k-SERVICE ASSIGNMENT (k-SA) problem an 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

positive capacity $cap(s)$, and each client $c \in C$ has positive demand $d(c)$, positive profit $p(c)$, and service requirement $\kappa(c) \leq k$, where $\kappa(c)$ is a positive integer. We define $n \triangleq |C|+|S|$. A feasible solution is 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 c is assigned either to $\kappa(c)$ servers or to none. That is, $\sum_{s \in S} x(c, s) \in \{0, \kappa(c)\}$, for every client $c \in C$.

Given a service assignment x, a client c is considered *satisfied* if it receives service from $\kappa(c)$ servers, i.e., if $\sum_{s \in S} x(c, s) = \kappa(c)$. Then the total service received is $\kappa(c) \cdot d(c)$. The set of satisfied clients is denoted by C_x, that is $C_x \triangleq \left\{c \in C : \sum_{s \in S} x(c, s) = \kappa(c)\right\}$. 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.

Without loss of generality, we assume that each client is adjacent to at least $\kappa(c)$ severs. Also, since our algorithms do not rely on the knowledge of k, it may be assumed that $k = \max_c \kappa(c)$. Furthermore, if $\kappa(c_i) = \kappa(c_j)$ for every $c_i, c_j \in C$, we refer to the instance as a *uniform* k-SA instance. In this case one may assume that $\kappa(c) = k$, for every client c.

2.2 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 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 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 c is adjacent to $\kappa(c)$ non-α-saturated servers.

Given a k-SA instance, the *cost effectiveness* of a client 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 [23] 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 assumption requires only local knowledge.

2.3 Execution model

We use the classic CONGEST model [24], which is a distributed 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 that are performed by our algorithms are polynomial, though). For our upper bounds, we implicitly assume that an 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) Receives 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 service assignments. This algorithm is used to obtain three results:

1. A $\frac{k+1-r}{1-r}$-approximation algorithm for r-restricted k-SA, for any $r \in (0, 1)$.
2. A $(k+1)$-approximation algorithm for r-restricted k-SA, for any $r \in (0, 1]$, using a resource augmentation factor of $1 + r$.
3. 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 [1]. However, we provide a simplified analysis that does not use the local ratio technique.

Algorithm **α-Greedy** (Algorithm 1) is a greedy algorithm. It starts by sorting the clients in a non-increasing order of cost-effectiveness, and then it tries to service the clients in that order. It assigns each client c to some $\kappa(c)$ 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 infeasible.

Algorithm **α-Greedy** is a variant of the natural greedy algorithm that tries to service clients by order of cost-effectiveness. However, there is a slight difference, namely that **α-Greedy** does not use a server which is α-saturated. We note that the more natural greedy algorithm could have been used, but we chose to use **α-Greedy**, since it comes with a simpler analysis.

In what follows we either assign $\alpha = 1 - r$ or $\alpha = 1$. In the former option the algorithm computes a feasible solution in which servers are not fully utilized. On the other hand, in the second option the algorithm better utilizes servers, but computes an infeasible solution.

Alg. 1 **α-Greedy**$(C, S, E, \kappa, cap, d, p)$

1: Let $\langle c_1, c_2, c_3, \ldots \rangle$ be a sequence of all clients sorted in a non-increasing order of ρ
2: $x \leftarrow 0$
3: **for** $i = 1, 2, 3, \ldots$ **do**
4: **if** $N(c_i)$ contains $\kappa(c_i)$ non-α-saturated servers **then**
5: Let $s_1, \ldots, s_{\kappa(c_i)} \in N(c_i)$ be non-α-saturated servers
6: $x(c_i, s_j) \leftarrow 1$, for every j
7: **end if**
8: **end for**

Observation 1 *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 $\kappa(c_i)$ non-α-saturated servers. It follows that when c_i is considered by **α-Greedy** these $\kappa(c_i)$ servers are non-α-saturated, which means that c_i would have received service. A contradiction. □

