SIAM J. OPTIM.
Vol. 30, No. 1, pp. 885–914

COMPLEXITY AND APPROXIMABILITY OF OPTIMAL RESOURCE ALLOCATION AND NASH EQUILIBRIUM OVER NETWORKS*

S. RASOUL ETESAMI†

Abstract. Motivated by emerging resource allocation and data placement problems such as web caches and peer-to-peer systems, we consider and study a class of resource allocation problems over a network of agents (nodes). In this model, which can be viewed as a homogeneous data placement problem, nodes can store only a limited number of resources while accessing the remaining ones through their closest neighbors. We consider this problem under both optimization and game-theoretic frameworks. In the case of optimal resource allocation, we will first show that when there are only $k = 2$ resources, the optimal allocation can be found efficiently in $O(n^2 \log n)$ steps, where n denotes the total number of nodes. However, for $k \geq 3$ this problem becomes NP-hard with no polynomial-time approximation algorithm with a performance guarantee better than $1 + \frac{1}{102k^2}$, even under metric access costs. We then provide a 3-approximation algorithm for the optimal resource allocation which runs only in $O(kn^2)$. Subsequently, we look at this problem under a selfish setting formulated as a noncooperative game and provide a 3-approximation algorithm for obtaining its pure Nash equilibria under metric access costs. We then establish an equivalence between the set of pure Nash equilibria and flip-optimal solutions of the Max-k-Cut problem over a specific *weighted* complete graph. While this reduction suggests that finding a pure Nash equilibrium using best response dynamics might be PLS-hard, it allows us to use tools from complementary slackness and quadratic programming to devise systematic and more efficient algorithms towards obtaining Nash equilibrium points.

Key words. resource allocation, data placement problem, facility location, network games, potential games, approximation algorithm, quadratic programming, computational complexity

AMS subject classifications. 91B32, 91A24, 91A43, 90C-xx

DOI. 10.1137/19M1242525

1. Introduction.

Resource allocation problems have always been challenging and widely studied by many researchers from multiple disciplines. Broadly speaking, in most of the resource allocation problems the goal is to allocate a limited number of resources to a set of agents to optimize a specific objective function, such as the overall access cost, network reliability, or resource availability. More particularly, depending on what constraints are to be satisfied at an optimal allocation, one may expect to see a wide range of results both in terms of computational complexity and solution structure. For instance, the solution structure can completely change if the entire allocation system is operated by a central authority or by the individual agents themselves.

In this regard, perhaps *facility location* problems are one of the most classic resource allocation problems which were extensively studied in the past literature [1, 32, 45, 26, 24, 43]. In simple words, an instance of the facility location problem is identified by a set of clients and a set of potential facilities. Opening each facility has a certain fixed cost, and clients may incur different costs when accessing different facilities. The goal is to open a subset of facilities and assign each client to an

*Received by the editors February 4, 2019; accepted for publication (in revised form) December 23, 2019; published electronically March 12, 2020.
https://doi.org/10.1137/19M1242525
Funding: This work was supported in part by NSF Career Award, EPCN-1944403.
†Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (etesami1@illinois.edu).

open facility in order to minimize the total facility opening costs and clients' access costs. In particular, depending on whether or not the facilities can serve a limited number of clients, one can distinguish between *capacitated* [31] and *uncapacitated* [26] alternatives.

In fact, one can view the facility location problem as a special case of a *single* resource allocation problem by viewing open facilities as copies of a single resource which are distributed in different locations to satisfy clients' demands at minimum cost. However, often each facility can provide only one type of service, while a client may need multiple service types (i.e., it must obtain access to different facilities). This can be viewed as a multiresource allocation problem (one resource for each facility type) where the goal is to minimize the access cost of all of the nodes to all of the resources. We refer readers to [15, 45, 17, 24, 43, 39] for several variants of multitype facility location problems with the main focus being on approximating the optimal allocations. In addition, in some facility location problems the set of potential facilities and clients are determined a priori, meaning that if an agent is determined as a client, it cannot serve as a facility or vice versa. On the other hand, motivated by the real emerging distributed system such as web caches or ad-hoc video streaming networks where a user/agent/node can act both as a facility or a client, a major question is what type of resources each node must provide so that the resource accessibility cost over the entire network is minimized.

The resource allocation problem we consider in this paper can be viewed as a multitype facility location problem in which nodes can simultaneously provide and request a service (resource). More specifically, we consider a network of agents where each agent has a limited cache size and can store only a small number of resources. However, each agent must obtain access to all of the resources. If a resource is available in an agent's cache, it incurs no extra cost for that specific resource. Otherwise, it incurs a cost for obtaining each missing resource from the closest neighbor who has that resource. We consider this problem from two different perspectives, namely optimal allocation and selfish (game) allocation. In the optimal allocation setting, we are interested to know how to assign resources to the agents in a centralized manner to minimize the overall access costs. In the game-theoretic setting, we want to determine whether the entire system converges to any stable equilibrium, provided that each agent selfishly caches its favorite resources, and we attempt to compute one such equilibrium point.

As an application of an optimal allocation setting, one can consider a data placement problem over a network of servers (service providers). Each node can be viewed as a server that can only store a limited amount of data in its cache. The communication network between service providers determines the access costs where the farther the servers are located from each other, the more delay costs they incur in obtaining access to each other. When a client refers to a server and requests data, if that server has the requested data in its cache, it can immediately provide that data to its client without any time delay. Otherwise, the requested data must be accessed through communication with the closest server possessing that data, which incurs some delay cost. Therefore, the central authority who operates the entire data center aims to minimize the overall delay costs by appropriately assigning data to different servers, taking into account the communication costs among them. In particular, when all data are equally popular to the clients or clients' request rates are unknown and can be chosen adversarially, the goal for the central planner is to minimize the overall access cost under worst-case request rates by making sure that all data are available to all nodes. Now, under the game-theoretic setting, one can address the same question

when the servers act selfishly and individually. Perhaps, in this case, one can consider a peer-to-peer data sharing system (e.g., movie sharing service) where the nodes are individuals who only care about minimizing their delay costs by storing appropriate data (movies) and accessing the remaining ones through others. Here, to ensure that all the data will be available one can define the cost of an individual not having access to a particular movie to be very large so that at least one copy of each movie is stored by an individual.

1.1. Related works. Facility location problems have been extensively studied in both engineering and computer science literature [1, 32, 26, 24, 43]. It is well known that the facility location problem, even in its simplest form i.e., uncapacitated with metric access costs, is NP-hard [1]. Therefore, a huge effort in the past few decades has been devoted to devising fast heuristic algorithms to deliver good suboptimal solutions. For instance, the sequence of works [32, 26, 43] provide different approximation algorithms to improve the best-known approximation factors. It has been shown that for the metric facility location problem finding a polynomial-time algorithm with an approximation guarantee better than 1.463 is NP-hard [22]. Given that our goal here is not to provide an exhaustive review of the past literature on the facility location problem (and its different variants, such as the p-median problem [26]), we refer interested readers to [15, 34] for comprehensive surveys.

The resource allocation problem under the optimization setting that we consider in this paper was first introduced in [4], where the authors provided a 20.5-approximation algorithm for the optimal resource placement with general heterogeneous request rates (i.e., when different items have different request rates). In a subsequent work by the same authors [3], this approximation ratio has been further improved to 10. The main approach in both of these works was based on the clustering technique of [9] adopted for the linear programming relaxation of the optimal allocation problem and then rounding the solution using a network flow problem. In particular, it was left open in [4] the question of whether there exists a simple local search or greedy algorithm for obtaining the optimal allocation with a good performance guarantee. In this work, we answer this question by providing the first combinatorial algorithm that substantially improves the earlier approximation factors to 3 under a *homogeneous* setting (i.e., when resources have identical installation costs and request rates). It is worth noting that if one considers a more restricted version of this problem in which agents can satisfy at most a limited number of their resource requests, then the optimal solution cannot be approximated within a constant factor unless the cache capacities are violated by a constant factor [23].

It is worth noting that although the heterogeneous resource allocation problem is a generalization of the homogeneous setting, in this paper, however, we mainly focus on the homogeneous case for three main reasons: (i) It is shown in section 2.1 that the homogeneous resource allocation problem with unit cache size can be viewed as a generalization of the well-known p-median problem [26]. (ii) The approximation algorithm that we develop for the homogeneous case works beyond the offline setting and provides a 3-competitive algorithm under an online setting where the nodes and their access costs are revealed over time (Remark 1). This is particularly important in recent data placement problems where based on the demand the network size dynamically grows over time. (iii) Unlike the homogeneous case, the heterogeneous resource allocation game does not necessarily admit a pure Nash equilibrium (NE), even under metric access costs (Example 5.1). Nevertheless, since the heterogeneous resource allocation problem is still very interesting, in section 8, we provide new

insights and future directions on how one may extend our results to obtain improved approximation ratios for the heterogeneous resource allocation problem.

On the other hand, resource allocation problems have been extensively studied from a game-theoretic perspective [37, 20, 12, 33, 13]. Typically, resource allocation games are identified by a set of available resources for each player, where the players are allowed to access each other through a communication network. Such a communication network determines the access cost among the players, and the goal for each player is to satisfy his/her customers' needs at a minimum cost. In fact, it has been observed in practice that resource allocation games often provide good resource distribution in terms of availability and reliability for the users [10, 21]. Resource allocation games arise in a wide variety of contexts, such as congestion games [35, 2, 14], load balancing [18], peer-to-peer systems [38], web caches [20], content management [38], and market sharing games [19]. Distributed replication games with servers that have access to all of the resources and are accessible at some cost by users have been studied in [30]. Moreover, the *uncapacitated* selfish resource allocation game where the agents have access to the set of all of the resources with no constraint on their cache size was studied in [10]. However, the resource allocation game that we consider in this paper is more complicated than the one in [10], as the constraints on cache sizes couple the strategic actions of the players in a much more complex form.

Another class of resource allocation games that are closely related to our work are *graph coloring* games [8, 29]. In graph coloring games, the nodes of an undirected/directed graph are the players, the strategy set of each node consists of the k colors, and the payoff of a node v in a given state or coloring is given by the number of outgoing neighbors with a color different from the one of v. It has been shown in [8] that coloring games on undirected graphs belong to the class of potential games [36] and hence have a pure NE. In particular, coloring games admit an exact potential function which allows us to find a pure NE in polynomial time. However, this is quite different from our setting, in which only the existence of a much weaker potential function, namely an *ordinal* potential function, is known, while the complexity of finding a pure NE is still open.

In this paper, we consider a class of resource allocation games that were first introduced and studied in [20]. In particular, it was shown that such games always admit a pure NE which can be found in polynomial time when there are only $k = 2$ resources. However, for $k \geq 3$ resources, it was conjectured in [20] that finding a NE, in general, is PLS-hard. In this work, we substantially expand upon their results by taking the first step toward settling this conjecture. This is done by establishing an equivalence between the set of NE points and the set of *flip-optimal* solutions of the Max-k-Cut problem with a *specific* weight structure. As finding a flip-optimal solution to the Max-k-Cut problem with *arbitrary* weights is known to be PLS-hard [41], this suggests that adopting similar reduction techniques as in [41] to our specific weight structure, one might be able to establish PLS-hardness of finding a NE in the resource allocation game.

1.2. Organization and contributions. In section 2, we first introduce the problem of optimal resource allocation, provide some of its salient properties, and discuss its similarities to facility location problems. We then study the computational complexity of finding an optimal allocation in section 3, where we show that this problem for the case of $k = 2$ can be solved efficiently without any assumption regarding the access costs. However, for $k \geq 3$, we show that it does not even admit a polynomial-time approximation algorithm within a factor better than $1+\frac{1}{102k^2}$ (unless

P = NP). In particular, we show that finding an optimal allocation with heterogeneous request rates is NP-hard even when there are only $k = 2$ resources. In section 4, we provide a greedy 3-approximation algorithm for the optimal allocation assuming metric access costs and homogeneous request rates. In section 5, we turn our attention to the resource allocation problem under the game-theoretic setting and show that the aforementioned greedy algorithm also delivers a 3-approximate NE under metric access costs. We then provide several complexity results regarding computing NE points and establish a connection between NE points and *flip-optimal* solutions of the weighted Max-k-Cut problem. Leveraging our reduction to the weighted Max-k-Cut, in section 6, we provide two new algorithms based on quadratic programming to compute NE points more efficiently. We conclude the paper by some discussion and identifying several future directions of research in section 8.

Notations. Throughout the paper, we adopt the following notations: For a positive integer n, we let $[n] := \{1, 2, \ldots, n\}$. We use $\mathcal{G} = (V, E)$ for an undirected graph with node set V and edge set E. We use $|S|$ to denote the cardinality of a finite set S. Given a matrix A, we denote its transpose by A^T, its ith row by A_i, and its ijth entry by $[A]_{ij}$. We sometimes refer to entries of a matrix by lowercase characters, i.e., $[A]_{ij} = a_{ij}$. We denote the Frobenius norm of A by $\|A\|$ and its trace by $\mathrm{tr}(A)$. A nonnegative matrix A is called stochastic if it has row sum 1 for every row. Given a vector v, we denote all of its entries except the ith one by v_{-i}. Finally, we use $\langle , \rangle$ for the inner product of two vectors and e_i to denote the ith basic unit vector in $\mathbb{R}^k$.

2. Optimal resource allocation. Let us consider a set of $[n] = \{1, 2, \ldots, n\}$ nodes and a set of k distinct resources $O = \{o_1, o_2, \ldots, o_k\}$. The access cost between every pair of nodes $i, j \in [n]$ is given by a nonnegative real number c_{ij}. In this section, we assume that the access costs satisfy the metric property, i.e.,

- $c_{ii} = 0 \ \forall i \in [n]$,
- $c_{ij} = c_{ji} \ \forall i, j$,
- triangle inequality: $c_{ij} + c_{jr} \geq c_{ir} \ \forall i, j, r \in [n]$.

