Tight Approximation Algorithms for Maximum General Assignment Problems

Lisa Fleischer* Michel X. Goemans† Vahab S. Mirrokni† Maxim Sviridenko*

Abstract

A *separable assignment problem (*SAP*)* is defined by a set of bins and a set of items to pack in each bin; a value, f_{ij}, for assigning item j to bin i; and a separate packing constraint for each bin – i.e. for bin i, a family $\mathcal{I}_i$ of subsets of items that fit in bin i. The goal is to pack items into bins to maximize the aggregate value. This class of problems includes the *maximum generalized assignment problem* (GAP)[1]) and a distributed caching problem (DCP) described in this paper.

Given a β-approximation algorithm for finding the highest value packing of a single bin, we give

1. A polynomial-time LP-rounding based $((1 - \frac{1}{e})\beta)$-approximation algorithm.

2. A simple polynomial-time local search $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm, for any $\epsilon > 0$.

Therefore, for all examples of SAP that admit an approximation scheme for the single-bin problem, we obtain an LP-based algorithm with $(1 - \frac{1}{e} - \epsilon)$-approximation and a local search algorithm with $(\frac{1}{2} - \epsilon)$-approximation guarantee. Furthermore, for cases in which the subproblem admits a fully polynomial approximation scheme (such as for GAP), the LP-based algorithm analysis can be strengthened to give a guarantee of $1 - \frac{1}{e}$. The best previously known approximation algorithm for GAP is a $\frac{1}{2}$-approximation by Shmoys and Tardos; and Chekuri and Khanna. Our LP algorithm is based on rounding a new linear programming relaxation, with a provably better integrality gap.

To complement these results, we show that SAP and DCP cannot be approximated within a factor better than $1-\frac{1}{e}$ unless NP$\subseteq$ DTIME$(n^{O(\log\log n)})$, even if there exists a polynomial-time exact algorithm for the single-bin problem.

We extend the $(1 - \frac{1}{e})$-approximation algorithm to a nonseparable assignment problem with applications in maximizing revenue for budget-constrained combinatorial auctions and the AdWords assignment problem. We generalize the local search algorithm to yield a $\frac{1}{2} - \epsilon$ approximation algorithm for the k-median problem with hard capacities. Finally, we study naturally defined game-theoretic versions of these problems, and show that they have price of anarchy of 2. We also prove the existence of cycles of best response moves, and exponentially long best-response paths to (pure or sink) equilibria.

1 Introduction

In this paper, we study a general class of maximizing assignment problems with packing constraints. In particular, we study the *separable assignment problems (*SAP*)*: we are given a set U of n bins and a set H of m items, and a value, f_{ij}, for assigning item j to bin i. We are also given a separate packing constraint for each bin i; *packing* here refers to the assumption that if a subset is feasible for a bin then any subset of it is also feasible. The goal is to find an assignment of items to bins with the maximum aggregate value. For each bin, we define *the single-bin subproblem* as the optimization problem over feasible sets associated with the packing constraint for the bin.

This general class of problems includes several more specific problems as special cases, some well-studied, others newer and motivating our work here. All of these problems are NP-complete since for all of them, the knapsack problem is a special case.

*Maximum Generalized Assignment Problem (*GAP*):* Given a set of bins with capacity constraint and a set of items that have a possibly different size and value for each bin, pack a maximum-valued subset of items into the bins. This problem has several applications in inventory planning.

*Distributed Caching Problem (*DCP*):* This problem motivated our study of SAP. There is a set of n cache locations and m requests. Each cache location has a storage capacity and a bandwidth limit. Each request is of a particular request type, and a particular bandwidth. Given a set of request with request types where each request type has a size, the service provider decides 1) which request types to store at each cache location, subject to storage capacity; and 2) which subset of

*IBM T. J. Watson Research Center. Email: {lkf,sviri}@watson.ibm.com

†MIT Department of Mathematics. Email: goemans@math.mit.edu, mirrokni@theory.csail.mit.edu. Supported in part by NSF grants CCR-0098018 and ITR-0121495, and ONR grant N00014-05-1-0148.

[1]GAP is as follows: given a set of bins and a set of items that have a different size and value for each bin, pack a maximum-valued subset of items into the bins.

requests to answer subject to the request type selection and available bandwidth at the cache. Since the storage space for a request type in a cache location is independent of the number of requests for that type, the bandwidth required to serve requests of the same type is the sum of bandwidth requirements for the individual requests. For each potential assignment of request to cache, there is an associated profit which depends on the connection cost for the request to the cache. The goal is to maximize the profit of providing requests.

Our Results. Our results depend on an algorithm to solve the single-bin subproblem in SAP. In an instance of the single-bin subproblem, we are given a bin i, a set of items with value v_j for each item j, and a packing constraint for bin i, i.e., a lower-ideal[2] family, $\mathcal{I}_i$ of feasible subsets of items that can be packed in bin i. A β-approximation algorithm for $\beta \leq 1$ for the single-bin subproblem is an algorithm that outputs a subset of items $S \in \mathcal{I}_i$ such that for any other subset $S' \in \mathcal{I}_i$ of items, $\sum_{j \in S} v_j \geq \beta \sum_{j \in S'} v_j$.