In Sect. 4, we will modify the demands and profits and therefore we state the analysis of **α-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 **α-Greedy** using p' and d', and let x^* be an optimal solution with respect to p and d. Then, we have that $p(C_x) \geq \frac{\alpha}{\delta\pi k+\alpha} p(C_{x^*})$.*

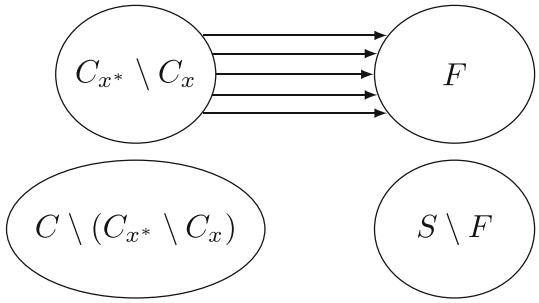


Fig. 1 The arrows represent the mapping f

Proof Let F be the set of servers that are α-saturated with respect to the **α-Greedy** solution x. Consider a client $c_i \in C_{x^*} \setminus C_x$ satisfied by the optimal solution x^* but not by **α-Greedy**. Since **α-Greedy** does not satisfy c_i and due to Observation 1, c_i must be connected to fewer than $\kappa(c_i)$ 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 Fig. 1.

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

$$d'(C_x(s)) = \sum_c x(c, s) d'(c) \geq \alpha \cdot cap(s). \tag{1}$$

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

$$d(f^{-1}(s)) \leq \sum_c x^*(c, s) d(c) \leq cap(s) \leq \frac{d'(C_x(s))}{\alpha} \tag{2}$$

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 **α-Greedy** tried to assign c_i. Thus, c_j must have been considered by **α-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

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

where the fourth inequality is due to (2).

It remains to bound the approximation ratio:

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

where the last inequality holds because each client is assigned to at most k servers from F. □

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

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** for r-restricted k-SA is at most $\frac{k+1-r}{1-r}$.*

We note that Lemma 2 and Corollary 3 still hold if instead of using r and $\alpha = 1 - r$, each server s makes its decisions according to its local versions of r and α, namely on $r(s) \triangleq \max_{c:(c,s)\in E} \frac{d(c)}{cap(s)}$ and $\alpha(s) \triangleq 1 - r(s)$. Replacing α with its local versions works since $\alpha(s) \ge \alpha$, for every server s. More specifically, if s is $\alpha(s)$-saturated, then it is also α-saturated, which means that (1) still holds. This observation will be useful in the distributed setting.

We show that the analysis of α-**Greedy** is almost tight. Consider the following uniform 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 $i \le q - 1$ and a positive parameter t, let $d(c_i) = q$, $p(c_i) = q^t + 1$, and $N(c_i) = \{s_1, \ldots, s_k\}$, while for $i \ge q$, 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 $i \le k$, and $cap(s_i) = kq^2$, for $i > k$. If we run α-**Greedy** (assuming $\delta = \pi = 1$), it considers 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 $\frac{kq}{q-1} = \frac{k}{\alpha}$ as t goes to infinity. 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 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 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, with a resource augmentation factor $(1+r)$, the approximation ratio of **1-Greedy** is $k+1$,

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 service assignment computed by **1-Greedy** with $\delta = \pi = 1$. Then, x can be partitioned into $k + 1$ feasible service assignments.*

Proof Consider the conflict digraph $G' = (C_x, E')$, where E' contains an edge (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.