We assume that each node i has a limited cache size $u_i \in \mathbb{Z}^+$ and can hold at most u_i resources in its cache, where different nodes may have different cache sizes. As a result, we can represent a feasible resource allocation profile by $P = (P_1, P_2, \ldots, P_n)$, where $P_i \subseteq O, |P_i| = u_i$ represents the set of resources held by node i.[1] Given an allocation profile P, the access cost for node i is given by

$$C_i(P) := \sum_{\ell \in O} c_{i,i(\ell)}, \tag{1}$$

where $i(\ell)$ denotes agent i's nearest node holding resource ℓ in the allocation profile P, i.e, $i(\ell) = \arg\min_j \{c_{ij} : \ell \in P_j\}$. If there does not exist such an $i(\ell)$, we simply define the cost of getting access to resource ℓ by node i to be very large so that every resource is assigned to at least one node. Also, note that without loss of generality we may assume that each node fills it cache since the model does not assume any resource storage cost. Finally, the goal is to find an allocation profile P which minimizes the total sum of the access costs over all the nodes, i.e.,

$$\min_{P \in O^{u_1} \times \cdots \times O^{u_n}} \sum_{i=1}^{n} C_i(P), \tag{2}$$

[1] Throughout this paper, we interchangeably use the terms caching/allocating/assigning a resource to a node.

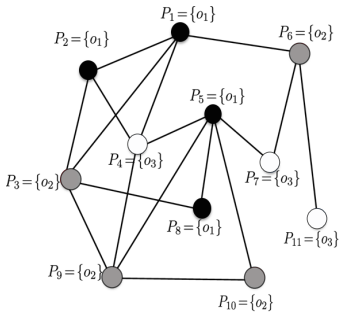

FIG. 1. *An instance of a feasible resource allocation P with $n = 11$ nodes and $k = 3$ resources. Different resources $O = \{o_1, o_2, o_3\}$ are denoted by different colors. Here, it is assumed that all the nodes have unit cache size and the access costs are determined by the graphical distances. For instance, we have $C_1(P) = 0 + 1 + 1 = 2$, and $C_8(P) = 0 + 1 + 2 = 3$.*

where O^u denotes the Cartesian product of O by itself, u times.

An instance of a feasible resource allocation for $n = 11$ nodes, $k = 3$ resources, and unit cache size $u_i = 1\ \forall i$ is illustrated in Figure 1. Here we are simply assuming that the access costs are induced by graphical distances and the cost of traversing an edge equals 1. For instance, node 8 incurs zero cost to access resource o_1 (since this resource is available in its cache) and incurs one and two units of costs to access resources o_2 and o_3, respectively. It is worth noting that in the case of *unit* cache size (i.e., $u_i = 1\ \forall i$), if $n \le k$, all the nodes will allocate distinct resources and the problem becomes trivial. Henceforth whenever we consider resource allocation with unit cache size we simply assume $n > k$.

2.1. Optimal resource allocation versus *p*-median problem. Before we get into the analysis of optimal allocation, in the following we discuss some of the similarities and differences between the optimal resource allocation problem given in (2) and a variant of the uncapacitated facility location problem known as the *p-median* problem [26]. Given a set of potential facilities N and a set of clients M, in the p-median problem one aims to open a subset $Q \subseteq N$ of at most p facilities such that the total facility access cost over all of the clients is minimized. This optimization problem can be written as

$$\min_{Q \subset N : |Q| \le p} \sum_{i \in M} \min_{j \in Q} c_{ij}, \tag{3}$$

where c_{ij} denotes the cost of getting access to facility i by client j. Now let us consider the resource allocation problem (2) under the special case of *unit* cache size (i.e., $u_i = 1\ \forall i$) and reformulate it as an optimization problem of the form (3). For this purpose, let V^ℓ denote the set of nodes which allocate resource ℓ, where we note that $V^\ell \neq \emptyset$, since by the definition of the access costs every resource is assigned to at least one node (or else the total cost becomes very large). Moreover, let us define $\mathcal{V}$ to be the set of all nonempty k-partitions of the set $[n]$, i.e.,

$$\mathcal{V} := \{(V^1, \ldots, V^k) : \cup_{\ell=1}^k V^\ell = [n], V^\ell \cap V^{\ell'} = \emptyset\ \forall \ell \neq \ell'\}.$$

Then the total cost of resource allocation $(V^1, \ldots, V^k) \in \mathcal{V}$ is given by

$$\sum_{i=1}^{n} \left(\min_{j \in V^1} c_{ij} + \cdots + \min_{j \in V^k} c_{ij} \right).$$

Therefore, finding the optimal resource allocation in (2) under unit cache size is equivalent to solving the following optimization problem:

$$\min_{(V^1,\ldots,V^k) \in \mathcal{V}} \sum_{i \in M} \left(\min_{j \in V^1} c_{ij} + \cdots + \min_{j \in V^k} c_{ij} \right). \tag{4}$$

Comparing (4) and (3), one can see a close relationship between the p-median problem and the optimal resource allocation problem. In fact, the resource allocation problem can be viewed as a multitype p-median problem where instead of optimizing over only one set Q, we are optimizing over multiple sets $(V^1, \ldots, V^k)$. This makes the analysis much more complicated, as now we have to deal with different types of resources (facilities) which are highly coupled. In other words, we cannot view the resource allocation problem as k separate p-median problems in isolation. This is because assigning a resource ℓ to a node i (opening a facility to provide service ℓ) completely changes the supply for other resources (the need for opening other types of facilities). Moreover, in the optimal resource allocation problem the set of nodes that can allocate resource ℓ is itself subject to design. While these extra design features complicate the analysis of the optimal resource allocation, from the computational complexity perspective it relaxes some of the constraints in the p-median problem (such as opening only p facilities).

It is worth noting that the optimal resource allocation problem (2) can also be cast as an integer linear program. For this purpose, let us define $x_{ij}^{\ell} = 1$ if node j requests resource ℓ from node i, and $x_{ij}^{\ell} = 0$ otherwise. Moreover, let $y_i^{\ell} = 1$ if node i allocates resource ℓ, and $y_i^{\ell} = 0$ otherwise. Now the set of all feasible resource assignments is given by the following constraints:

- $x_{ij}^{\ell} \le y_i^{\ell}$ $\forall i, j \in [n]$, $\ell \in [k]$; i.e., node j can only request resource ℓ from node i if node i has this resource in its cache.
- $\sum_{\ell=1}^{k} y_i^{\ell} = u_i$ $\forall i \in [n]$; i.e., node i can allocate exactly u_i resources.
- $\sum_{i=1}^{n} x_{ij}^{\ell} = 1$ $\forall \ell, j$; i.e., each node j receives resource ℓ from some node i.

Therefore, the homogeneous optimal resource allocation (2) can be formulated as

$$\begin{aligned}
\min & \sum_{i,j=1}^{n} \sum_{\ell=1}^{k} c_{ij} x_{ij}^{\ell}, \\
& x_{ij}^{\ell} \le y_i^{\ell} \quad \forall i, j \in [n],\ \ell \in [k], \\
& \sum_{\ell=1}^{k} y_i^{\ell} = u_i \quad \forall i \in [n], \\
& \sum_{i=1}^{n} x_{ij}^{\ell} = 1 \ \forall \ell \in [k],\ j \in [n], \\
& x_{ij}^{\ell}, y_i^{\ell} \in \{0, 1\}.
\end{aligned}$$

In what follows, we focus on the optimal resource allocation problem (2) as a k-level extension of the p-median problem and study the complexity and approximability of its optimal solutions.

3. Complexity and approximability of the optimal allocation. In this section, we first consider the optimal resource allocation with only $k = 2$ resources and then study the complexity of computing an optimal allocation for $k \geq 3$ resources. In particular, we establish our complexity results for the special case where all the nodes have a unit cache size, which then immediately applies to the general case of arbitrary cache sizes. In the following theorem, we show that when the number of resources is $k = 2$, then a simple algorithm can find an optimal allocation in no more than $O(n^2 \log n)$ steps.

THEOREM 3.1. *The optimal resource allocation with $k = 2$ resources can be found in $O(n^2 \log n)$ steps, where n is the total number of nodes.*

Proof. Let us view the nodes in the resource allocation problem as nodes of a weighted complete graph where the weight of edge $\{i, j\}$ is given by the access cost c_{ij}. We sort the edges of this complete graph in nondecreasing weight order. This can be done in $O(n^2 \log n)$, which is the dominant stage in our algorithm. At iteration $t = 1, 2, \ldots$, we select the lowest weight edge which spans at least one more node (i.e., at least one of its endpoints has not yet been assigned a resource). If none of the endpoints of the selected edge is assigned a resource, we arbitrarily assign resource 1 to one of them and resource 2 to the other one. But if one of its endpoints was already assigned a resource, we assign to the other endpoint an opposite resource. Following this process, a simple induction shows that the cost of an arbitrary node i at the termination is given by $C_i = \min_{j \neq i} c_{ij}$, which is the best possible that can be achieved by node i. Thus the allocation profile obtained at the termination will be an optimal allocation. ∎

It is worth noting that the proof of Theorem 3.1 does not use the metric property of the costs and works under general symmetric access costs. Unfortunately, if we increase the number of resources from $k = 2$ to $k = 3$, the problem becomes much more complicated. This has been shown formally in the following theorem.

THEOREM 3.2. *It is NP-hard to find an optimal allocation with $k \geq 3$ resources, even under metric access costs and unit cache size.*

Proof. The proof is by reduction from the Max-k-Cut problem, which is known to be NP-hard [16]. An instance of the Max-k-Cut problem is given by an arbitrary undirected connected graph $\mathcal{G} = (V, E)$, and the goal is to find a partition $(V_1, \ldots, V_k)$ of the vertices V with the maximum number of crossing edges, i.e., $\max_{(V_1,\ldots,V_k)} |\delta(V_1, \ldots, V_k)|$, where

$$\delta(V_1, \ldots, V_k) := \{e = (u, v) \in E : u \in V_i, v \in V_j \text{ for } i \neq j\}.$$

Given $\mathcal{G} = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$, we construct an instance of the resource allocation problem in the form of a weighted complete graph $\mathcal{K}$ with $n + m$ nodes $v_1, \ldots . v_n, e_1, \ldots, e_m$, where the weight of every edge (v_i, v_j) equals 1, and the weight of every edge (e_i, e_j) equals 2. Moreover, we set the weight of an edge (v_i, e_j) as being 1 if v_i is an endpoint of e_j in the graph $\mathcal{G}$, and as being 2 otherwise (see Figure 2). Thus, $\mathcal{K}$ is a complete graph with all of the edges having weights of either 1 or 2. Clearly by this construction, the edge weights of this complete graph $\mathcal{K}$ (i.e., the access costs) satisfy the triangle inequality and hence are metric. We claim that finding an optimal allocation with k resources on $\mathcal{K}$ is equivalent to solving the Max-k-Cut problem on $\mathcal{G}$.

Consider an arbitrary k-resource allocation on $\mathcal{K}$, and for any $i \in [k]$ let (V_i, E_i) be the nodes which are allocated resource i. Note that $(V_1, \ldots, V_k)$ and $(E_1, \ldots, E_k)$ form

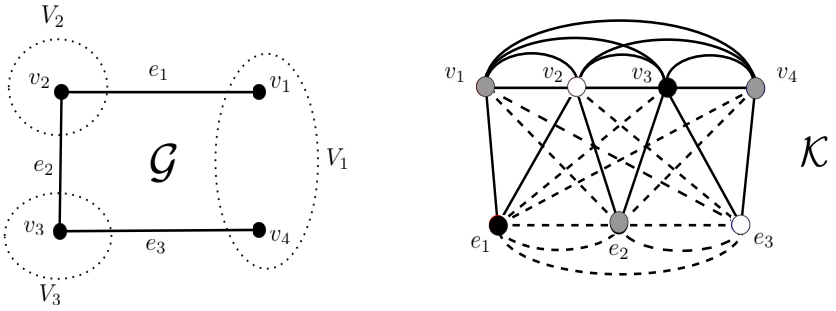

FIG. 2. *An illustration of the reduction in the proof of Theorem* 3.2. *The original graph* $\mathcal{G}$ *with four vertices and three edges is given on the left. The complete graph* $\mathcal{K}$ *with seven nodes is given on the right. Each solid edge in* $\mathcal{K}$ *has weight* 1*, and each dashed edge has weight* 2*.* V_1, V_2*, and* V_3 *denote a Max-3-Cut on* $\mathcal{G}$*. Equivalently, the resource allocation on the right figure with three resources (white, gray, and black) is an optimal allocation on* $\mathcal{K}$*. In this example,* $E_1 = \{e_2\}$*,* $E_2 = \{e_3\}$*, and* $E_3 = \{e_1\}$*. In particular,* (V_1, E_1)*,* (V_2, E_2)*, and* (V_3, E_3) *denote the optimal resource classes for gray, white, and black resources, respectively.*

a partition of V and E, respectively. Let $E[V_i] = \{e = (u, v) \in E : u \in V_i, v \in V_i\}$ be the set of all of the edges with both endpoints in V_i. Now for any $e \in E[V_i]$, $i \in [k]$, the cost of e for obtaining access to all of the resources is at least $2k - 3$. This is because if $e \in E[V_i]$, then both endpoints of e belong to V_i. Therefore, in the best case, even if $e \in E_j$ for some $j \neq i$, the cost of accessing all the resources for e is at least $0 + 1 + 2(k - 2) = 2k - 3$, a cost of 1 for accessing the resource in i, and a cost of 2 for accessing other $k - 2$ resources in (V_ℓ, E_ℓ), $\ell \neq i, j$. Moreover, every vertex in V incurs a cost of at least $k - 1$, as it has to access $k - 1$ different resources at distance at least 1 from it. In addition, every $e \in E \setminus \cup_{i=1}^{k} E[V_i]$ incurs a cost of at least $0 + 1 + 1 + 2(k - 3) = 2k - 4$, due to the fact that e has exactly two endpoints in V, meaning that in the complete graph $\mathcal{K}$ exactly two of the links adjacent to e have weight 1 (and all others have weight 2). Therefore, the cost of this allocation is at least

$$\begin{aligned}
&(2k-3)|\cup_{i=1}^{k} E[V_i]| + (k-1)n + (2k-4)(m - |\cup_{i=1}^{k} E[V_i]|) \\
&\quad = (k-1)n + (2k-4)m + \sum_{i=1}^{k} |E[V_i]| \\
&\quad = (k-1)n + (2k-3)m - |\delta(V_1, \ldots, V_k)|, \qquad (5)
\end{aligned}$$

where in the last equality we have used $\sum_{i=1}^{k} |E[V_i]| = m - |\delta(V_1, \ldots, V_k)|$.