Given a β-approximation algorithm for the single-bin subproblem, we give

1. A polynomial-time LP-rounding based $((1-\frac{1}{e})\beta)$-approximation algorithm.
2. A simple, polynomial-time local search $(\frac{\beta}{\beta+1}-\epsilon)$-approximation algorithm.[3]

For all of the specific problems mentioned above, there is an approximation scheme for the single-bin subproblem, so that we obtain an LP-based algorithm with $(1-\frac{1}{e}-\epsilon)$-approximation and a local search algorithm with $\frac{1}{2}-\epsilon$-approximation guarantee. Furthermore, if the single-bin subproblem admits an FPTAS, a more careful analysis gives a $1-\frac{1}{e}$-approximation algorithm for SAP. To complement these results, we show that SAP and DCP cannot be approximated within a factor better than $1-\frac{1}{e}$ unless NP$\subseteq$ DTIME$(n^{O(\log\log n)})$, even if there exists a polynomial-time exact algorithm for the single-bin subproblem.

We extend the $1-\frac{1}{e}$-approximation algorithm to a non-separable assignment problem with applications in maximizing revenue for the budget-constrained combinatorial auctions and the AdWords assignment problem. We generalize the local search algorithm to yield a $\frac{1}{2}-\epsilon$ approximation algorithm for the k-median problem with hard capacities. Finally, we study naturally defined game-theoretic versions of these problems, and show that they have price of anarchy of 2. We also prove the existence of cycles of best response moves, and exponentially long best-response paths to (pure or sink) equilibria. Thus a natural local search algorithm based on selfish bin behavior does not converge in polynomial time.

Due to the space constraint, some proofs are omitted in this extended abstract. We refer to Mirrokni [23] for the details of many missing proofs.

Previous Work. For the special case of GAP, Shmoys and Tardos [34] present an LP-rounding 2-approximation algorithm for the minimization problem. Chekuri and Khanna [6] observed that a $\frac{1}{2}$-approximation for GAP is implicit in [34]. Their method is based on an LP formulation with the integrality gap of at least 2. (Thus the LP we introduce here is provably stronger.) Chekuri and Khanna [6] develop PTAS's for a special case of this problem called the multiple knapsack problem. In this problem, each item has the same size and the same profit for all cache locations. They also classify the APX-hard special cases of GAP.

Fisher, Nemhauser, and Wolsey [10] consider the problem of maximizing a submodular function subject to the set being the independent set of a matroid. They describe both a simple greedy algorithm, and a local search algorithm that give an approximation guarantee of $\frac{1}{2}$. Chekuri [5] shows that SAP can be modelled as such a problem; his reduction is described in Section 3. Therefore, the proofs of Fisher et al. imply that if the single bin subproblem is solved optimally, their algorithms yield $\frac{1}{2}$ approximation for SAP. In this paper, we describe a similar local search algorithm, and show that it also yields a good guarantee for SAP even when the single bin subproblem is solved approximately.

Sviridenko shows that the greedy algorithm gives a $1-\frac{1}{e}$-approximation for maximizing a submodular function subject to one knapsack constraint [36], but does not consider assignment type problems with sets of packing constraints. Indeed the simple greedy algorithm does worse than $1-\frac{1}{e}$ even for the multiple knapsack problem. Using LP techniques [1, 35], approximation algorithms with the guarantee of $1-\frac{1}{e}$ for some maximum coverage problems are known. These techniques are different from ours and cannot handle SAP as the packing constraints in SAP are more general.

Gomes, Regis, and Shmoys [18] use a set packing LP and a rounding scheme similar to the one we use here but to obtain a $1-\frac{1}{e}$-approximation algorithm to solve the partial latin square extension problem. In particular, their LP does not capture knapsack packing constraints. Independent of our work, Nutov, Beniaminy, and Yuster [26] give a $1-\frac{1}{e}$-approximation algorithm for a special case of GAP with fixed profits. They use a different LP LP formulation that does not capture the general GAP.

Baev and Rajaraman [3] study a problem of data placement in networks. They formalize a minimization version of our problem in which they need to place objects in caches to minimize the total connection costs. They give a constant-factor approximation for this problem, which is improved to

[2] A family $\mathcal{I} \subseteq 2^H$ of subsets of a set H is lower-ideal, if for each two subsets S and R such that $R \subseteq S$, if $S \in \mathcal{I}$, then $R \in \mathcal{I}$.

[3] For the local search algorithm, we can set ϵ to be exponentially small, i.e., $\epsilon = \frac{1}{2^{cn}}$ for any constant $c > 0$.

factor 10 by Swamy [37]. In the conclusion of [3], they suggest considering the problem with bandwidth as an important extension.