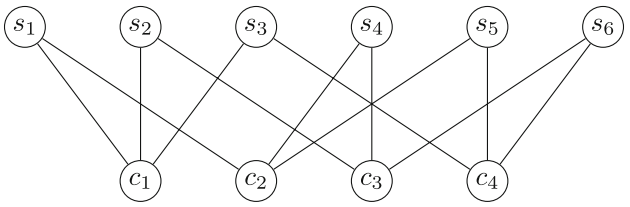


Fig. 2 Construction showing that Lemma 6 is tight for $k = 3$

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 [12]. 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 v has at most k neighbors that 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. □

We show that the analysis is tight by considering a graph with $k+1$ clients and $\binom{k+1}{2}$ servers. Each server is connected to a distinct set of 2 clients. Note that, by construction, each client is connected to exactly k servers. See Fig. 2 for an example. All demands are 2, and all capacities are 3. All clients have the same profit. Indeed we can only take one of the clients as taking 2 clients would violate the capacity constraint of the server which is connected to both.

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 α**-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 $4k(k+1)(1+\gamma)$-approximation algorithm for k-SA, all for any constant $\gamma > 0$. The three algorithms terminate in $O(k^2\gamma^{-2}\,\mathrm{polylog}(n))$ rounds.

The distributed implementation of Algorithm α**-Greedy** is based on the following idea. We partition the client set into subsets, such that the clients in each subset have similar cost-effectiveness, and instead of trying to augment the solution one client at a time, we do so for subsets. The crux is that augmenting the solution with respect to a subset becomes an unweighted star packing problem that we solve using a variant of Luby's algorithm [21] for computing a maximal independent set.

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 for integers ℓ and i

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

and $C[\ell] \triangleq \bigcup_i C[\ell, i]$. We call $C[\ell, i]$ a subclass of class $C[\ell]$. 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 $C[\ell]$ 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 $C[\ell]$ 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 of each client 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 service assignment for $C_{\geq \ell+1}$ into a service assignment for $C_{\geq \ell}$. Algorithm **Dist-α-Greedy** (Algorithm 2) uses **Augment** to iteratively construct a 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**.

Alg. 2 Dist-α-Greedy $(C, S, E, \kappa, cap, d, p)$

```
1: x ← 0
2: for ℓ = W' downto W do
3:     x ← Augment(C, S, E, κ, cap, ℓ, x)
4: end for
```

As shown in the sequel, Algorithm **Augment** considers the subclasses of $C[\ell]$ one by one and augments the given service assignment with a service assignment for each $C[\ell, i]$. 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 $C[\ell]$ 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 service assignment and let y be an optimal 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

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

which concludes the proof. □

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

Alg. 3 Augment$(C, S, E, \kappa, cap, \ell, x)$

```
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, κ, cap, i, ℓ, x)
4: end for
5: return x
```

Clearly, if **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 service assignment for the clients in a given subclass $C[\ell, i]$, augmenting a given 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 $m^i_\ell(x, s)$ on the number of clients from $C[\ell, i]$ that can be assigned to s before it becomes α-saturated can be computed as follows:

$$m^i_\ell(x, s) = \min\left\{\max\left\{0, q^i_\ell(x, s)\right\}, \deg(s)\right\}$$

with

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

We note that α may be replaced by $\alpha(s)$ in case $\alpha = 1-r$ and global knowledge of r is not assumed. (Recall that $\alpha(s) = 1 - r(s)$ and $r(s) = \max_{c:(c,s)\in E} \frac{d(c)}{cap(s)}$.)

A *star* centered at a client $c \in C$ is a subgraph of G that contains c and $\kappa(c)$ servers adjacent to c. We call the servers the leaves of the star. Per server $s \in S$ we introduce $m^i_\ell(x, s)$ copies denoted s_1, s_2, and so forth. An *incarnation* of a star replaces each leaf s with a copy s_q, where $1 \le q \le m^i_\ell(x, s)$. Note that incarnations never have two leaves which are copies of the same server. Also note that some stars have no incarnations, namely if $m^i_\ell(x, s) = 0$ 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 $c \in C[\ell, i]$. There is an edge between two nodes of $H(i, \ell, x)$, i.e., between two incarnations, if and only if the two are 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 $H(i, \ell, x)$.