Next, given a k-cut $(V_1, \ldots, V_k)$ on $\mathcal{G}$ with $V_i \neq \emptyset$ $\forall i \in [k]$, we build a solution to the resource allocation problem on $\mathcal{K}$. For this purpose, let $\delta(V_i, V_j) = \{e = (u, v) \in E : u \in V_i, v \in V_j\}$, and consider the allocation partition (V_i, E_i), $i \in [k]$:

$$\begin{aligned}
&E_1 = E[V_2] \cup \Big(\cup_{i \neq j \neq 1} \delta(V_i, V_j) \Big), \\
&E_i = E[V_{i+1}] \cup \delta(V_1, V_{i+1}) \text{ for } i = 2, \ldots, k-1, \\
&E_k = E[V_1] \cup \delta(V_1, V_2). \qquad (6)
\end{aligned}$$

Now one can easily check that in this allocation each vertex $v \in V$ has a cost of $k - 1$, each $e \in E[V_i]$, $i \in [k]$, has a cost of $2k - 3$, and each $e \in \delta(V_i, V_j)$, $i \neq j$, has a cost of $2k - 4$. This means that the lower bound in (5) is achieved for the resource allocation

sets (V_i, E_i), $i \in [k]$, where E_i, $i \in [k]$, are defined by (6). This shows that an optimal resource allocation is given by $\arg\min_{(V_1,\dots,V_k)}(k-1)n + (2k-3)m - |\delta(V_1,\dots,V_k)|$. Since $(k-1)n + (2k-3)m$ is constant, finding an optimal allocation is equivalent to finding a solution to the Max-k-Cut on $\mathcal{G}$.

Finally, in the above analysis we have assumed that $V_i \neq \emptyset$ $\forall i \in [k]$. This, however, can be assumed without any loss of generality, as if only $V_1,\dots,V_p \neq \emptyset$ for some $p < k$, then the lower bound in (5) on the minimum cost would become $(k-1)n + (2k-3)m - |\delta(V_1,\dots,V_p)|$. As the optimal value of the Max-p-Cut is no more than that of the Max-k-Cut, the lower bound in (5) can only increase. Hence we may assume $V_i \neq \emptyset$ $\forall i \in [k]$, and this completes the proof. ∎

Theorem 3.1 implies that the optimal allocation problem with $k = 2$ resources can be solved in polynomial time, while Theorem 3.2 states that this problem for $k \geq 3$ resources is NP-hard. However, the reduction used in Theorem 3.2 is based on Max-k-Cut, which is still NP-hard even when $k = 2$. Although this looks contradictory in view of Theorem 3.1, it appears, however, that the Max-k-Cut reduction used in the proof of Theorem 3.2 fails to hold when $k = 2$. This is because in the first part of the proof, the lower bound for the cost of each edge $e \in E \setminus \cup_{i=1}^{k} E[V_i]$ becomes $0+1$ (rather than $0+1+1+2(k-3) = 0$), which strictly increases the lower bound obtained in (5). Moreover, the construction of the allocation partition given in (6) is no longer feasible when $k = 2$, which again hinders the use of the Max-2-Cut reduction. Next, in the following theorem, we show that for $k \geq 3$ even approximating the optimal resource allocation better than a certain constant factor is a hard problem.

THEOREM 3.3. *Unless* P = NP, *for* $k \geq 3$ *there is no polynomial time approximation algorithm with performance guarantee better than* $1 + \frac{1}{102k^2}$ *for the optimal resource allocation with metric access costs.*

Proof. It has been shown in [27] that if P ≠ NP, no polynomial time approximation algorithm can achieve an approximation factor better than $1 - \frac{1}{34k}$ for the Max-k-Cut problem. Given an arbitrary undirected graph $\mathcal{G}$ with n nodes and m edges, let us consider the same reduction as the one given in the proof of Theorem 3.2. Now to derive a contradiction, assume that the optimal resource allocation with k resources admits a polynomial time $(1 + \frac{1}{102k^2})$-approximation algorithm. Let $(k-1)n + (2k-3)m - |\delta(\tilde{\boldsymbol{V}})|$ be the overall cost of the solution generated by this approximation algorithm when it is applied on $\mathcal{K}$, where $\tilde{\boldsymbol{V}} = (\tilde{V}_1,\dots,\tilde{V}_k)$ denotes the set of vertex partitions generated by the approximation algorithm. Therefore, if we denote a solution to Max-k-Cut on $\mathcal{G}$ by $\boldsymbol{V}^o = (V_1^o,\dots,V_k^o)$, we must have

$$(k-1)n + (2k-3)m - |\delta(\tilde{\boldsymbol{V}})| \leq \left(1 + \frac{1}{102k^2}\right)[(k-1)n + (2k-3)m - |\delta(\boldsymbol{V}^o)|].$$

By rearranging the terms, we obtain

$$\begin{aligned}
|\delta(\tilde{\boldsymbol{V}})| &\geq \left(1 + \frac{1}{102k^2}\right)|\delta(\boldsymbol{V}^o)| - \frac{1}{102k^2}[(k-1)n + (2k-3)m] \\
&\geq \left(1 - \frac{1}{34k}\right)|\delta(\boldsymbol{V}^o)| + \frac{1}{34k}|\delta(\boldsymbol{V}^o)| - \frac{1}{102k^2}[(k-1)n + (2k-3)m] \\
&\geq \left(1 - \frac{1}{34k}\right)|\delta(\boldsymbol{V}^o)| + \frac{1}{34k}\left(1 - \frac{1}{k}\right)m - \frac{3(k-1)}{102k^2}m \\
&= \left(1 - \frac{1}{34k}\right)|\delta(\boldsymbol{V}^o)|, \qquad (7)
\end{aligned}$$

where in the last inequality we have used the fact that $m \geq n$, and $|\delta(\boldsymbol{V}^o)| \geq (1-\frac{1}{k})m$. Note that the bound $|\delta(\boldsymbol{V}^o)| \geq (1-\frac{1}{k})m$ can be obtained using a standard probabilistic argument: choose each vertex in $\mathcal{G}$ to be in one of the k partition sets with probability $\frac{1}{k}$. Then the expected number of edges in the cut is $(1-\frac{1}{k})m$, and hence the size of the optimal solution to the Max-k-Cut is at least $(1-\frac{1}{k})m$. But (7) implies that the Max-k-Cut can be approximated within a factor of $1-\frac{1}{34k}$, which is a contradiction. ∎

3.1. Complexity of optimal allocation with heterogeneous rates. We complete the results of this section by providing another complexity result which essentially shows that if we consider the optimal resource allocation (2) with *heterogeneous* request rates, i.e,

$$\min_{P \in O^{u_1} \times \cdots \times O^{u_n}} \sum_{i=1}^{n} \sum_{\ell \in O} \omega_i(o_\ell) c_{i,i(\ell)},$$

where $\omega_i(o_\ell) \geq 0$ is the request rate received by agent i for the resource o_ℓ, then as opposed to Theorem 3.2, finding an optimal allocation with even $k = 2$ becomes NP-hard. Again, we establish this result for the special case of unit cache size which immediately applies to arbitrary cache sizes.

THEOREM 3.4. *It is NP-hard to find the optimal resource allocation with heterogeneous request rates even for $k = 2$ resources and metric access costs.*

Proof. We use a reduction from the vertex cover problem. An instance of the vertex cover problem is given by an arbitrary undirected graph $\mathcal{G} = (V, E)$, and the goal is to find a minimum cardinality subset of vertices which covers all of the edges (i.e., each edge has at least one endpoint in that subset). It is known that this problem is NP-hard. Given an arbitrary instance of the vertex cover $\mathcal{G} = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$, let $\mathcal{K}$ be the same weighted complete graph with $n+m$ nodes as in the proof of Theorem 3.2. Moreover, let us denote the available resources by $O = \{o_1, o_2\}$, and assume all of the vertices $v \in V$ have request rates $w_v(o_1) = \frac{3}{5}$ and $w_v(o_2) = \frac{2}{5}$ for the resources o_1 and o_2, respectively. In addition, we let all the nodes $e \in E$ have request rates $w_e(o_1) = \frac{3}{4}$ and $w_e(o_2) = \frac{1}{4}$. We will show that finding an optimal allocation with these request rates on $\mathcal{K}$ is equivalent to finding the minimum vertex cover on $\mathcal{G}$.

First, we note that in an optimal allocation at least one of the nodes in V must cache o_2. Otherwise, let $e \in E$ be a node holding o_2,[2] and consider one of its endpoints $v \in V$. Then by assigning o_1 to e and o_2 to v, the cost of all of the other nodes in $(V \setminus \{v\}) \cup (E \setminus \{e\})$ will not increase. This is the case because by this change all of the nodes in $V \setminus \{v\}$ are still within a distance of 1 from resource o_2. Moreover, all the nodes in $E \setminus \{e\}$ are within a distance of 1 from resource o_1 (as each edge in E has two endpoints) and within at most the same distance as before from resource o_2. By this change, the total cost of nodes v and e becomes $\frac{3}{5} + \frac{1}{4} = \frac{17}{20}$, while before this change it was $\frac{2}{5} + \frac{3}{4} = \frac{23}{20}$. Since the former is strictly less than the latter, this contradicts the optimality of the initial allocation. Similarly, one can show that at least one of the nodes in V must cache o_1. Otherwise, if all of the nodes in V are holding resource o_2, then by choosing an arbitrary node $v \in V$ and assigning resource o_1 to it, the costs of all of the other $n + m - 1$ nodes do not increase, while the cost of node v decreases by at least $\frac{3}{5} - \frac{2}{5} = \frac{1}{5}$, which is a contradiction.

[2]Note that at least one node has the resource o_2; otherwise, the allocation cost will be infinity.

Next, we claim that in an optimal allocation all of the nodes in E must cache resource o_1. Otherwise, assume that $i \geq 1$ of the nodes in E have resource o_2 and the remaining $m-i$ nodes have resource o_1. Since both resources appear at least once among the nodes in V and the access costs between these nodes is 1, the access costs of the nodes in V are fully determined by themselves regardless of resource allocation in E. Therefore, if we switch o_2 to o_1 in all the i nodes in E, the cost of the initial (optimal) allocation minus that of the new allocation would be at least $i \times \frac{3}{4} \times 1 - i \times \frac{1}{4} \times 2 > 0$. This again contradicts the optimality of the initial allocation, and hence the claim follows.

Finally, let p be the number of nodes in V which are holding resource o_2, and let j be the number of nodes in E with at least one endpoint among these p nodes (or, equivalently, j is the number of edges adjacent to these p vertices in the graph $\mathcal{G}$). Then the overall cost of such allocation equals

$$\left(j \cdot \frac{1}{4} \cdot 1 + (m-j) \cdot \frac{1}{4} \cdot 2\right) + \left(p \cdot \frac{3}{5} \cdot 1 + (n-p) \cdot \frac{2}{5} \cdot 1\right) = \left(\frac{2n}{5} + \frac{m}{2}\right) + \left(\frac{p}{5} - \frac{j}{4}\right).$$

Therefore, to achieve the optimal allocation we need to minimize $\frac{p}{5} - \frac{j}{4}$ in the above expression, as all the other terms are constant. But this minimum is achieved when j takes its maximum value, i.e., $j = m$. This is because if $j < m$, let $e^* = (u, v) \in E$ be such that none of its endpoints is among the p vertices holding resource o_2. Then increasing p to $p+1$ by allowing u to cache resource o_2, the value of j will also increase to at least $j+1$ (since now at least e^* is also covered). As a result, the value of $\frac{p}{5} - \frac{j}{4}$ decreases to at most $\frac{p+1}{5} - \frac{j+1}{4} = \frac{p}{5} - \frac{j}{4} - \frac{1}{20}$, which shows that in order to achieve the minimum we must have $j = m$. This means that the set of p nodes holding resource o_2 must form a vertex cover for $\mathcal{G}$. As we are looking for the minimum value of $\frac{p}{5} - \frac{j}{4}$, thus p must be the cardinality of the minimum vertex cover. Therefore, an allocation profile on $\mathcal{K}$ is optimal if and only if it assigns resource o_2 to those nodes in V forming a minimum vertex cover on $\mathcal{G}$ and resource o_1 to the remaining nodes. This completes the proof. ∎

4. An approximation algorithm for the optimal resource allocation. As we saw in Theorem 3.3, it seems computationally hard to approximate the optimal resource allocation better than a certain constant factor, even in the special case of unit cache size. Therefore, an important question concerns how closely one can approximate the optimal allocation (2). In this section, we show that if the access costs satisfy the metric property, one can quickly obtain a 3-approximation of the optimal allocation with identical request rates. Here we assume that the sum of the cache sizes is larger than the total number of resources, i.e., $\sum_i u_i > k$. Otherwise, by convention, the objective cost of any allocation is defined to be very large.[3]

THEOREM 4.1. *There is a deterministic* 3*-approximation algorithm for the optimal resource allocation* (2) *with metric access costs and different cache sizes which runs in at most* $O(kn^2)$.