Goemans, Li, Mirrokni, and Thottan [15], formalize a decentralized version of DCP in the context of the service provider 3G cellular networks. In this setting, *resident* subscribers are cache locations that can provide content to *clients*. However, [15] only models a special case of the problem by ignoring the bandwidth constraint and connection costs.

The Distributed Caching Problem (DCP)**.** Here, we formally define the distributed caching problems that are special cases of SAP. The general distributed caching problem that we formalize here is denoted by CapIBDC:

CapIBDC: Let U be a set of n cache locations with given available capacities A_i and given available bandwidths B_i for each cache location i. There are k request types;[4] each request type t has a size a_t ($1 \leq t \leq k$). Let H be a set of m requests with a reward R_j, a required bandwidth b_j, a request type t_j for each request j, and a cost c_{ij} for connecting each cache location i to each request j. The profit of providing request j by cache location i is $f_{ij} = R_j - c_{ij}$. A cache location i can service a set of requests S_i, if it satisfies the *bandwidth constraint*: $\sum_{j \in S_i} b_j \leq B_i$, and the *capacity constraint*: $\sum_{t \in \{t_j | j \in S_i\}} a_t \leq A_i$ (this means that the sum of the sizes of the request types of the requests in cache location i should be less than or equal to the available capacity of cache location i). We say that a set S_i of requests is feasible for cache location i if it satisfies both bandwidth and capacity constraints for bin i. The goal of the CapIBDC problem is to find a feasible assignment of requests to cache locations to maximize the total profit; i.e., the total reward of requests that are provided minus the connection costs of these requests.

We also consider the following special cases of the CapIBDC problem: The CapDC problem is a special case of CapIBDC problem without bandwidth constraint. The IBDC problem is a special case of CapIBDC problem without capacity constraint. From this definition, one can see that IBDC is a special case of CapDC where there exists exactly one request of each request type, the available capacity of cache locations in CapDC is equal to the available bandwidth of cache locations in IBDC, and the size of request types in CapDC corresponds to the bandwidth requirement of requests in IBDC. The uniform CapDC problem is a special case of the CapDC problem where the size of all request types is the same, i.e., $a_t = a$ for all $1 \leq t \leq k$. The uniform IBDC problem is a special case of the IBDC problem where the bandwidth requirement of all requests is the same, i.e., $b_j = b$ for all $j \in H$. We refer to all variants of the distributed caching problems as DCP. We also consider the fractional variant of CapIBDC, denoted by CapFBDC. In CapFBDC, a cache location may satisfy a request fractionally. The main reason to consider this setting is that a request may capture the requests for the same file in a region. In this case, some of these files may get service from one cache location and others from another cache location.

[4] Request type can be thought as different files that should be delivered to clients.

2 LP-Based Approximation Algorithms

In this section, we give a $((1 - \frac{1}{e})\beta)$-approximation for SAP and its variants where β is the approximation factor of the algorithm for the single-bin subproblem. The general approach is to formulate an (exponential-size) integer program, solve the linear program relaxation approximately, round the solution to the linear program, and prove that the rounded solution has this guarantee. There are two main issues here: proving the goodness of the rounded solution, and obtaining a good solution to the large linear program in polynomial time. We first discuss the approach in the context of SAP, and then discuss some extensions.

2.1 Separable Assignment Problems

Formulation. We give an exponential-size integer programming formulation for SAP. Let $\mathcal{I}_i$ for $i \in U$ be the set of all feasible assignments of items to bin i; these are the feasible solutions to the single-bin subproblem for bin i. For a set $S \in \mathcal{I}_i$, let X^i_S be the indicator variable that indicates if we choose S as the subset of items for bin i. The first constraint is that we cannot assign more than one set to a bin i, thus for all $i \in U$, $\sum_{S \in \mathcal{I}_i} X^i_S = 1$. Moreover, we cannot assign each item to more than one bin: $\sum_{i, S \in \mathcal{I}_i : j \in S} X^S_i \leq 1$. Our objective is to find an assignment of items to bins to maximize the sum of profits, i.e., $\sum_{i, S \in \mathcal{I}_i} f^S_i X^S_i$ where $f^S_i = \sum_{j \in S} f_{ij}$. By relaxing the 0-1 variables to nonnegative real variables, we obtain the following linear programming relaxation:

$$
\begin{array}{llll}
(2.1) & \max & \sum_{i \in U, S \in \mathcal{I}_i} f^S_i X^S_i & \\
& \text{s.t.} & \sum_{i \in U, S \in \mathcal{I}_i : j \in S} X^S_i \leq 1 & \forall j \in H \\
& & \sum_{S \in \mathcal{I}_i} X^S_i = 1 & \forall i \in U \\
& & X^S_i \geq 0 & \forall i \in U, \forall S \in \mathcal{I}_i
\end{array}
$$

Let LP(SAP) denote the objective function value of this LP.