Alg. 4 Uniform-Augment$(C, S, E, \kappa, cap, i, \ell, x)$

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

The computation of the MIS is based on Luby's algorithm [21]. We rely on the analysis of Wattenhofer [30] 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. In the next lemma we show 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 with high probability in $O(k^2 \log n)$ rounds.*

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

$$\binom{\deg(c)}{\kappa(c)} n^{\kappa(c)} \le n^{2\kappa(c)} \le n^{2k}$$

vertices. Thus, in total, $V(i, \ell, x)$ contains

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

vertices (incarnations of stars of G).

We would like to execute Luby's algorithm [21] 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 [30].

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

1. A client $c \in C[\ell, i]$ draws a random priority for each (surviving) 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. Each server determines the highest priority received for each of its copies. 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 of 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 adjacent 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 adjacent servers. The message size is $O(\log n)$. Finally, servers need to inform clients about which copies 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 $O(n^{2k})$ different incarnations. Observe that these incarnations are interchangeable, since the identity of the copy s_q of each server s in the star is not important. Hence, we may assume that the available

copies of a server s are always in the range $\{1, \ldots, m_s\}$, where m_s is the current number of available copies of s. In other words, 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 is important while the actual server copy indexes are not. In conclusion, it suffices to inform the clients only about the number of available copies per server, thus providing them with the new range of available copies. 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 actually draw a 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., [11]).

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

Lemma 10 *Algorithm **Dist-α-Greedy** terminates with high probability 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 Sect. 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 $C[\ell]$ 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 $C[\ell, i]$ 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 $C[\ell]$. 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

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

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 to 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 with high probability 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

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

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 with high probability 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 with high probability 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 with high probability 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 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, for each server s, consider the set of clients that were last assigned to a server s by an invocation of **Uniform-Augment** and choose as $c(s)$ the client with the largest identifier.

Recall the conflict graph $G' = (C_x, E')$ that was defined in 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 $U = \emptyset$. We denote by $N^-(c)$ the set of in-neighbors of c in G' and by $N^+(c)$ the set of out-neighbors. We define a local version of k for each client $c \in C_x$, namely $k(c) = \max_{c'\in N^+(c)}\kappa(c')$ if $N^+(c) \neq \emptyset$ and $k(c) = 1$ otherwise. Add each client $c \in C_x$ to U independently with probability $\frac{1}{\beta k(c)}$. 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 client c has an in-neighbor in U with probability at most

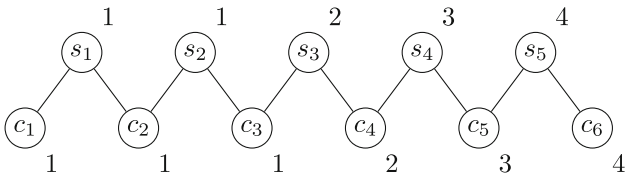


Fig. 3 Each client is labeled with the index of its class, each server is labeled with $i_\ell^{\max}(s)$

$$\sum_{c'\in N^-(c)} \frac{1}{\beta k(c')} \leq \kappa(c)\frac{1}{\beta\kappa(c)} = \frac{1}{\beta}\,.$$

Thus, a client c is in I with probability at least

$$\frac{1}{\beta k(c)}\left(1-\frac{1}{\beta}\right) = \frac{\beta-1}{\beta^2 k(c)} \geq \frac{\beta-1}{\beta^2 k}\,.$$

We choose $\beta = 2$ to maximize $\frac{\beta-1}{\beta^2}$, so a node of C_x is in I with probability at least $\frac{1}{4k}$. Thus $\mathbb{E}[p(I)] \geq \frac{1}{4k}p(C_x)$.

The set I can be easily constructed by the following distributed algorithm. Each client $c \in C_x$ sends $\kappa(c)$ to every server s with $x(c,s) = 1$. Each server s then sends the value $\max_{c\neq c(s)}\kappa(c)$ to the client $c(s)$. At this point, every client $c \in C_x$ knows the value of $k(c)$. Each client c then informs every 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 **Mod-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 Sect. 2.