Proof. Consider a greedy algorithm in which at each time $t = 1, \ldots, n$, we select arbitrarily an unselected node and iteratively fill its cache by fetching into it a resource which has the farthest distance to previously assigned resources. More precisely, let us assume that nodes $1, 2, \ldots, t-1$ are all the nodes that are processed so far. At time t,

[3]In case that $\sum_i u_i = k$, the proposed greedy algorithm achieves the optimal cost by assigning each resource to exactly one cache location.

we select a new node (say node t) whose cache content is initially empty, i.e., $P_t = \emptyset$. We then iteratively fill P_t by fetching into it the new resource $\arg\max_{o \in O} \min\{c_{it} : o \in P_i,\, i \in [t]\}$, where ties are broken arbitrarily. Here, if a resource o has not yet been assigned anywhere (i.e., $o \notin P_i\ \forall i \in [t]$), we define $\min\{c_{it} : o \in P_i, i \in [t]\} = \infty$. Moreover, the distance between two resources in the same cache is defined to be zero. We show that the resource allocation profile $P = (P_1, \ldots, P_n)$ obtained at the end of this process when all the nodes are processed exactly once is a 3-approximation of the optimal allocation with different cache sizes.

Consider an arbitrary node (which without loss of generality we assume to be node 1), and relabel all the other nodes based on their access costs from node 1 by $2, 3, \ldots, n$, i.e., $0 = c_{11} \le c_{12} \le c_{13} \le \cdots \le c_{1n}$. In this sequence, let $j^* \ge 2$ be a node of smallest index such that $(P_1, \ldots, P_{j^*})$ contains a replicated resource for the first time (note that such a node exists as $\sum_i u_i > k$), i.e.,

$$|\cup_{i=1}^{j^*-1} P_i| = \sum_{i=1}^{j^*-1} |P_i|, \qquad |\cup_{i=1}^{j^*} P_i| < \sum_{i=1}^{j} |P_i|.$$

Let $j' \in \{1, 2, \ldots, j^* - 1\}$ be another node whose cache contains at least one same resource as node j^* (i.e., $P_{j^*} \cap P_{j'} \neq \emptyset$), and define t_{j^*} and $t_{j'}$ be the time instances that j^* and j' are selected to be processed by the algorithm, respectively. Finally, let us denote the set of all the nodes whose access cost from node 1 is at most $3c_{1j^*}$ by $S := \{i \in [n] : c_{1i} \le 3c_{1j^*}\}$.

We claim that in the final allocation profile P produced by the algorithm each resource is assigned to at least one node in S, i.e., $|\cup_{i \in S} P_i| = k$. Otherwise, let us assume that there exists at least one resource o which is missing among the nodes in S at this final profile P. Consider the time $\hat{t} = \max\{t_{j^*}, t_{j'}\}$, where $\hat{j} \in \{j^*, j'\}$ is selected by the algorithm. Note that at this time, each resource is assigned to at least one node; otherwise, there is no reason why the algorithm should assign a repeated resource to node $\hat{j}$. Since resource o is missing among the nodes in S at the final allocation P, clearly it is also missing at time $\hat{t}$ among the nodes in S. Thus, if we let j'' be the closest node to node $\hat{j}$ at time $\hat{t}$ which has resource o in its cache, then $c_{1j''} > 3c_{1j^*}$. Moreover, we have $c_{\hat{j}j''} \le c_{j^*j'}$. Otherwise, if $c_{\hat{j}j''} > c_{j^*j'}$, then the algorithm would not have chosen to place at $\hat{j}$ the same resource, as it did at the other node (either j^* or j'), since we have shown that there is at least one resource, o, which is farther away. Given that access costs are metric and $c_{1\hat{j}} \le \max\{c_{1j^*}, c_{1j'}\} = c_{1j^*}$, we can write

$$c_{1j''} \le c_{1\hat{j}} + c_{\hat{j}j''} \le c_{1j^*} + c_{j^*j'} \le c_{1j^*} + c_{1j^*} + c_{1j'} \le 3c_{1j^*}.$$

This contradicts $c_{1j''} > 3c_{1j^*}$, and the claim follows.

Finally, let us denote an optimal resource allocation profile by P^o. Then the cost of node 1 in P^o is at least

$$C_1(P^o) \ge \sum_{i=1}^{j^*-1} |P_i| c_{1i} + \left(k - \sum_{i=1}^{j^*-1} |P_i|\right) c_{1j^*}, \tag{8}$$

where $|P_i| = u_i$ denotes the cache size of node i. This is because at best, node 1 can get access to $\sum_{i=1}^{j^*-1} |P_i|$ of the resources through nodes $1, 2, \ldots, j^* - 1$, with an overall access cost $\sum_{i=1}^{j^*-1} |P_i| c_{1i}$, and access the remaining $k - \sum_{i=1}^{j^*-1} |P_i|$ resources

at a minimum cost of c_{1j^*}. On the other hand, since in the allocation P all of the resources which are assigned to nodes $1, \ldots, j^*-1$ are distinct while all of the resources appear at least once in the collective caches of nodes in S (which are of distance at most $3c_{1j^*}$ from node 1), the cost of node 1 is at most

$$C_1(P) \le \sum_{i=1}^{j^*-1} |P_i| c_{1i} + \left(k - \sum_{i=1}^{j^*-1} |P_i|\right) 3c_{1j^*}. \tag{9}$$

Therefore, from (8) and (9) we have $C_1(P) \le 3C_1(P^o)$. Since node 1 was chosen arbitrarily, by summing this relation over all the nodes we obtain $C(P) \le 3C(P^o)$.

Finally, we note that the running time of the algorithm is at most $O(kn^2)$. This is because at the time of processing node $t+1$, for every resource o we need to find the minimum distance between node $t+1$ and all the preprocessed nodes $[t]$ that have resource o in their cache. This requires finding the minimum of at most t numbers $\{c_{1,t+1}, \ldots, c_{t,t+1}\}$ and can be done in $O(t)$. Thus, after at most $O(kt)$ computation we can fill the cache of node $t+1$ with the $|P_{t+1}|$ resources of highest distance. Therefore, the running time of the algorithm is at most $O(k\sum_{t=1}^{n} t) = O(kn^2)$. ∎

Remark 1. We note that the result of Theorem 4.1 still holds under a more general online setting in which the nodes arrive over time and reveal their access costs and cache sizes. This is because the greedy algorithm described above does not require any specific order on how to process the nodes: it only needs the information of the previously processed nodes. Therefore, the allocation profile obtained by following the greedy algorithm is 3-competitive with respect to the best offline allocation when all the access costs and cache sizes are known a priori.

Next, in the following we show that the above approximation factor is nearly tight, meaning that the greedy algorithm cannot achieve a performance guarantee better than 2 even over unweighted networks with *graphical distance* access costs. Given arbitrary positive integer $k \ge 2$, let us consider a network of $n := k(k-1)$ nodes as shown in Figure 3. Here we denote the resources by $\{o_1, o_1, \ldots, o_k\}$. The bottom part of the network is composed of a clique of $k-1$ nodes, and the top part forms an independent set of $(k-1)^2$ nodes, all of which are connected to the bottom part. The top figure constitutes an output of the greedy algorithm where the algorithm iteratively selects one node from each resource cluster until all the nodes are selected exactly once. Here we assume that ties are broken accordingly so that the number of resources of each type at the end of the algorithm is the same and equals $k-1$ (as shown in the top figure). Therefore, the total cost of such an allocation, which is achieved by the greedy algorithm, is equal to

$$C(\text{greedy}) = (k-1)[(k-1)(1+(k-2)\times 2)] + (k-1)^2 = 2(k-1)^3,$$

where the first term is the total cost of all the nodes that belong to resource classes $o_2, \ldots, o_k$, and the second term is the overall cost of all the nodes that belong to the resource class o_1 (black nodes). On the other hand, it can be seen easily that the bottom figure illustrates an optimal allocation where the cost of each node equals $k-1$ with an overall optimal cost of $C(\text{OPT}) = k(k-1)^2$. Thus, we have $C(\text{greedy}) = (2-\frac{2}{k})C(\text{OPT})$. This shows that for large k the greedy algorithm can perform nearly twice worse than the optimal algorithm.

5. Resource allocation game. In this section, we consider a selfish version of the optimal resource allocation problem formulated as a noncooperative game between

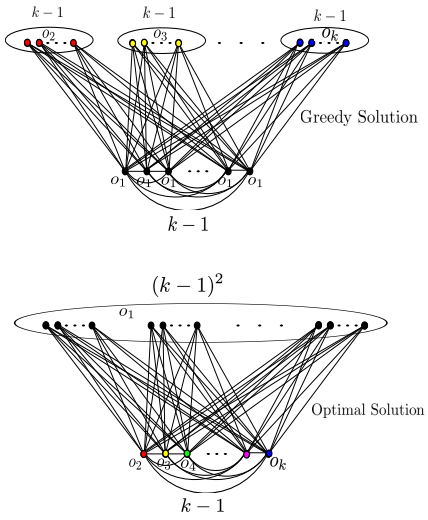

FIG. 3. *Illustration of the resource allocation obtained by the greedy algorithm (top figure) and an optimal allocation (bottom figure).*

nodes (players). More formally, we consider a noncooperative game with n players (one player per node). Each player has access to the set of all of the resources but can only select one of the resources to be in its cache (see Remark 2 below for arbitrary cache sizes). In other words, an action for player $i \in [n]$ is to choose only one resource $P_i \in [k]$. Finally, given an allocation (action) profile P, the cost of player i is naturally given as before by (1):

$$C_i(P) := \sum_{\ell \in O} c_{i,i(\ell)},$$

which is the minimum cost player i incurs to access all the resources. Here $i(\ell)$ refers to the closest node to player i holding resource ℓ in its cache given the allocation profile P. One of the immediate questions with regard to this game is whether it admits a pure-strategy Nash equilibrium (henceforth referred to as NE), and if yes, how can we obtain one such equilibrium points efficiently?

Remark 2. All the results of this section can be extended to the games of different cache size in which different players may have different cache sizes. This can be done by replacing a player of cache size u_i with u_i collocated players of unit cache size. Now finding a NE for this unit cache size game, one can obtain a NE for the original instance by putting together cache contents of all the u_i copies of player i.

Before delving into the analysis of the above homogeneous resource allocation game, we note that if we instead consider a heterogeneous game where players have different request rates for different resources, then the heterogeneous game does not necessarily admit a NE even for $k = 3$ resources and metric access costs (see Example 5.1). In fact, this is one of the main reasons that in the first place we restricted our attention to the homogeneous setting.

Example 5.1. Let us consider a heterogeneous game with $n = 6$ players and three resources $O = \{o_1, o_2, o_3\}$. We assume that each player has a unit cache size, and we define the metric access costs and the resource request rates to be as those given in

Figure 4. For instance, the request rate for resource o_3 to player 1 is $w_1(o_3) = \frac{25}{17}$, and the access cost between player 1 and player 3 is $c_{13} = \frac{2+\epsilon}{3}$. Here $\epsilon > 0$ can be chosen to be a small positive number (e.g., $\epsilon = 0.005$). We show that the heterogeneous game in which the cost of player i equals $C_i(P) = w_i(o_1)c_{i,i(o_1)} + w_i(o_2)c_{i,i(o_2)} + w_i(o_3)c_{i,i(o_3)}$ does not admit a NE. To see this, we first note that in any NE players 4, 5, and 6 must cache resources o_3, o_1, and o_2, respectively. This is because the request rates for other resources to these players are 0, and hence they have no incentive to cache any other resources. Moreover, player 1 is only interested in resources $\{o_2, o_3\}$, player 2 is only interest in resources $\{o_1, o_3\}$, and player 3 is only interest in resources $\{o_2, o_3\}$. These together constitute eight possible allocation of resources (P_1, P_2, P_3) to the players $\{1, 2, 3\}$, which are listed in the following left column. However, for each of these eight possibilities, one of the players in $\{1, 2, 3\}$ can deviate and strictly reduce its cost by caching a different resource. All the cost-savings due to such deviations are listed in the following right column. They are strictly positive. For instance, the allocation profile $(P_1, P_2, P_3) = (o_2, o_1, o_3)$ does not constitute a NE because if player 2 changes its resource from o_1 to o_2, it can strictly reduce its cost by $w_2(o_3)c_{23} - w_2(o_1)c_{25} = 2 \cdot \frac{5}{9} - 1 \cdot 1 = \frac{1}{9} > 0$. Similarly, as shown in the following table none of the other allocations can constitute a NE, which shows that this heterogeneous game does not admit a NE.

$$\begin{array}{c} \\ o_1 \\ o_2 \\ o_3 \end{array} \begin{array}{c} \begin{array}{cccccc} w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ \frac{25}{17} & 2 & \frac{4}{5} & 1 & 0 & 0 \end{bmatrix} \end{array}$$

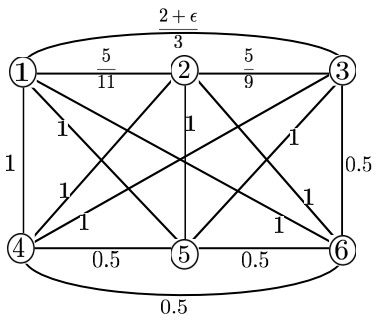

FIG. 4. *An example for a resource allocation game with heterogeneous request rates which does not admit a pure NE. The game has six players, and there are $O = \{o_1, o_2, o_3\}$ resources. The access costs are illustrated on the right graph and satisfy the metric property. Moreover, the request rates are given in the left table. For instance the request rate for resource o_3 to player 1 is $w_1(o_3) = \frac{25}{17}$.*