Rounding the Fractional Solution. Given a solution to the linear program (2.1), independently for each bin i, assign set S to i with probability X^i_S. In the resulting solution, some item j may be contained in more than one of the sets assigned to the bins. In this case, item j is assigned to the bin among these bins with the maximum f_{ij}-value. Note that the resulting assignment after this step is feasible, since the family $\mathcal{I}_i$ for each bin i is lower-ideal.

THEOREM 2.1. *The expected value of the rounded solution* *is at least* $\left(1-\left(1-\frac{1}{n}\right)^n\right)$LP*(*SAP*).*

Proof. For item j, sort the bins i for which $Y_i = \sum_{S\in\mathcal{I}_i:j\in S} X_i^S$ is nonzero in the non-increasing order of f_{ij}. Without loss of generality assume that these bins are $1,2,\ldots,l$ and $f_{1j}\geq f_{2j}\ldots\geq f_{lj}\geq 0$. With probability Y_1 the set that is assigned to bin 1 contains j, thus item j is assigned to bin 1. In this case, the value of item j is f_{1j}. With probability $(1-Y_1)Y_2$, bin 1 does not have item j in its subset and bin 2 has item j in its set. In this case, the value of item j is f_{2j}. Proceeding similarly, we obtain that the expected value for request j in the rounded solution is $f_{1j}Y_1 + f_{2j}(1-Y_1)Y_2 + \ldots + f_{lj}(\Pi_{i=1}^{l-1}(1-Y_i))Y_l$. The contribution of item j in the value of the fractional solution is $\sum_{1\leq i\leq l} f_{ij}Y_i$. This in conjunction with Lemma 2.1 below yields the result.

LEMMA 2.1. *$f_{1j}Y_1 + f_{2j}(1-Y_1)Y_2 + \ldots + f_{lj}(\Pi_{i=1}^{l-1}(1-Y_i))Y_l \geq \left(1-\left(1-\frac{1}{l}\right)^l\right)\sum_{1\leq i\leq l} f_{ij}Y_i$ whenever $Y_i\geq 0$ for all i and $\sum_i Y_i\leq 1$ and $f_{1j}\geq f_{2j}\geq\ldots\geq f_{lj}\geq 0$.*

Proof. From the arithmetic/geometric mean inequality, one can derive (see Lemma 3.1 in [17]):

$$\begin{aligned} 1-(\prod_{i=1}^{k}(1-Y_i)) & \\ &= Y_1+(1-Y_1)Y_2+\cdots+(\prod_{i=1}^{k-1}(1-Y_i))Y_k \\ &\geq \left(1-\left(1-\frac{1}{k}\right)^k\right)\sum_{i=1}^{k} Y_i \end{aligned}$$

for any k. Multiplying this inequality by $f_{kj}-f_{k+1,j}\geq 0$ where $f_{l+1,j}=0$, summing over k and using the monotonicity of $(1-(1-1/k)^k)$, we derive the lemma.

Solving the LP. The number of variables in the linear program (2.1) is exponential. To solve this LP, we first solve its dual (2.2) given below.

$$\text{(2.2)}\quad \begin{array}{lll} \min & \sum_{i\in U} q_i + \sum_{j\in H}\lambda_j & \\ \text{s.t.} & q_i + \sum_{j\in S}\lambda_j \geq f_i^S & \forall i\in U, \forall S\in\mathcal{I}_i \\ & \lambda_j\geq 0 & \forall j\in H. \end{array}$$

The dual linear program (2.2) has a polynomial number of variables, but exponentially many constraints. We rewrite it as a fractional covering problem as follows:

$$\text{(2.3)}\quad \begin{array}{lll} \min & \sum_{i\in U} q_i + \sum_{j\in H}\lambda_j & \\ \text{s.t.} & (q_i,\lambda)\in\mathcal{P}_i & \forall i\in U \\ & \lambda_j\geq 0 & \forall j\in H. \end{array}$$

Here, $\mathcal{P}_i$ is the polytope defined by constraints of the form $q_i\geq\sum_{j\in S}(f_{ij}-\lambda_j)$ for all $S\in\mathcal{I}_i$. To solve the LP, we will need a separation algorithm for $\mathcal{P}_i$. We define an β-*approximate separation algorithm* for polytope $\mathcal{P}_i$ to be an algorithm that given a point $(q_i,\lambda_j|j\in H)$ returns either a violated constraint, or guarantees that $(\frac{q_i}{\beta},\lambda_j|j\in H)$ is feasible for $\mathcal{P}_i$. We let LP(Dual SAP) denote the objective function value of the linear program (2.2).

LEMMA 2.2. *For any $\delta>0$, given a polynomial-time β-approximate separation algorithm for $\mathcal{P}_i$, we can design a polynomial-time $(\beta-\delta)$-approximation algorithm to solve the linear program (2.2) and hence the linear program (2.1).*

The proof of this lemma is omitted here. The fact that, for a class of packing-covering linear programs, an approximate separation oracle for the dual implies an approximate solution for the primal is also observed by Carr and Vempala [4] and by Jain, Mahdian and Salavatipour [19]. An approximate solution to the linear programs (2.1) and (2.2) can also be obtained via Lagrangian LP algorithms [13, 30, 39, 40].