Without knowledge of $p_{\max}$, intuitively, we would like to start with the non-empty subclass $C[\ell, i]$ of 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 Fig. 3 for an example. As the demands and profits are known, a server s can easily determine $i_\ell^{\max}(s)$.

Algorithm **Mod-Augment** (Algorithm 5) 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 $C[\ell, i]$. 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.

Alg. 5 Mod-Augment $(C, S, E, \kappa, cap, \ell, x)$

```
1: each s ∈ S runs
2:   for i = i_ℓ^max(s) downto 0 do
3:     Send START message to every client c ∈ N(s) ∩ C[ℓ, i]
4:     Using an α-synchronizer,
         execute T_U synchronous rounds
         of Uniform-Augment(G(i, ℓ), κ, cap, i, ℓ, x)
5:   end for
6: end

1: each c ∈ C[ℓ] runs
2:   while not all adjacent servers
         have sent a START message do
3:     Receive all messages and save any
         messages sent by Uniform-Augment
4:   end while
5:   Using an α-synchronizer and delivering all saved
       messages, execute T_U synchronous rounds
       of Uniform-Augment(G(i, ℓ), κ, cap, i, ℓ, x)
6: end
```

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 the number of synchronous rounds of **Uniform-Augment** that it has completed. 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 service assignment and servers continue with the next iteration of the loop over i.

Consider the graph shown in Fig. 3. 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 $C[\ell,1]$. However, client c_3 locally delays the execution of **Uniform-Augment** for class $C[\ell,1]$ until it receives a START message from server s_3. This only happens after s_3 and c_4 finish executing **Uniform-Augment** for class $C[\ell,2]$. This in turn is delayed until s_4 and c_5 finish executing **Uniform-Augment** for class $C[\ell,3]$. We observe that the execution of **Uniform-Augment** for some class $C[\ell,i]$ 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 only need to consider all subclasses of $C[\ell]$ that contain heavy clients, we simply stop the execution of **Mod-Augment** after $O(T_U \varepsilon^{-1}\log n)$ rounds and take the service assignment computed by then. Abruptly stopping the execution may render the local view of the computed 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 **Mod-Augment** consider all heavy clients.

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

Proof Let $i_\ell = \max_{s\in S} i_\ell^{\max}(s)$ and by $S(i)$ denote the set of servers with $i \le i_\ell^{\max}(s)$, i.e., $S(i) \triangleq \{s \in S : i \le i_\ell^{\max}(s)\}$. Also, fix an execution e of Algorithm **Mod-Augment** and let T_U be the worst-case running time of **Uniform-Augment**. Then let $M(i)$ denote the set of servers that have sent a START message to all adjacent clients in class $C[\ell,i]$ within the first $1+(i_\ell - i)(T_U+1)$ rounds of e. Note that all servers adjacent to a client of $C[\ell,i]$ are in $S(i)$.

We show that $S(i) \subseteq M(i)$, for all i, by induction on i. An implication of $S(i) \subseteq M(i)$ is that all clients in class $C[\ell,i]$ and their adjacent servers start executing algorithm **Uniform-Augment** for class $C[\ell,i]$ at the latest after round $1+(i_\ell - i)(T_U+1)$ of e and thus have finished its execution before the end of round $1+(i_\ell - i)(T_U+1)+T_U$.

The base case is $S(i_\ell) \subseteq M(i_\ell)$, which is true since clearly $s \in M(i_\ell^{\max}(s))$, for all $s \in S$. For $i > 0$ we show that if $S(i) \subseteq M(i)$, then $S(i-1) \subseteq M(i-1)$. By the argument given above, all servers in $S(i)$ finish the execution of **Uniform-Augment** before the end of round $1+(i_\ell - i)(T_U+1)+T_U$. So they will send a START message to all adjacent clients in class C_ℓ^{i-1} in the next round, which is $1+(i_\ell - i+1)(T_U+1)$. Thus, $S(i) \subseteq M(i-1)$. It remains to prove that $s \in M(i-1)$ for each $s \in S(i-1)\setminus S(i)$. Note that such a server satisfies $i_\ell^{\max}(s) = i-1$ and thus $s \in M(i-1)$. Consequently, $S(i-1) \subseteq M(i-1)$.