(P_1, P_2, P_3)		Cost-saving for the deviating player
(o_2, o_1, o_3)	player 2 deviates from o_1 to o_3 $\longrightarrow$	$w_2(o_3)c_{23} - w_2(o_1)c_{25} = 2 \cdot \frac{5}{9} - 1 \cdot 1 = \frac{1}{9}$
(o_2, o_1, o_2)	player 1 deviates from o_2 to o_3 $\longrightarrow$	$w_1(o_3)c_{14} - w_1(o_2)c_{13} = \frac{25}{17} \cdot 1 - \frac{2+\epsilon}{3} > 0$
(o_2, o_3, o_2)	player 1 deviates from o_2 to o_3 $\longrightarrow$	$w_1(o_3)c_{12} - w_1(o_2)c_{13} = \frac{25}{17} \cdot \frac{5}{11} - \frac{2+\epsilon}{3} > 0$
(o_2, o_3, o_3)	player 3 deviates from o_3 to o_2 $\longrightarrow$	$w_3(o_2)c_{36} - w_3(o_3)c_{32} = 1 \cdot \frac{1}{2} - \frac{4}{5} \cdot \frac{5}{9} > 0$
(o_3, o_1, o_2)	player 3 deviates from o_2 to o_3 $\longrightarrow$	$w_3(o_3)c_{31} - w_3(o_2)c_{36} = \frac{4}{5} \cdot \frac{2+\epsilon}{3} - \frac{1}{2} > 0$
(o_3, o_1, o_3)	player 1 deviates from o_3 to o_2 $\longrightarrow$	$w_1(o_2)c_{16} - w_1(o_3)c_{13} = 1 - \frac{25}{17} \cdot \frac{2+\epsilon}{3} > 0$
(o_3, o_3, o_2)	player 2 deviates from o_3 to o_1 $\longrightarrow$	$w_2(o_1)c_{25} - w_2(o_3)c_{21} = 1 \cdot 1 - 2 \cdot \frac{5}{11} > 0$
(o_3, o_3, o_3)	player 2 deviates from o_3 to o_1 $\longrightarrow$	$w_2(o_1)c_{25} - w_2(o_3)c_{21} = 1 \cdot 1 - 2 \cdot \frac{5}{11} > 0.$

The resource allocation game with heterogeneous request rates was first introduced in [20], where it was shown that such a game for *ultrametric* access costs always admits a NE.[4] In particular, a polynomial algorithm to compute a NE under ultrametric access costs was given in [20, Theorem 1]. Ultrametric costs which are a special case of metric costs induce a simple hierarchical or tree-like structure on the nodes' access costs which makes the analysis much simpler. The motivation for the use of ultrametric access costs is that in many recent works, hierarchical networks have been extensively used to model communication content delivery costs or cooperative caching in P2P networks [28]. However, the complexity of finding a NE for *general* access costs and under best-response dynamics, where at each iteration only one player updates its action by caching its most favorite resource, was left open. In particular, it was conjectured in [20] that for general access costs finding a NE is PLS-hard. Here we take the first affirmative step toward settling this conjecture by showing that finding a NE with $k \geq 3$ resources is equivalent to finding a flip-optimal solution of Max-k-Cut over a *weighted* complete graph of n nodes and exactly one edge of weight 2^i, $i = 0, \ldots, \binom{n}{2} - 1$. As solving Max-$k$-Cut using flip local moves is known to be PLS-hard [41], this strongly suggests that finding a NE using best response updates is also PLS-hard. But before we study the complexity of finding a NE, we first provide a positive result about approximability of the NE points under *metric* access costs.

DEFINITION 5.2. *Given $\beta > 1$, an allocation profile P is called a β-NE if no player can unilaterally deviate and reduce its cost by a factor of more than β, i.e., $C_i(P) \leq \beta C_i(P'_i, P_{-i})\ \forall P'_i \in [k], \forall i \in [n]$. Note that for $\beta = 1$ this definition reduces to the exact definition of a NE.*

PROPOSITION 5.3. *For the metric access costs, the allocation profile obtained at the end of the greedy algorithm is a 3-NE.*

Proof. As we saw earlier in the proof of Theorem 4.1, the allocation profile P obtained at the end of the greedy algorithm has the property that for every arbitrary node (say node 1), $C_1(P) \leq \sum_{\ell=1}^{j-1} c_{1\ell} + (k-j+1)3c_{1j}$, where here the definition of cost is written for the case of unit cache size game. Since the cost of node 1 in *any* allocation profile, and in particular for any deviation, is at least $\sum_{\ell=1}^{j-1} c_{1\ell} + (k-j+1)c_{1j}$, we immediately conclude that P is a 3-NE. ∎

While Proposition 5.3 shows that it is possible to compute an approximate NE under metric access costs, it does not help in finding an exact NE. Therefore, in the remainder of this section our goal is to develop a methodology for studying the complexity of finding an exact NE under *general* (not necessarily metric) access costs.

5.1. Complexity of NE for general access costs. In the remainder of this section, we consider the homogeneous resource allocation game and provide several results on the complexity of finding a NE under general access costs. For this purpose, we first consider the following definition, which allows us to provide a simple characterization for a NE.

DEFINITION 5.4. *Given a feasible allocation profile P, let us denote the set of players holding resource ℓ in their cache by V^ℓ. For an arbitrary vertex $i \in V$, we define $d(i, V^\ell) := \min\{c_{ij} : j \in V^\ell, j \neq i\}$. If $V^\ell = \emptyset$, we define $d(i, V^\ell) := \infty$.*

Geometrically, $d(i, V^\ell)$ can be thought of as the distance between vertex i and its

[4] A set of symmetric access costs $c_{ij} = c_{ji} \geq 0$ form an ultrametric if $c_{ik} \leq \max\{c_{ij}, c_{jk}\}\ \forall i, j, k$.

projection on the set $V^\ell \setminus \{i\}$.[5] Now let us consider an arbitrary allocation profile P and assume that player i deviates by changing its resource from P_i to P_i'. This means that player i leaves its resource class from V^{P_i} to $V^{P_i'}$. Such a deviation will change the cost of player i by

$$C_i(P_i', P_{-i}) - C_i(P) = d(i, V^{P_i}) - d(i, V^{P_i'}).$$

As a result, an allocation profile P is a NE if and only if $d(i, V^{P_i}) - d(i, V^{P_i'}) \leq 0 \; \forall i \in [n], \forall P_i' \in [k]$, or equivalently,

$$d(i, V^{P_i}) = \max_{\ell \in [k]} d(i, V^\ell) \quad \forall i \in [n]. \tag{10}$$

In other words, in a NE the distance of each node to its own resource class must be the largest among its distances to other resource classes. While (10) suggests that a resource allocation profile is a NE (or close to a NE) if the nodes within each class V^ℓ have large pairwise distances, a dual approach is to find a resource partition with small pairwise distances across different resource classes. Leveraging this idea and for the sake of completeness, in the following we use an argument similar to that in [20] to show the existence of a NE for the homogeneous game with general access costs.

PROPOSITION 5.5. *The homogeneous resource allocation game with general access costs always admits a NE.*

Proof. Given an allocation profile P, let us write $i \nsim_P j$ if nodes i and j do not belong to the same resource class induced by P. Consider the multiset $\Psi(P) := \{c_{ij} : i \nsim_P j\}$, whose elements are sorted in an increasing order (e.g., $\{1, 1, 2, 2, 2, 3\}$). Now if player i changes its resource from P_i to P_i', the resulting new multiset associated with $P' = (P_i', P_{-i})$ can be obtained from $\Psi(P)$ by removing the elements $N_i' := \{c_{ij}, j \in V^{P_i'}\}$ and adding new elements $N_i := \{c_{ij}, j \in V^{P_i} \setminus \{i\}\}$, i.e.,

$$\Psi(P') = (\Psi(P) \setminus N_i') \cup N_i, \tag{11}$$

where here the union and subtraction are with respect to multisets (i.e., repetition of elements is allowed). Now if player i's deviation is a strictly better move so that $C_i(P_i', P_{-i}) < C_i(P)$, this means that $d(i, V^{P_i}) < d(i, V^{P_i'})$. As $d(i, V^{P_i'}) = \min N_i'$, and $d(i, V^{P_i}) = \min N_i$, this implies that $\Psi(P')$ is lexicographically smaller than $\Psi(P)$. Thus, after every strictly better move by a player i, the sorted multiset $\Psi(P)$ strictly decreases lexicographically. As a result, after a finitely many number of moves we reach an allocation profile P^* with lexicographically smallest multiset $\Psi(P^*)$, which must be a NE (as no player can perform a strictly better move). ∎

Although Proposition 5.5 provides an inefficient way of proving the existence of a NE, a major question here is how many strictly better moves by players can happen in the worst case. In particular, how can we leverage this multiset *potential function* to find a NE more efficiently? To address these questions, we next consider the following definition, which allows us to make a connection between Max-k-Cut and the problem of finding a NE.

DEFINITION 5.6. *An instance of Max-k-Cut is called flip-optimal if the cut value cannot be increased by moving a single vertex from one partition set to another one.*

[5] One can view the greedy algorithm as a successive projection algorithm that sequentially takes a new node i and includes it into the set $\arg\max_{V^\ell} d(i, V^\ell)$ with farthest projected distance from i.

THEOREM 5.7. *Consider an arbitrary instance of the resource allocation game with n players and access costs $c_{e_0} > c_{e_1} > \cdots > c_{e_m}$, $m = \binom{n}{2} - 1$, where each e_i represents a distinct pair of players. Let K be a weighted complete graph of n nodes with edge e_i having weight 2^i. Then an allocation profile P is a NE if and only if the resource classes induced by P constitute a flip-optimal solution to Max-k-Cut on K.*

Proof. Given sorted access costs $c_{e_0} > c_{e_1} > \cdots > c_{e_m}$ in the resource allocation instance, let us consider the weighted complete graph K in which the edge corresponding to e_i has weight 2^i. Now let us consider an arbitrary NE profile P with corresponding resource classes V^ℓ, $\ell \in [k]$. If this partition is not flip-optimal for Max-k-Cut on K, this means that by moving a vertex v from some V^1 to V^2 we can strictly improve the value of the cut. Let $e_{i_1}, \ldots, e_{i_r}$, where $i_1 < \cdots < i_r$, denote all the edges between v and the vertices in V^2. Similarly, let $e_{j_1}, \ldots, e_{j_s}$, where $j_1 < \cdots < j_s$, denote all the edges between v and the vertices in $V^1 \setminus \{v\}$. Since moving v from V^1 to V^2 strictly improves the value of the cut, we must have

$$2^{i_1} + \cdots + 2^{i_r} < 2^{j_1} + \cdots + 2^{j_s}.$$

This implies that $j_s > i_r$, and hence $c_{e_{j_s}} < c_{e_{i_r}}$. As $c_{e_{j_s}} = \min\{c_{e_{j_1}}, \ldots, c_{e_{j_s}}\}$ and $c_{e_{i_r}} = \min\{c_{e_{i_1}}, \ldots, c_{e_{i_r}}\}$, we conclude that such a move from V^1 to V^2 is also a strictly better move for player v. This is because before such a deviation, the cost of player v for accessing resources $\ell = 1$ and $\ell = 2$ was 0 and $c_{e_{i_r}}$, respectively. However, after that deviation the cost of player v for accessing resources $\ell = 1$ and $\ell = 2$ becomes $c_{e_{j_s}}$ and 0, respectively. As the cost of player v for accessing all the other resources $\ell \notin \{1, 2\}$ remains unchanged, this shows that the cost of player v due to its deviation strictly reduces by exactly $c_{e_{i_r}} - c_{e_{j_s}} > 0$. This contradicts the fact that P was a NE.

Conversely, assume that V^ℓ, $\ell \in [k]$, form a flip-optimal solution to Max-k-Cut on K. By contradiction, assume that $\{V^\ell, \ell \in [k]\}$, viewed as resource classes, do not constitute a NE. By the NE characterization given in (10), this means that there exists a player v, and two resource classes V^1 and V^2, such that $v \in V^1$ and $d(v, V^1) < d(v, V^2)$. Since $d(v, V^1) = c_{e_{j_s}}$ and $d(v, V^2) = c_{e_{i_r}}$, we must have $j_s > i_r$. Hence by moving vertex v from V_1 to V_2, the value of the cut strictly increases by exactly

$$\begin{aligned} 2^{j_1} + \cdots + 2^{j_s} - (2^{i_1} + \cdots + 2^{i_r}) &\geq 2^{j_s} - (1 + \cdots + 2^{i_r}) \\ &= 2^{j_s} - 2^{i_r+1} + 1 \geq 1, \end{aligned}$$

where the last inequality holds since $j_s > i_r$. However, this contradicts the flip-optimality of V^ℓ, $\ell \in [k]$, which completes the proof. ∎

Remark 3. One can easily extend the statement of Theorem 5.7 to the case where the access costs in the original instance are *not* distinct. This can be simply done by encoding the original access costs into K using weights $\{1, n, \ldots, n^m\}$ (rather than powers of two). A straightforward calculation shows that all the above results can be carried over to this generalized setting where now there could be multiple edges of the same weight n^i. We note that even in this generalized case, the length of the input size is still polynomial in terms of the problem description, as $\log n^m = O(n^2 \log n)$.

As an immediate corollary of Theorem 5.7, for finding a NE the actual values of the access costs c_{ij} are not important: what really matters is the relative size of these access costs. In other words, every two collections of access costs $\{c_{ij}\}$ and $\{\hat{c}_{ij}\}$ which have the same relative order (i.e., $c_{ij} > c_{pq}$ if and only if $\hat{c}_{ij} > \hat{c}_{pq}$) will have

identical sets of NE points. Next, we show in the following theorem that finding a NE with the lexicographically smallest multiset (potential) is an NP-hard problem.