We can use a β-approximation algorithm for the single-bin subproblem for bin i to design a β-approximate separation algorithm for $\mathcal{P}_i$. The β-approximate separation algorithm for $\mathcal{P}_i$ asks, given $(q_i,\lambda_j|j\in H)$, find a set $S\in\mathcal{I}_i$ such that $q_i<\sum_{j\in S}(f_{ij}-\lambda_j)$. It is sufficient to find the set $S\in\mathcal{I}_i$ that maximizes $\sum_{j\in S}(f_{ij}-\lambda_j)$. Since $\mathcal{I}_i$ is lower-ideal, we know that if $q_i<0$ then the set $S=\emptyset$ violates the above inequality. Moreover, we can consider only items j for which $f_{ij}-\lambda_j$ is positive. In fact, we can set $\max(0,f_{ij}-\lambda_j)$ as the value of item j in the single-bin subproblem and use a β-approximation algorithm for the single-bin subproblem, to find a subset $S^*\in\mathcal{I}_i$ with value q_i^* such that for any set $S'\in\mathcal{I}_i$, $q_i^*=\sum_{j\in S^*}(f_{ij}-\lambda_j)\geq\beta\sum_{j\in S'}(f_{ij}-\lambda_j)$. We know that either $q_i^*>q_i$ in which case we find a violated constraint, or $q_i^*\leq q_i$. In the later case, we know that for any subset $S'\in\mathcal{I}_i$, $\sum_{j\in S'}(f_{ij}-\lambda_j)\leq\frac{q_i^*}{\beta}\leq\frac{q_i}{\beta}$. Therefore, in this case $(\frac{q_i}{\beta},\lambda_j|j\in U)$ is feasible for $\mathcal{P}_i$. Hence, a β-approximation algorithm for the single-bin subproblem is a β-approximate separation algorithm for $\mathcal{P}_i$. The above and previous discussion yields a guarantee of $\left(1-\left(1-\frac{1}{n}\right)^n\right)(\beta-\delta)$. Since δ can be chosen to be exponentially small (it determines the parameters of a binary search) and we have that $\left(1-\left(1-\frac{1}{n}\right)^n\right)\geq 1-\frac{1}{e}+\frac{1}{32n^2}$, we can remove the dependence on δ and obtain the following general result.[5]

THEOREM 2.2. *Given a polynomial-time β-approximation algorithm for the single-bin subproblem, there exists a polynomial-time $((1-\frac{1}{e})\beta)$-approximation algorithm for* SAP.

[5]The observation that one has enough room between $(1-1/e)$ and $(1-(1-1/n)^n)$ was pointed out to us by Raphael Yuster (similarly to Proposition 2.1 in [26]); see also the improvement in the forthcoming Theorem 2.4.

Solving the single-bin subproblem. In this section, we show that the single-bin problem for each problem class in SAP discussed in the introduction has an approximation scheme. Thus, for all problem classes, this yields polynomial-time $1 - \frac{1}{e} - \epsilon$-approximation algorithms.

GAP: the single-bin subproblem is a knapsack problem, for which an efficient FPTAS is well-known.

CapDC: the single-bin subproblem is to pack request types into the bin (respecting the bin capacity) to maximize the value of the items that can then be assigned to the bin. An item j can only be assigned if the corresponding type t_j is assigned. This is a knapsack problem, and thus has an FPTAS.

CapIBDC: the single-bin subproblem is the following 2-dimensional knapsack problem: bin i has A_i available space and B_i available bandwidth. Item $j \in S$ has value v_j, type t_j, and bandwidth consumption r_j. Each type t has size s_t. A feasible packing of items and types into bin i, satisfies total bandwidth of items in bin i is at most B_i, the total size of request types is at most A_i, and an item j is packed only if type t_j is also packed. The goal is to to maximize the total value of items packed in all bins. Shachnai and Tamir [33] describe a PTAS for this problem. We design a simpler PTAS which also handles a fractional variant, CapFBDC.

THEOREM 2.3. *There exists a polynomial-time* $(1 - \frac{1}{e} - \epsilon)$-*approximation algorithm for* GAP*,* CapDC*, and* CapIBDC *for any* $\epsilon > 0$.

Furthermore, if the subproblem admits an FPTAS as in GAP or CapDC, we can get rid of the ϵ by using the same argument as before Theorem 2.2 (and running the FPTAS with a guarantee of $1 - \frac{O(1)}{n^2}$). This was brought to our attention by Raphael Yuster. Thus,

THEOREM 2.4. *There exists a polynomial-time* $(1 - \frac{1}{e})$-*approximation algorithm for* GAP *and* CapDC.

2.2 Extensions to SAP.

The above general framework can be extended very slightly to also capture other classes of problems. Here we describe two extensions. The first adds a two-sided packing constraint: we are not only packing items in bins, but the items are in classes, and there is a limit, in terms of a budget constraint, on the set of items that can be packed in each class. The second extends DCP to fractional problems. The issue in this last case is how to solve the LP, since the number of feasible packings is no longer finite, and thus the straightforward modification of (2.1) yields a linear program with an infinite number of constraints.