Each heavy client of $C[\ell]$ is contained in some class $C[\ell,i]$ with $i \ge i_\ell - \lceil 3\log_{1+\varepsilon} n\rceil$. As shown above, the clients in any such class $C[\ell,i]$ have been considered after at most $1+(i_\ell - i)(T_U+1)+T_U \le 1+\lceil 3\log_{1+\varepsilon} n\rceil (T_U+1)+T_U$ rounds. Note that **Uniform-Augment** is invoked by **Mod-Augment** for at most n non-empty classes. Therefore we have that $T_U \in O(k^2\log n)$ with high probability. □

By replacing Algorithm **Augment** with **Mod-Augment** we obtain Algorithm **Mod-Dist-α-Greedy** (Algorithm 6). Note that **Mod-Augment** is a drop-in replacement in the

Alg. 6 Mod-Dist-α-Greedy (C, S, E, d, p, cap)

```
x ← 0
for ℓ = W' downto W do
    x ← Mod-Augment(C, S, E, cap, ℓ, x)
                                  ▷ stop after O(k^2 ε^{-1} log^2 n) rounds
end for
```

sense that it preserves all significant properties of **Augment**: (i) All subclasses of $C[\ell]$ with heavy clients are considered, (ii) **Mod-Augment** has the same asymptotic runtime as Algorithm **Augment**, and (iii) the subclasses $C[\ell, i]$ are (locally) considered in decreasing order of index i. We proceed to show that in particular Lemmas 10 and 11 remain true for **Mod-Dist-α-Greedy**. As all subsequent results in Sect. 4.2 are mainly derived from these two lemmas, they also remain true.

Lemma 18 *Mod-Dist-α-Greedy terminates with high probability in $O(k^2\varepsilon^{-2}\log^3 n)$ rounds.*

Proof **Mod-Dist-α-Greedy** consists of $O(\varepsilon^{-1}\log n)$ invocations of Algorithm **Mod-Augment**, which in turn requires $O(k^2\varepsilon^{-1}\log^2 n)$ rounds according to Lemma 17. □

Lemma 19 *Given a k-SA instance, Algorithm **Mod-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 **Mod-Dist-α-Greedy** considers $C_{W'}, \ldots, C_W$ in a decreasing order of ℓ. Since all clients in $C[\ell]$, for any 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 **Mod-Augment** (locally) considers subclasses $C[\ell, i]$ in a decreasing order of i, that is unless $i = i_{\ell}^{\max}(s)$, a server waits with the execution of **Uniform-Augment** for class $C[\ell, i]$ until the execution of **Uniform-Augment** for class $C[\ell, i+1]$ is finished. 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 **Mod-Augment** can be seen as trying to service clients with the same cost-effectiveness in an arbitrary order. It follows that **Mod-Dist-α-Greedy** is a specific implementation of α-**Greedy**. Also, by Lemma 17, all heavy clients in each class $C[\ell]$ are considered. □

Acknowledgements We thank Boaz Patt-Shamir for helpful discussions.