DEFINITION 5.8. *A graph is called k-colorable if there exists an assignment of at most k colors to its vertices with no adjacent monochromatic vertices.*

THEOREM 5.9. *It is NP-hard to find the lexicographically smallest NE of the resource allocation game for $k \geq 3$ resources. Moreover, there is a deterministic algorithm to find such a NE after at most $O(n^3 2^n)$ steps.*

Proof. By Theorem 5.7, it is enough to show that solving Max-k-Cut ($k \geq 3$) on a complete graph K with edge weights $1, 2, \ldots, 2^m$ is NP-hard. This implies that finding the lexicographically smallest NE which is associated with the optimal Max-k-Cut on K is also NP-hard. Let us consider the edges of K in a decreasing order of weights by $\hat{e}_m, \ldots, \hat{e}_0$ where edge $\hat{e}_i$ has weight 2^i. Initially, set $E_m = \{\hat{e}_m\}$. For each $i = m-1, \ldots, 0$, let $E_i = E_{i+1} \cup \{\hat{e}_i\}$ if the induced graph (V, E_i) is k-colorable, and $E_i = E_{i+1}$ otherwise. It is easy to see that at the end of this process E_0 spans all of the vertices of V. Let

$$V^\ell = \{v \in V : v \text{ has color } \ell\},\ \ell \in [k],$$

be the color classes of the k-proper coloring on (V, E_0). We claim that $\{V^\ell\}_{\ell \in [k]}$ constitutes an optimal solution to the Max-k-Cut problem on K with cut edges E_0. This is the case because in any optimal Max-k-Cut solution no edge of weight 2^i can be discarded from being in the cut for the sake of selecting edges of lower weights $2^{i-1}, \ldots, 1$ (note that all of these lower weight edges have a total weight of at most $2^i - 1$). More precisely, let $\hat{e}_j$ be the edge of maximum weight which has appeared in the optimal cut but not in E_0. Since E_{j+1} cannot contain an edge outside of the optimal cut edges (otherwise, by the above property of powers of 2 the value of the cut edges in E_{j+1} will be more than that of the optimal cut), we conclude that all of the edges in E_{j+1} must be present in the optimal cut. As at iteration j the edge $\hat{e}_j$ is a feasible choice of highest weight for the algorithm, we must have $E_j = E_{j+1} \cup \{e_j\} \subseteq E_0$. This contradiction shows that every edge which is present in the optimal cut must also be in E_0. Therefore, by running the k-coloring algorithm at most $m = O(n^2)$ times, we can find an optimal solution to the Max-k-Cut problem on K. It is known that the k-coloring problem for a graph of n nodes and any k can be solved exactly in $O(n2^n)$ [6], so the above process delivers a NE in at most $O(n^3 2^n)$ steps.

Finally, given an arbitrary graph G with n nodes and r edges, we can encode G into K by assigning weights $2^m, \ldots, 2^{m-r+1}$ to its r edges and assigning weights $2^{m-r}, \ldots, 1$ to the edges of the G-complement (in any arbitrary order). Now it is easy to see that the graph G is k-colorable if and only if the optimal solution to Max-k-Cut on K returns a proper k-coloring for G. Therefore, if we can find the lexicographically smallest NE of any instance of the resource allocation game efficiently, we can also find a proper k-coloring for any graph G efficiently. This completes the hardness proof. ∎

COROLLARY 5.10. *Checking the 2-colorability of a graph can be done in polynomial time; thus for $k = 2$ resources a NE can be found in polynomial time.*

6. NE computation. In this section, we take one step further toward computing the NE points. As seen earlier, it is computationally hard to find the lexicographically smallest NE in any instance of the resource allocation game. But what if we only want to find one of such equilibrium points? In that case, we do not necessarily need

to solve the *global* optimum of the Max-k-Cut problem on K. Thanks to Theorem 5.7, on the equivalence between the set of NE points and flip-optimal solutions of the weighted Max-k-Cut problem, below we provide a different characterization of the NE points in terms of *integral local minima* of a quadratic function. Leveraging this characterization, we propose two new algorithms to find a pure NE more systematically for any general instance of the resource allocation game.

It is well known [44, 7] that the Max-k-Cut problem on a complete graph K with edge weights $\{a_{ij}\}$ can be formulated as the following quadratic integer program:

$$\begin{aligned} &\min \frac{1}{2}\sum_{\ell=1}^{k}\sum_{i,j=1}^{n} a_{ij}x_i^\ell x_j^\ell, \\ &\sum_{\ell=1}^{k} x_i^\ell = 1 \quad \forall i \in [n], \\ &x_i^\ell \in \{0,1\} \quad \forall i \in [n],\ \ell \in [k], \end{aligned} \tag{12}$$

where $x_i^\ell = 1$ if and only if node i belongs to the ℓth partition set. Note that the objective function in (12) computes the total sum of edge weights within partition sets $V^\ell = \{i : x_i^\ell = 1\}$, $\ell \in [k]$. However, as the total sum of the edge weights $\sum_{i,j} a_{ij}$ is constant, minimizing the total sum of edge weights within partition sets is equivalent to maximizing the total sum of edge weights across them. Therefore, a solution to (12) will be a solution to the Max-k-Cut problem on K.

On the other hand, it has been shown in [7, Theorem 1] that a natural relaxation of the quadratic program (12) which is obtained by replacing its integral constraints $x_i^\ell \in \{0,1\}$ by $x_i^\ell \geq 0$ always admits an integral solution. Thus, without any loss of generality, we can consider the *relaxed* quadratic program in (12) and derive the Karush–Kuhn–Tucker (KKT) optimality conditions for its optimal solutions. Doing that, we obtain the following set of KKT conditions [7, Theorem 2]:

$$\begin{aligned} &r_i^\ell - \lambda_i \geq 0 \quad \forall i, \ell, \\ &r_i^\ell = \sum_j a_{ij}x_j^\ell \quad \forall i, \ell, \\ &(r_i^\ell - \lambda_i)x_i^\ell = 0 \quad \forall i, \ell, \\ &\sum_\ell x_i^\ell = 1 \quad \forall i, \\ &x_i^\ell \geq 0 \quad \forall i, \ell, \end{aligned} \tag{13}$$

where λ_i denotes the Lagrange multiplier associated with the constraint $\sum_{\ell=1}^{k} x_i^\ell = 1$.

LEMMA 6.1 (see [44, Theorem 1]). *A feasible solution (x_i^ℓ) to Max-k-Cut on K is flip-optimal if and only if it is an integral feasible solution to the KKT conditions* (13).

Now we state one of the main results of this section, which allows viewing NE points of the resource allocation game as integral local minima of a quadratic function over the polytope of stochastic matrices. We then leverage this characterization to devise more efficient algorithms for obtaining NE points.

THEOREM 6.2. *Given an arbitrary instance of the resource allocation game, let $A = (a_{ij})$ be the edge-weight matrix of its associated weighted complete graph K.*

Moreover, let $X = (x_i^\ell) \in \{0,1\}^{n\times k}$ be a feasible resource allocation profile where $x_i^\ell = 1$ if and only if player i caches resource ℓ. Then X is a NE if and only if it is an integral local minimum of $g(X) = \frac{1}{2}tr(X^TAX)$ over $(\mathcal{S}, \|\cdot\|)$, where $\mathcal{S}$ denotes the polytope of stochastic matrices of size $n \times k$, and $\|\cdot\|$ is the Frobenius norm.

Proof. Let us rewrite the KKT conditions (13) in a slightly different form: First, we replace the last two constraints in (13) by $X \in \mathcal{S}$. Moreover, the second constraint can be written as $R = AX$, where $R = (r_i^\ell) \in \mathbb{R}_+^{n\times k}$. Finally, the first and third constraints together imply $\lambda_i = \min_\ell r_i^\ell \ \forall i$ (i.e., λ_i is equal to the smallest entry in the ith row of R). Next, we write the objective function in (12) in a matrix form as

$$g(X) := \frac{1}{2}\sum_{\ell=1}^{k}\sum_{i,j=1}^{n} a_{ij}x_j^\ell x_i^\ell = \frac{1}{2}\mathrm{tr}(X^TAX).$$

According to the necessary conditions of optimality, if X is an integral local minimum of $g(\cdot)$ over $(\mathcal{S}, \|\cdot\|)$, then it must be an integral feasible solution of KKT conditions (13). Using Lemma 6.1, an integral solution to KKT conditions must be flip-optimal to Max-k-Cut on K. This in view of Theorem 5.7 shows that X must be a NE.

Conversely, if X is a NE for the resource allocation game, by Theorem 5.7 it must be a flip-optimal solution to Max-k-Cut on K. Therefore, by Lemma 6.1, X forms an *integral* solution to the KKT constraints (13). Moreover, every two nonzero columns of X must be distinct; otherwise, we would have at least two 1's in the same row of X, contradicting the stochasticity of X. Moreover, since every row of A contains different powers of 2 (recall that each entry a_{ij} of A is a distinct number from $\{1, 2, \ldots, 2^m\}$), all the elements in the ith row of $R = AX$ are distinct. This is simply because the binary representation of the elements in the ith row of R are given by the columns of X, which we just showed to be distinct. As a result, for each i, the minimum entry in the ith row of R is determined uniquely. Since X satisfies KKT conditions, $\lambda_i = \min_\ell r_i^\ell \ \forall i$. This shows that for each $i \in [n]$, there exists *exactly* one $\ell(i)$ such that $\lambda_i = r_i^{\ell(i)}$, and $r_i^\ell > \lambda_i \ \forall \ell \neq \ell(i)$. In other words, exactly n of the constraints $r_i^\ell - \lambda_i \geq 0$ in (13) are tight. Finally, it has been shown in [7, Theorem 3] that if X is a feasible solution to the KKT conditions for which at most n of the constraints $r_i^\ell - \lambda_i \geq 0$ are tight, then X must be a local minimum of $g(\cdot)$ over $(\mathcal{S}, \|\cdot\|)$. This completes the proof. ∎

As a result of Theorem 6.2, finding a NE is equivalent to finding an integral local minimum of $g(\cdot)$ with respect to the conventional Frobenius norm $(\mathcal{S}, \|\cdot\|)$. This lays down the main idea of the following two algorithms, which systematically aim to find integral local minima of $\min\{g(X) : X \in \mathcal{S}\}$. The first algorithm resembles a simplex-type method that explores the extreme points of $\mathcal{S}$ systematically until an integral local minimum is achieved. The second algorithm, however, allows cutting through the feasible region $\mathcal{S}$ and uses some ideas from a gradient descent method together with randomized rounding to find an integral local minimum.

6.1. A bilinear algorithm for finding a NE. Our first *Bilinear Algorithm* for finding a NE starts from an approximate NE generated by the greedy algorithm and proceeds in several finite length rounds. At the end of each round, the algorithm finds an *integral stationary* point X^* which is very likely to be a NE. Otherwise, if X^* is not a NE, an unsatisfied player i is identified and a strict better move is executed. The algorithm then continues to start a new round. A formal description of this algorithm is given in Algorithm 1.

Algorithm 1 (Bilinear Algorithm)

1: **Initialization:** Run the greedy algorithm to find an approximate NE, $X^{(1)}$.
2: **Bilinear Round:** Let $X^{(r)}$ be the allocation profile at the beginning of round r, and let $X^{(t)}$ be the allocation profile obtained at time $t \geq r$ during round r.
   (a) For fixed $X^{(t)}$, solve the linear program $\min\{\text{tr}((X^{(t)})^T AX) : X \in \mathcal{S}\}$, and let $Y^{(t)}$ be its optimal integral solution.
   (b) For fixed $R^{(t+1)} = AY^{(t)}$, solve the linear program $\min\{\text{tr}(X^T R^{(t+1)}) : X \in \mathcal{S}\}$, and let $X^{(t+1)}$ be its optimal integral solution. If $X^{(t+1)} \neq X^{(t)}$, set $t \leftarrow t+1$ and go to step 2(a). Otherwise, go to step 3.
3: Let $X^{(t+1)}$ be the stationary allocation obtained at the end of round r. Check whether $X^{(t+1)}$ satisfies the KKT conditions (13), which only requires checking $([AX^{(t+1)}]_{i\ell} - \min_\ell [AX^{(t+1)}]_{i\ell})[X^{(t+1)}]_{i\ell} = 0 \ \ \forall i, \ell$. If "Yes," output $X^{(t+1)}$ as a NE. Otherwise, let $X^* := \arg\min_{X \in \{X^{(r)}, X^{(t+1)}\}} \text{tr}(X^T AX)$.
4: Identify an unsatisfied player i in the allocation profile X^*, and let him play his best response. Set the allocation profile at the beginning of the next round $X^{(r+1)}$ to the one obtained from X^* after this deviation, and go to step 2.

PROPOSITION 6.3. *Algorithm* 1 *correctly returns a NE after finitely many steps.*

Proof. If the algorithm terminates, the output allocation would be a NE (as it is an integral allocation that satisfies the KKT conditions). Thus, we only need to argue that the algorithm indeed terminates after finitely many steps. To show this, we note that during each iteration of the bilinear round the value of the function $\text{tr}(X^T R)$ can only decrease. This is because

$$\text{tr}((X^{(t)})^T R^{(t)}) > \text{tr}((X^{(t)})^T R^{(t+1)}) > \text{tr}((X^{(t+1)})^T R^{(t+1)}),$$

where the first inequality is by step 2(a) and the second inequality is by step 2(b). Moreover, given that $X^{(t)}$ and $R^{(t)}$ change over finitely many extreme points of polytopes $\mathcal{S}$ and $\mathcal{R} := \{AX : X \in \mathcal{S}\}$, after finitely many iterations we must end up at a stationary point $X^{(t+1)} = X^{(t)}$. Thus, each bilinear round contains at most finitely many iterations and ends with a stationary allocation $X^{(t+1)}$. Moreover, in the case where $X^{(t+1)}$ is not a NE, since at the end of each round one player plays a strictly better response, the value of the objective function $g(X) = \text{tr}(X^T AX)$ at the beginning of the next round is less than that at the end of the previous round, i.e.,

$$\text{tr}((X^{(r+1)})^T AX^{(r+1)}) < \text{tr}((X^*)^T AX^*) \leq \text{tr}((X^{(r)})^T AX^{(r)}).$$

As this objective function only takes nonnegative integer values and its value decreases at each round, the algorithm terminates with a NE after finitely many steps. ∎

6.2. A projected gradient descent algorithm for finding a NE. As we saw earlier, one of the difficulties in finding a NE is the integral nature of pure equilibrium points. However, one way of dealing with this issue is to allow the algorithm to search over fractional feasible allocations $X \in \mathcal{S}$ and then round them to integral extreme points of $\mathcal{S}$ and check whether KKT conditions (13) are satisfied. In this regard, we propose the following projected gradient descent algorithm, which allows cutting over the interior points of the feasible region.