The AdWords Assignment Problem (AAP) is as follows: we are given a set of n bidders with a limited budget B_i for each bidder i. Each bidder i also has an ad, Υ_i, to advertise. Ad Υ_i is a rectangle of width w_i and length l_i. We are also given a set of m AdWords. For each AdWord j, we can advertise the ads of a subset of bidders in a rectangular area of width W_j and length L_j. The rectangles for ads may not overlap and may not be rotated. For each AdWord j and each bidder i, we are given a value v_{ij} that is the maximum value that bidder i is willing to pay for AdWord j. For an assignment of AdWords to bidders, if we assign the set S_i of AdWords to bidder i, the revenue from bidder i is equal to $\min(B_i, \sum_{j \in S_i} v_{ij})$. Our goal is to find an assignment of AdWords to bidders to maximize the total revenue, i.e., the sum of revenue from all bidders.

In related work [2], a $(1-\frac{1}{e})$-approximation algorithm is given for budget-constrained auctions. Budget-constrained auctions [14, 2] are special cases of AAP where we can assign each AdWord to at most one bidder. AAP cannot be formalized as a separable assignment problem. However, it can be formulated as an exponential-size program similar to the linear program that we used for SAP. Let $\mathcal{J}_j$ be the family of the feasible sets of bidders for the AdWord j. In the linear program, for each AdWord j and each bidder i, there exists a variable z_{ij}: $z_{ij} = 1$ if bidder i is assigned to AdWord j and $z_{ij} = 0$ otherwise. For each AdWord j and each subset $T \subseteq \mathcal{J}_j$, there exists a variable Y_j^T: $Y_j^T = 1$ if set T of bidders is assigned to AdWord j and $Y_j^T = 0$ otherwise. We relax the 0-1 variables to real variables. The following are the primal and dual linear programs.

$$
\begin{array}{llll}
(2.4) & \max & \sum_{i \in U, j \in H} v_{ij} z_{ij} & \\
& \text{s.t.} & \sum_{T \in \mathcal{J}_j : i \in T} Y_j^T = z_{ij} & \forall i \in U, j \in H \\
& & \sum_{T \in \mathcal{J}_j} Y_j^T \leq 1 & \forall j \in H \\
& & \sum_{j \in H} v_{ij} z_{ij} \leq B_i & \forall i \in U \\
& & Y_j^T \geq 0 & \forall j \in H, \forall T \in \mathcal{J}_j \\
& & z_{ij} \geq 0 & \forall i \in U, j \in H
\end{array}
$$

$$
\begin{array}{llll}
(2.5) & \min & \sum_{i \in U} B_i q_i + \sum_{j \in H} \lambda_j & \\
& \text{s.t.} & \lambda_j + \sum_{i \in T} y_{ij} \geq 0 & \forall j \in H, \forall T \in \mathcal{J}_j \\
& & v_{ij} q_i - y_{ij} \geq v_{ij} & \forall j \in H, \forall i \in U \\
& & \lambda_j \geq 0 & \forall j \in H \\
& & q_i \geq 0 & \forall i \in U.
\end{array}
$$

Similar to solving linear program (2.1), and using a β-approximation polynomial-time algorithm for the rectangle packing problem for AdWord j, we can solve the primal linear program (6.6) within a factor β in polynomial time. Now, given a fractional solution, we round the solution as follows: for each AdWord j, we assign exactly one set of bidders to j and we assign a set T of bidders to j with probability Y_j^T. The following lemma proves that in the rounding process, we do not lose more than a factor $1 - \frac{1}{e}$.

LEMMA 2.3. *Let* B_i' *be the revenue of bidder* i *in the fractional solution and* E_i *be the expected revenue from bidder* i *in the rounded solution. Then* $E_i \geq B_i'(1 - \frac{1}{e})$.

Proof. We know that $B'_i = \sum_{j\in H} v_{ij} z_{ij}$. Let X_{ij} be the indicator random variable that indicates if AdWord j is assigned to bidder i in the rounded solution. Therefore, $E_i = \mathbf{E}[min(B_i, \sum_{j\in H} X_{ij} v_{ij})]$. We know that $\mathbf{Pr}[X_{ij} = 1] = z_{ij}$. Therefore, similar to the proof of Theorem 4 in [2], we can show that $E_i \geq B'_i(1 - \frac{1}{e})$.

The above lemma along with the $(\frac{1}{2} - \epsilon)$-approximation algorithm for the rectangle packing problem [20] give a 0.31-approximation algorithm for AAP. If the width of the rectangles of ads and the rectangular area for advertisement (W_j's) are the same, then the packing problem for each AdWord is a knapsack problem, therefore, our algorithm is a 0.63-approximation algorithm for this special case of AAP.