References

1. Amzallag, D., Bar-Yehuda, R., Raz, D., Scalosub, G.: Cell selection in 4G cellular networks. IEEE Trans. Mobile Comput. **12**(7), 1443–1455 (2013)
2. Awerbuch, B.: Complexity of network synchronization. J. ACM **32**(4), 804–823 (1985)
3. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM **48**(5), 1069–1090 (2001)
4. Bar-Yehuda, R., Bendel, K., Freund, A., Rawitz, D.: Local ratio: a unified framework for approximation algorithms. ACM Comput. Surv. **36**(4), 422–463 (2004)
5. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Ann. Discret. Math. **25**, 27–46 (1985)
6. Chekuri, C., Khanna, S.: On multidimensional packing problems. SIAM J. Comput. **33**(4), 837–851 (2004)
7. Chekuri, C., Khanna, S.: A polynomial time approximation scheme for the multiple knapsack problem. SIAM J. Comput. **35**(3), 713–728 (2005)
8. Cohen, R., Grebla, G.: Joint scheduling and fast cell selection in OFDMA wireless networks. IEEE/ACM Trans. Netw. **23**(1), 114–125 (2015)
9. Cohen, R., Katzir, L., Raz, D.: An efficient approximation for the generalized assignment problem. Inf. Process. Lett. **100**(4), 162–166 (2006)
10. Dawande, M., Kalagnanam, J., Keskinocak, P., Salman, F.S., Ravi, R.: Approximation algorithms for the multiple knapsack problem with assignment restrictions. J. Comb. Optim. **4**(2), 171–186 (2000)
11. Emek, Y., Halldórsson, M.M., Mansour, Y., Patt-Shamir, B., Radhakrishnan, J., Rawitz, D.: Online set packing. SIAM J. Comput. **41**(4), 728–746 (2012)
12. Erdös, P., Hajnal, A.: On chromatic number of graphs and set-systems. Acta Math. Hung. **17**(1–2), 61–99 (1966)
13. Feige, U., Vondrák, J.: Approximation algorithms for allocation problems: improving the factor of 1-1/e. In: 47th IEEE Annual Symposium on Foundations of Computer Science, pp. 667–676 (2006)
14. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 611–620 (2006)
15. Frieze, A.M., Clarke, M.R.B.: Approximation algorithms for the m-dimensional $0-1$ knapsack problem: worst-case and probabilistic analyses. Eur. J. Oper. Res. **15**, 100–109 (1984)
16. Gurewitz, O., Sandomirsky, Y., Scalosub, G.: Cellular multi-coverage with non-uniform rates. In: INFOCOM, pp. 1330–1338 (2014)
17. Halldórsson, M.M., Köhler, S., Patt-Shamir, B., Rawitz, D.: Distributed backup placement in networks. In: 27th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 274–283 (2015)
18. Håstad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Math. **182**(1), 105–142 (1999)
19. Hazan, E., Safra, S., Schwartz, O.: On the complexity of approximating k-set packing. Comput. Complex. **15**(1), 20–39 (2006)
20. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM **22**(4), 463–468 (1975)
21. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**(4), 1036–1053 (1986)
22. Magazine, M.J., Chern, M.S.: A note on approximation schemes for multidimensional knapsack problems. Math. Oper. Res. **9**(2), 244–247 (1984)
23. Patt-Shamir, B., Rawitz, D., Scalosub, G.: Distributed approximation of cellular coverage. J. Parallel Distrib. Comput. **72**(3), 402–408 (2012)
24. Peleg, D.: Distributed computing: a locality-sensitive approach. Society for Industrial and Applied Mathematics, Philadelphia (2000)

25. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**(4), 365–374 (1987)
26. Rawitz, D., Voloshin, A.: Flexible cell selection in cellular networks. In: 12th International Symposium on Algorithms and Experiments for Wireless Sensor Networks. LNCS, vol. 10050, pp. 112–128 (2016)
27. Sahni, S.: Approximate algorithms for the 0/1 knapsack problem. J. ACM **22**(1), 115–124 (1975)
28. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**, 461–474 (1993)
29. Srinivasan, A.: Improved approximation guarantees for packing and covering integer programs. SIAM J. Comput. **29**(2), 648–670 (1999)
30. Wattenhofer, R.: Principles of distributed computing: maximal independent set. http://www.dcg.ethz.ch/lectures/fs15/podc/lecture/chapter7.pdf. Accessed 27 Aug 2015