To describe this algorithm, it is more convenient to rewrite the relaxed optimization problem (12) in a slightly different form. Let us define $H := A \otimes I_{k \times k}$ to be the *Kronecker* product of A and the identity matrix $I_{k \times k}$. Moreover, instead of viewing

variables in a matrix form X, we view them as a column vector $\boldsymbol{x} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$, where $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^k)$. Then the relaxed optimization (12) can be written as

$$\min \left\{ g(\boldsymbol{x}) := \frac{1}{2} \boldsymbol{x}^T H \boldsymbol{x} : \quad F\boldsymbol{x} \geq b, \ \boldsymbol{x} \in \mathbb{R}^{n \times k} \right\}.$$

Here F is an $n(k+1) \times nk$ constraint matrix whose first n rows correspond to the constraints $\sum_{\ell=1}^{k} x_i^\ell = 1$, $i \in [x]$, and whose last nk rows form an identity matrix corresponding to the nonnegativity constraints $x_i^\ell \geq 0 \ \forall i, \ell$. Similarly, the right-hand side vector b has 1 in its first n entries and 0 for the remaining ones. Note that in this form the gradient of the objective function $g(\cdot)$ with respect to vector variable $\boldsymbol{x}$ is given by $H\boldsymbol{x}$. Now we are ready to describe our algorithm, which is mainly based on *Rosen's gradient projection* method [11] combined with a randomized rounding step in order to find an integral solution to the KKT conditions. This has been summarized in Algorithm 2.

Algorithm 2 Rounded Projected Gradient Descent

1: Run the greedy algorithm to obtain an initial approximate NE $\boldsymbol{x}^0$, and choose a constant $c > 0$.

2: Given a solution point $\boldsymbol{x}^t = (\boldsymbol{x}_1^t, \ldots, \boldsymbol{x}_n^t)^T$, randomly and independently round each $\boldsymbol{x}_i^t$, $i \in [n]$, to a basic unit vector using its induced probability distribution. Let $\hat{\boldsymbol{x}}^t$ be the resulting rounded solution. If $\hat{\boldsymbol{x}}^t$ satisfies KKT conditions (13), output $\hat{\boldsymbol{x}}^t$ as a NE. Otherwise, go to step 3.

3: Let $J^t = \{j : F_j \hat{\boldsymbol{x}}^t = b_j\}$ be the index set of active constraints at $\hat{\boldsymbol{x}}^t$, and let F_{J^t} be a submatrix of F consisting of rows F_j, $j \in J^t$. Define $P_{J^t} = I - F_{J^t}(F_{J^t}^T F_{J^t})^{-1} F_{J^t}^T$, and let $\boldsymbol{u}^t = -(F_{J^t}^T F_{J^t})^{-1} F_{J^t}^T H \hat{\boldsymbol{x}}^t$. Let $\boldsymbol{u}_h^t = \max\{\boldsymbol{u}_j^t : j \in J^t\}$, and choose a feasible descent direction:

$$d^t = \begin{cases} -P_{J^t} H \hat{\boldsymbol{x}}^t & \text{if } \|P_{J^t} H \hat{\boldsymbol{x}}^t\| > c\boldsymbol{u}_h^t, \\ -P_{J^t \setminus \{h\}} H \hat{\boldsymbol{x}}^t & \text{if } \|P_{J^t} H \hat{\boldsymbol{x}}^t\| \leq c\boldsymbol{u}_h^t. \end{cases}$$

Update $\boldsymbol{x}^{t+1} = \hat{\boldsymbol{x}}^t + \alpha d^t$, $t \leftarrow t+1$, and go to step 2. Here α is a step size where

$$\alpha = \begin{cases} 1 & \text{if } Fd^t \geq 0, \\ \min\{\frac{b_j - F_j \hat{\boldsymbol{x}}^t}{A_j d^t} : F_j d^t < 0, j \notin J^t\} & \text{else.} \end{cases}$$

Algorithm 2 starts from an approximate NE and iteratively improves the solution by minimizing the objective function $g(\cdot)$ using a projected gradient descent method. At each major iteration (step 3) and given a current integral feasible solution $\hat{\boldsymbol{x}}^t$, the algorithm finds a feasible descent direction d^t and a step size α such that $\boldsymbol{x}^{t+1} := \hat{\boldsymbol{x}}^t + \alpha d^t$ remains feasible while the objective value strictly decreases $g(\boldsymbol{x}^{t+1}) < g(\hat{\boldsymbol{x}}^t)$. The step size α is obtained by a line search so that moving along the feasible direction d^t gives the steepest descent. The feasible direction is obtained by projecting the gradient of the objective function $\nabla g(\hat{\boldsymbol{x}}^t) = H\hat{\boldsymbol{x}}^t$ to the set of active constraints via the projection matrix P_{J^t}. This ensures that moving along d^t does not violate any of the active constraints while it is a descent direction. Here $\boldsymbol{u}^t$ can be thought of as Lagrange multipliers associated with the active constraints where since $\nabla g(\hat{\boldsymbol{x}}^t) = P_{J^t} \nabla g(\hat{\boldsymbol{x}}^t) - F_{J^t} \boldsymbol{u}^t$, one can show that $\hat{\boldsymbol{x}}^t$ satisfies KKT conditions if and only if

$P_{J^t} H\hat{\boldsymbol{x}}^t = 0$ and $\boldsymbol{u}^t \leq 0$. Now if one of the Lagrange multipliers $\boldsymbol{u}_h^t$ is very large, it indicates that the corresponding constraint need not be in the set of active constraints. In this case, the corresponding constraint with index number h is evicted from the set of active constraints J^t and the new projection matrix $P_{J^t \setminus \{h\}}$ is recalculated to find a feasible descent direction. Following this algorithm, we have the following result.

PROPOSITION 6.4. *Algorithm* 2 *correctly returns a NE in finite expected time.*

Proof. In step 2 of the algorithm, we independently round $\boldsymbol{x}_i^t$, $i \in [n]$, to unit vectors; that is, for each $i \in [n]$ we independently set $\hat{\boldsymbol{x}}_i^t = e_\ell$ with probability $(\boldsymbol{x}^t)_i^\ell$. Since rounding is done independently for distinct $i \neq j$, by the linearity of expectation,

$$\mathbb{E}[g(\hat{\boldsymbol{x}}^t)|\boldsymbol{x}^t] = \frac{1}{2}\mathbb{E}\left[\sum_{i,j} a_{ij}\langle \hat{\boldsymbol{x}}_i^t, \hat{\boldsymbol{x}}_j^t\rangle | \boldsymbol{x}^t\right]$$
$$= \frac{1}{2}\sum_{i,j} a_{ij}\langle \mathbb{E}[\hat{\boldsymbol{x}}_i^t|\boldsymbol{x}^t], \mathbb{E}[\hat{\boldsymbol{x}}_j^t|\boldsymbol{x}^t]\rangle = \frac{1}{2}\sum_{i,j} a_{ij}\langle \boldsymbol{x}_i^t, \boldsymbol{x}_j^t\rangle = g(\boldsymbol{x}^t).$$

By taking one more expectation from the above expression with respect to $\boldsymbol{x}^t$, we have $\mathbb{E}[g(\hat{\boldsymbol{x}}^t)] = \mathbb{E}[\mathbb{E}[g(\hat{\boldsymbol{x}}^t)|\boldsymbol{x}^t]] = \mathbb{E}[g(\boldsymbol{x}^t)]$. As a result, the expected value of the objective function remains the same after rounding step 2. Thus, after each major iteration we obtain a new integral solution $\hat{\boldsymbol{x}}^{t+1}$ such that $\mathbb{E}[g(\hat{\boldsymbol{x}}^{t+1})] = \mathbb{E}[g(\boldsymbol{x}^{t+1})] < \mathbb{E}[g(\hat{\boldsymbol{x}}^t)]$, where the inequality follows from the fact that $g(\boldsymbol{x}^{t+1}) < g(\hat{\boldsymbol{x}}^t)$ with probability 1. This shows that in expectation no integral extreme point of the feasible region is visited twice, and hence the algorithm terminates in expected finite time. In particular, the convergent integral point must satisfy KKT conditions, as otherwise the algorithm can find a feasible descent direction in step 3 to strictly decrease the expected value of the objective function.[6] Thus, the algorithm terminates with a valid NE. ∎

Of course, we have no a priori bound on the expected running time of the above algorithm. However, in general, projected gradient descent algorithms with a proper choice of step size are known to converge relatively fast to stationary points of the objective function [5, 42]. In particular, projected gradient descent algorithms are typically faster than naive local search methods with slightly more computations per iteration [40]. Therefore, to evaluate the effectiveness of Algorithm 2, we use a numerical example to illustrate the outperformance of this algorithm compared to a naive enumeration search for finding a NE.

7. Numerical results. We consider a set of $n = 10$ players and generate 10 different access cost matrices with off-diagonal entries uniformly and independently sampled from the set $\{1, 2, \ldots, 2^{44}\}$. For each instance, we increase the number of resources from $k = 2$ to $k = 9$ and run Algorithms 1 and 2 until a NE is found. The results of these experiments are summarized in Tables 1 and 2, respectively, where in each of these tables a row corresponds to a random cost matrix instance $\mathcal{G}_i$, $i = 1, \ldots, 10$, while the columns correspond to different numbers of resources.

The results of applying Algorithm 1 on the randomly generated cost matrices are reported in Table 1. The entries of this table are in the form of pairs where the first coordinate denotes the total number of bilinear iterations (step 2) to find stationary points during the run of Algorithm 1, and the second coordinate shows the total number of single deviations by a player during the examination of stationary points

[6] In fact, it was shown in [11, Theorem 3.9] that only iterating step 3 of Algorithm 2 (excluding the rounding step) generates a sequence whose accumulation points satisfy the KKT conditions.

TABLE 1

Result of applying Algorithm 1 *for finding a NE in the resource allocation game for* 10 *players and different numbers of resources. The first* 10 *rows correspond to* 10 *different uniformly sampled access cost matrices. The last row represents the average iterations over all the instances.*

Case	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
$\mathcal{G}_1$	4, 2	6, 3	3, 1	10, 4	6, 2	2, 1	4, 2	2, 1
$\mathcal{G}_2$	10, 4	4, 1	11, 4	15, 5	7, 4	7, 4	3, 2	4, 2
$\mathcal{G}_3$	4, 2	16, 7	17, 6	8, 3	6, 3	6, 3	2, 1	2, 1
$\mathcal{G}_4$	5, 3	3, 2	4, 2	8, 4	6, 3	2, 1	6, 2	1, 1
$\mathcal{G}_5$	12, 6	15, 6	8, 4	3, 1	11, 4	16, 7	19, 6	14, 5
$\mathcal{G}_6$	3, 1	2, 1	7, 4	4, 2	18, 9	13, 6	4, 2	4, 2
$\mathcal{G}_7$	12, 6	21, 9	21, 6	29, 10	11, 5	7, 3	2, 1	2, 1
$\mathcal{G}_8$	14, 7	10, 4	19, 7	8, 4	6, 4	9, 4	4, 2	3, 1
$\mathcal{G}_9$	3, 2	4, 2	3, 2	4, 2	1, 1	4, 2	4, 2	2, 1
$\mathcal{G}_{10}$	2, 1	10, 4	18, 6	12, 6	19, 7	16, 7	21, 8	2, 1
BR	10.75	18.25	27.12	29.12	30.62	32.25	24.12	20.25

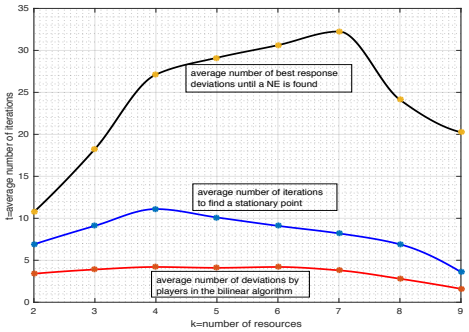

FIG. 5. *Average number of players' deviations in Algorithm* 1 *(bottom curve) versus average number of BR iterations (top curve).*

(step 4). For instance, the entry $(4, 2)$ corresponding to row $\mathcal{G}_6$ and column $k = 5$ shows that Algorithm 1 has found a NE after four bilinear steps (each involves solving a linear program) and a total of two deviations by unsatisfied players. Moreover, we use the best response (BR) dynamics as our comparison benchmark, where at each iteration an unsatisfied player is chosen at random and performs the BR update (i.e., caches a resource which reduces its cost the most). The last row of Table 1 denotes the average number of BR iterations over all the instances until a NE is found. Finally, in Figure 5 we have illustrated the average number of deviations in the bilinear algorithm (bottom curve), the average number of bilinear steps to find a stationary point (middle curve), and the average number of BR iterations (top curve). As can be seen, the bilinear algorithm substantially reduces the average number of players' deviations compared to the BR dynamics by solving a few intermediate linear programs.

Next, we evaluate the result of the projected gradient descent algorithm (Algorithm 2) over the set of random access costs $\mathcal{G}_i$, $i = 1, \ldots, 10$. The results are reported in Table 2. Again, each row of this table corresponds to a random cost matrix instance, while the columns correspond to the different numbers of resources. The entries of this table denote the number of gradient descent iterations during the

TABLE 2

Result of applying Algorithm 2 for finding a NE in the resource allocation game. The first 10 rows correspond to 10 different uniformly sampled access cost matrices. The last row represents the average number of iterations until a NE is found taken over all 10 instances.