Fractional DCP. We reformulate the infinite linear program so that it has a finite, although exponential number of constraints. There is a PTAS for the corresponding separation problem, and thus we obtain polynomial time $(1 - \frac{1}{e} - \epsilon)$ approximation algorithm. Details are omitted in this extended abstract.

3 Local Search Algorithms

In this section, for any $\epsilon > 0$, we give a simple local search $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm for separable assignment problems given a β-approximation algorithm for the single-bin subproblem. This, in turn, gives a combinatorial $(\frac{1}{2} - \epsilon)$-approximation algorithm for GAP and all variants of DCP. We also show how to extend this algorithm to give a $(\frac{1}{2} - \epsilon)$-approximation algorithm for the k-median problem with hard capacities and packing constraints.

Fisher, Nemhauser, and Wolsey [10] describe both a simple greedy algorithm and a local search algorithm that both yield $\frac{1}{2}$-approximation guarantees for the problem of maximizing a submodular function subject to the constraint that the set is an independent set of a matroid, which we'll call *submodular function maximization over matroids (*SFMM*)*. Chekuri [5] gives a reduction (below) from SAP to SFMM, which implies that the algorithms in [10] yield $\frac{1}{2}$-approximation guarantees for SAP when $\beta = 1$.

Reduction to SFMM. Let $\mathcal{I}_i$ be the set of all feasible packings of bin i and let $\mathcal{I} = \cup_i \mathcal{I}_i$ (with repeated copies for sets feasible for different bins), so that each element $S \in \mathcal{I}$ corresponds to a set of items that can packed in some bin. A feasible solution for SAP is a set of sets $A \subseteq \mathcal{I}$ such that $|A \cap \mathcal{I}_i| \leq 1$ for all i. The goal is to maximize $f(A)$ where $f(A)$ is the profit obtained by the sets in A. Define this as $\sum_{j\in H} max_{S\in A} f_{S,j}$. It can be verified that $f(A)$ is a submodular set function over $\mathcal{I}$. The constraint that one picks at most one element of $\mathcal{I}$ from each $\mathcal{I}_i$ is a simple matroid constraint.

3.1 Our Local Search Algorithm.

We first give a naive local search $\frac{\beta}{\beta+1}$-approximation algorithm whose running time might be exponential. Then, we refine the algorithm and change it to a polynomial-time algorithm. Let $\mathcal{S} = (S_1, \ldots, S_n)$ be an assignment of items to bins, where S_i is the set of items in bin i. For an assignment $\mathcal{S} = (S_1, \ldots, S_n)$ of items to bins, we denote the value of this assignment by $v(\mathcal{S})$. Also, let $\alpha_i(\mathcal{S})$ be the total value of items satisfied by bin i in $\mathcal{S}$. For an item j, let $v_j(\mathcal{S})$ the value of item j in $\mathcal{S}$.

The naive algorithm repeatedly iterates over the bins. For bin i, it runs procedure $\mathsf{Local}(i)$. $\mathsf{Local}(i)$, given current solution $\mathcal{S}$, finds a repacking S'_i of bin i. If replacing S_i with S'_i improves the solution then this replacement is made. When no further improvements can be made on any bin, the algorithm halts. We call the result an *β-approximate local optimal* solution. Specifically, $\mathsf{Local}(i)$ does the following:

1. For each item j, let $\mathsf{value}_j(\mathcal{S})$ be equal to $f_{i'j}$ if j is assigned to a bin $i' \neq i$ in $\mathcal{S}$, and be equal to zero if j is unassigned or is assigned to bin i in $\mathcal{S}$.

2. For each item j, let the *marginal value* of j be $w_j = f_{ij} - \mathsf{value}_j(\mathcal{S})$.

3. Use the β-approximation algorithm for the single-bin subproblem for bin i to pack a subset of items in bin i with the maximum *marginal value*.

LEMMA 3.1. *Let $\mathcal{S} = (S_1, \ldots, S_n)$ be a β-approximate local optimal solution and* OPT *teh value of the optimum assignment. Then $v(\mathcal{S}) \geq \frac{\beta \mathrm{OPT}}{\beta+1}$.*

The proof of this lemma is ommited. Lemma 3.1 shows that if we can find an β-approximate local solution then we have a $\frac{\beta}{\beta+1}$-approximation algorithm. We prove it is PLS-hard to find a local solution. The proof of this fact is very similar to the proof of Theorem 6.7. An implication of the PLS-hardness of this problem is that there exists a set of instances for which the above local search algorithm may take exponential time to converge to a local optimal solution. Below, we modify the naive algorithm to get a polynomial-time $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm. The analysis of this algorithm uses the following fact (which we prove in the proof of Theorem 3.1). Using this fact, we show that after a polynomial number of local improvements the value of the solution is a good approximate solution.

FACT 3.1. *If $v(\mathcal{S}) \leq \frac{\beta}{\beta+1}\mathrm{OPT}$ then there is a bin i for which $\mathsf{Local}(i)$ finds a packing with marginal value at least $\frac{\beta}{n}\mathrm{OPT} - \frac{1+\beta}{n} v(\mathcal{S})$.*

Local Search Algorithm.