Case	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
$\mathcal{G}_1$	6	8	12	17	15	11	7	10
$\mathcal{G}_2$	6	13	15	15	16	15	12	11
$\mathcal{G}_3$	5	12	10	21	13	9	11	10
$\mathcal{G}_4$	7	16	11	11	14	18	7	9
$\mathcal{G}_5$	8	12	8	23	14	19	12	9
$\mathcal{G}_6$	8	12	16	15	9	14	10	7
$\mathcal{G}_7$	6	18	15	21	14	13	9	13
$\mathcal{G}_8$	6	22	18	20	10	12	12	8
$\mathcal{G}_9$	7	18	25	10	14	10	8	7
$\mathcal{G}_{10}$	4	13	22	12	15	16	12	10
Ave	6.3	13.2	15.2	17.7	13.4	13.7	10	9.4

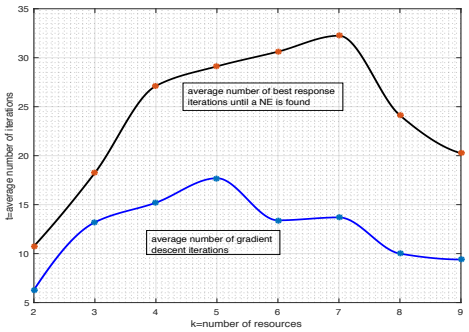

FIG. 6. *Average number of gradient descent iterations in Algorithm 2 (bottom curve) versus average number of BR iterations (top curve).*

executions of Algorithm 2. For instance, the entry 15 corresponding to row $\mathcal{G}_6$ and column $k=5$ shows that Algorithm 2 has found a NE after overall 15 rounding and gradient descent iterations. Finally, given a fixed number of resources k, the last row of Table 2 denotes the average number of gradient descent iterations over all the instances $\mathcal{G}_i$, $i=1,\ldots,10$. This row of the table has also been illustrated in Figure 6. As can be seen, the average number of iterations still outperforms the average number of iterations obtained by following the BR dynamics. Moreover, one can see from this figure that the worst-case running time happens for some midrange number of resources which in this experiment is for $k=5$ resources.

8. Conclusions. In this paper, we considered a resource allocation problem under two different settings. We studied this problem mainly from algorithmic and computational aspects. For the optimal allocation setting, we observed that this problem can be viewed as a generalization of the p-median problem with multiple types of facilities. We then provided an efficient algorithm to find the optimal allocation for $k=2$ resources. For $k\geq 3$, we showed that even approximating the optimal solution better than a constant factor is NP-hard and provided a simple 3-approximate algorithm

for the optimal allocation assuming metric access costs. For the game settings, we established an equivalence between NE points and flip-optimal solutions of the Max-k-Cut problem and provided several results on complexity and approximability of the equilibrium points. In particular, we provided two new algorithms for computing NE points using tools from quadratic programming.

8.1. Discussion and future directions. This work opens several important directions for future research. To name a few, one can consider the following:

(1) One can generalize our greedy algorithm to obtain improved approximation ratios for heterogeneous optimal allocation with resource placement costs. Unfortunately, developing a simple combinatorial algorithm for heterogeneous request rates does not seem so immediate. However, motivated from primal-dual interpretation/analysis of the greedy algorithms in the case of standard uncapacitated facility location problems [25], we believe that dual fitting analysis of our modified greedy algorithm can provide improved approximation ratios for the heterogeneous problem. To further motivate this idea, one can write a linear program and its dual for the heterogeneous problem as

$$
\begin{aligned}
z_{LP} = \min \sum_{i,j,\ell} w_j(\ell) c_{ij} x_{ij}^{\ell} + \sum_{i,\ell} f_i^{\ell} y_i^{\ell}, \qquad & z_D = \max \sum_{j,\ell} \beta_j^{\ell} - \sum_i \alpha_i, \\
x_{ij}^{\ell} \le y_i^{\ell} \quad \forall i,j,\ell, \qquad & \beta_j^{\ell} - u_{ij}^{\ell} \le w_j(\ell) c_{ij} \quad \forall i,j,\ell, \\
\sum_{i=1}^{n} x_{ij}^{\ell} \ge 1 \quad \forall j,\ell, \qquad & \sum_j u_{ij}^{\ell} - \alpha_i \le f_i^{\ell} \quad \forall i,\ell, \\
\sum_{\ell=1}^{k} y_i^{\ell} \le 1 \quad \forall i, \quad x_{ij}^{\ell}, y_i^{\ell} \ge 0, \qquad & u_{ij}^{\ell}, \beta_j^{\ell}, \alpha_i \ge 0.
\end{aligned}
$$

Now consider a greedy algorithm that at time $t \in [n]$ selects a node i_t and assigns to it a resource $\ell(i_t)$ based on some rule.[7] Let $X_t = \{i_1, \dots, i_t\}$ be the set of nodes that are processed up to time t, and define $X_t^{\ell} = \{i \in X_t : \ell(i) = \ell\}$, so that $\{X_t^{\ell}\}_{\ell=1}^{k}$ partition the set X_t. Defining $c(i, X_t^{\ell}) = \min_{j \in X_t^{\ell}} w_i(\ell) c_{ij}$, one choice for dual variables is the following:

- $\beta_{i_t}^{\ell} := c(i_t, X_t^{\ell})$, $\ell \in [k]$, $t \in [n]$. Thus, the dual variable $\beta_{i_t}^{\ell}$ measures the instantaneous cost which is incurred by node i_t in the greedy algorithm to get access to resource ℓ. In particular, $\sum_{\ell=1}^{k} \beta_{i_t}^{\ell}$ is equal to the access cost of node i_t at the time of being processed by the greedy algorithm.
- $u_{ii_t}^{\ell} := \left(\beta_{i_t}^{\ell} - w_j(\ell) c_{ii_t}\right)^+ \ \forall i,t,\ell$, where $(a)^+ = \max\{0, a\}$. This is the minimum value to choose for $u_{ii_t}^{\ell}$ and satisfy the first set of dual constraints.
- $\alpha_{i_t} := \sum_{j \in X_{t-1}} \left(c(j, X_t^{\ell(i_t)}) - c_{i_t j}\right)^+ - f_{i_t}^{\ell(i_t)}$, $t \in [n]$. In other words, α_{i_t} measures the overall cost-saving to all the previously processed nodes due to assigning $\ell(i_t)$ to node i_t at time t.

Now, due to such a selection of dual variables, the cost of the dual objective is precisely equal to the cost of the greedy algorithm (which is constructed incrementally), i.e., $\sum_{t \in [n], \ell \in [k]} \beta_{i_t}^{\ell} - \sum_{t \in [n]} \alpha_{i_t} = z(\text{Greedy})$. Moreover, the first and the last sets of dual constraints are fully satisfied. Thus, an effective selection rule $\ell(i_t)$ for the greedy algorithm which also ensures the feasibility of the second set of dual constraints by possibly scaling down the dual variables by a constant factor $\frac{1}{\gamma}$ will establish a γ-

[7] For the sake of generality, we rather not determine any specific selection rule.

approximation ratio for the heterogeneous resource allocation problem. For more details on this method, we refer the interested readers to [25].

(2) It is known that finding a flip-optimal solution to Max-k-Cut on complete graphs with arbitrary edge weights is PLS-hard [41]. Generalizing this result to the case where the edge weights can only belong to $\{1, \ldots, 2^m\}$ is another interesting research problem. Such a generalization not only implies PLS-hardness of finding a NE in the resource allocation game using the BR updates, but it also establishes the same complexity result for a broader class of congestion games which include the above resource allocation game as a special case.

REFERENCES

[1] K. Aardal, F. A. Chudak, and D. B. Shmoys, *A 3-approximation algorithm for the k-level uncapacitated facility location problem*, Inform. Process. Lett., 72 (1999), pp. 161–167.

[2] H. Ackermann, H. Röglin, and B. Vöcking, *On the impact of combinatorial structure on congestion games*, J. ACM, 55 (2008), 25.

[3] I. Baev, R. Rajaraman, and C. Swamy, *Approximation algorithms for data placement problems*, SIAM J. Comput., 38 (2008), pp. 1411–1429, https://doi.org/10.1137/080715421.

[4] I. D. Baev and R. Rajaraman, *Approximation algorithms for data placement in arbitrary networks*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 661–670.

[5] E. G. Birgin, J. M. Martínez, and M. Raydan, *Nonmonotone spectral projected gradient methods on convex sets*, SIAM J. Optim., 10 (2000), pp. 1196–1211, https://doi.org/10.1137/S1052623497330963.

[6] A. Björklund, T. Husfeldt, and M. Koivisto, *Set partitioning via inclusion-exclusion*, SIAM J. Comput., 39 (2009), pp. 546–563, https://doi.org/10.1137/070683933.

[7] R. Carlson and G. L. Nemhauser, *Scheduling to minimize interaction cost*, Oper. Res., 14 (1966), pp. 52–58.

[8] R. Carosi, M. Flammini, and G. Monaco, *Computing approximate pure Nash equilibria in digraph k-coloring games*, in Proceedings of the 16th ACM Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 911–919.

[9] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, J. Comput. System Sci., 65 (2002), pp. 129–149.

[10] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, *Selfish caching in distributed systems: A game-theoretic analysis*, in Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, 2004, pp. 21–30.

[11] D.-Z. Du and X.-S. Zhang, *Global convergence of Rosen's gradient projection method*, Math. Programming, 44 (1989), pp. 357–366.

[12] S. R. Etesami and T. Başar, *Pure Nash equilibrium in a capacitated selfish resource allocation game*, IEEE Trans. Control Netw. Syst., 5 (2018), pp. 536–547.

[13] S. R. Etesami and T. Başar, *Price of anarchy and an approximation algorithm for the binary-preference capacitated selfish replication game*, Automatica, 76 (2017), pp. 153–163.

[14] A. Fabrikant, C. Papadimitriou, and K. Talwar, *The complexity of pure Nash equilibria*, in Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, 2004, pp. 604–612.

[15] R. Z. Farahani, M. SteadieSeifi, and N. Asgari, *Multiple criteria facility location problems: A survey*, Appl. Math. Model., 34 (2010), pp. 1689–1709.

[16] A. Frieze and M. Jerrum, *Improved approximation algorithms for max-k-cut and max bisection*, Algorithmica, 18 (1997), pp. 67–81.

[17] A. F. Gabor and J.-K. C. van Ommeren, *A new approximation algorithm for the multilevel facility location problem*, Discrete Appl. Math., 158 (2010), pp. 453–460.

[18] B. Ghosh and S. Muthukrishnan, *Dynamic load balancing in parallel and distributed networks by random matchings*, in Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, 1994, pp. 226–235.

[19] M. X. Goemans, L. Li, V. S. Mirrokni, and M. Thottan, *Market sharing games applied to content distribution in ad hoc networks*, IEEE J. Sel. Areas Commun., 24 (2006), pp. 1020–1033.

[20] R. Gopalakrishnan, D. Kanoulas, N. N. Karuturi, C. P. Rangan, R. Rajaraman, and

R. Sundaram, *Cache me if you can: Capacitated selfish replication games*, in LATIN 2012: Theoretical Informatics, Springer, Berlin, Heidelberg, 2012, pp. 420–432.
[21] S. Goyal and F. Vega-Redondo, *Learning, Network Formation and Coordination*, technical report, Tinbergen Institute, Amsterdam, Rotterdam, 2000.
[22] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.
[23] S. Guha and K. Munagala, *Improved algorithms for the data placement problem*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 106–107.
[24] H.-C. Huang and R. Li, *A k-product uncapacitated facility location problem*, European J. Oper. Res., 185 (2008), pp. 552–562.
[25] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani, *Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP*, J. ACM, 50 (2003), pp. 795–824.
[26] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.
[27] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi, *On the hardness of approximating Max k-Cut and its dual*, Chic. J. Theoret. Comput. Sci., no. 1997, 5.
[28] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Placement algorithms for hierarchical cooperative caching*, J. Algorithms, 38 (2001), pp. 260–302.
[29] J. Kun, B. Powers, and L. Reyzin, *Anti-coordination games and stable graph colorings*, in Algorithmic Game Theory, Springer, Heidelberg, 2013, pp. 122–133.
[30] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, *Distributed selfish replication*, IEEE Trans. Parallel Distrib. Syst., 17 (2006), pp. 1401–1413.
[31] R. Levi, D. B. Shmoys, and C. Swamy, *LP-based approximation algorithms for capacitated facility location*, in Integer Programming and Combinatorial Optimization, Springer, Berlin, 2004, pp. 206–218.
[32] M. Mahdian, Y. Ye, and J. Zhang, *Approximation algorithms for metric facility location problems*, SIAM J. Comput., 36 (2006), pp. 411–432, https://doi.org/10.1137/S0097539703435716.
[33] A. M. Masucci and A. Silva, *Strategic Resource Allocation for Competitive Influence in Social Networks*, preprint, https://arxiv.org/abs/1402.5388, 2014.
[34] M. T. Melo, S. Nickel, and F. Saldanha-Da-Gama, *Facility location and supply chain management—A review*, European J. Oper. Res., 196 (2009), pp. 401–412.
[35] I. Milchtaich, *Congestion games with player-specific payoff functions*, Games Econom. Behav., 13 (1996), pp. 111–124.
[36] D. Monderer and L. S. Shapley, *Potential games*, Games Econom. Behav., 14 (1996), pp. 124–143.
[37] V. Pacifici and G. Dan, *Convergence in player-specific graphical resource allocation games*, IEEE J. Sel. Areas Commun., 30 (2012), pp. 2190–2199.
[38] G. G. Pollatos, O. A. Telelis, and V. Zissimopoulos, *On the social cost of distributed selfish content replication*, in Networking, Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, Springer, Berlin, Heidelberg, 2008, pp. 195–206.
[39] R. Ravi and A. Sinha, *Multicommodity facility location*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 342–349.
[40] J. B. Rosen, *The gradient projection method for nonlinear programming. Part* I. *Linear constraints*, J. Soc. Indust. Appl. Math., 8 (1960), pp. 181–217, https://doi.org/10.1137/0108011.
[41] A. A. Schäffer and M. Yannakakis, *Simple local search problems that are hard to solve*, SIAM J. Comput., 20 (1991), pp. 56–87, https://doi.org/10.1137/0220004.
[42] T. Serafini, G. Zanghirati, and L. Zanni, *Gradient projection methods for quadratic programs and applications in training support vector machines*, Optim. Methods Softw., 20 (2005), pp. 353–378.
[43] D. B. Shmoys, É. Tardos, and K. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, 1997, pp. 265–274.
[44] T. Vredeveld and J. K. Lenstra, *On local search for the generalized graph coloring problem*, Oper. Res. Lett., 31 (2003), pp. 28–34.
[45] L. Wang, R. Li, and J. Huang, *Facility location problem with different type of clients*, Intell. Inf. Manag., 3 (2011), pp. 71–74.