1. Start with the empty solution, i.e., $\mathcal{S} = (S_1, \ldots, S_n)$ and $S_i = \emptyset$ for all $i \in U$.

2. For an appropriate $\epsilon' > 0$, run the following loop for $\frac{1}{\beta} n \ln(\frac{1}{\epsilon'})$ times:

(a) Let the current assignment be $\mathcal{S} = (S_1, \ldots, S_n)$.

(b) For each bin i, run $\mathsf{Local}(i)$. Let the marginal value of this solution for bin i be W_i and let S'_i be the set of items with marginal value W_i.

(c) For each bin i, let $\Delta_i = W_i - \alpha_i(\mathcal{S})$.

(d) Let bin i^* be the bin with the maximum Δ_i, i.e., $\Delta_{i^*} \geq \Delta_i$ for any bin i.

(e) If $\Delta_{i^*} > 0$, change the set S_{i^*} of items for bin i^* to S'_{i^*}.

THEOREM 3.1. *For any $\epsilon > 0$, the above local search algorithm is a polynomial-time $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm for* SAP.

Proof. Let Ω be an optimal assignment, and let $\mathcal{S}$ be an intermediate assignment obtained in the local search algorithm. Let R be the set of items that are better served in Ω than in $\mathcal{S}$ and L be the rest of the items. Let R_i be the set of items in R satisfied by bin i in Ω. For any set T of items, let $o(T)$ be the value of the items in T in assignment Ω and $l(T)$ be the value of items of T in assignment $\mathcal{S}$. Thus, we have $\mathrm{OPT} = o(R) + o(L)$ and $v(\mathcal{S}) = l(R) + l(L)$. For each item $j \in R_i$, the marginal value of j is $f_{ij} - \mathsf{value}_j(\mathcal{S}) \geq f_{ij} - v_j(\mathcal{S})$. Thus, the total marginal value of items in set R_i for bin i is at least $\sum_{j\in R_i}(f_{ij} - v_j(\mathcal{S})) = o(R_i) - l(R_i)$ and R_i is a feasible solution for bin i. Since we use a β-approximation algorithm to find W_i, $W_i \geq \beta(o(R_i) - l(R_i))$. Therefore, $\sum_{i\in U} W_i \geq \beta \sum_{i\in U}(o(R_i) - l(R_i)) = \beta(o(R) - l(R))$. Since $o(L) \leq l(L)$, $\sum_{i\in U} W_i \geq \beta(o(R) - l(R) + o(L) - l(L)) = \beta(\mathrm{OPT} - v(\mathcal{S}))$. Thus,

$$\begin{aligned}\sum_{i\in U}\Delta_i &= \sum_{i\in U} W_i - \sum_{i\in U}\alpha_i(S)\\ &\geq \beta(\mathrm{OPT} - v(\mathcal{S})) - v(\mathcal{S})\\ &= \beta\mathrm{OPT} - (1+\beta)v(\mathcal{S}).\end{aligned}$$

In particular, $\Delta_{i^*} \geq \frac{\beta}{n}\mathrm{OPT} - \frac{1+\beta}{n}v(\mathcal{S})$. Let $\mathcal{S}'$ be the assignment after changing the set of items of i^* to S'_{i^*}, i.e., $\mathcal{S}' = (S_1, S_2, \ldots, S_{i^*-1}, s'_{i^*}, S_{i^*+1}, \ldots, S_n)$. As a result,

$$\begin{aligned}v(\mathcal{S}') &= v(\mathcal{S}) + \Delta_{i^*}\\ &\geq v(\mathcal{S}) + \frac{\beta}{n}\mathrm{OPT} - \frac{1+\beta}{n}v(\mathcal{S})\\ &= v(\mathcal{S})(1 - \frac{1+\beta}{n}) + \frac{\beta}{n}\mathrm{OPT}.\end{aligned}$$

Let y_k be the total value of the assignment after the k^{th} execution of Step 2. From the above discussion, $y_k \geq (1 - \frac{1+\beta}{n})y_{k-1} + \frac{\beta}{n}\mathrm{OPT}$ and $y_0 = 0$. Using induction, we get that for any $1 \leq i \leq k$: $y_k \geq \frac{\beta}{1+\beta}(1 - (1 - \frac{\beta+1}{n})^k)\mathrm{OPT}$. By setting $k = \frac{n\ln(\frac{1}{\epsilon'})}{\beta+1}$, we get $y_k \geq \frac{\beta}{1+\beta}(1 - \frac{1}{e^{\ln(\frac{1}{\epsilon'})}})\mathrm{OPT} = \frac{\beta}{1+\beta}(1-\epsilon')\mathrm{OPT}$. Therefore, for $\epsilon' = \epsilon\frac{1+\beta}{\beta}$, the value of the output of the above algorithm is at least $(\frac{\beta}{1+\beta} - \epsilon)\mathrm{OPT}$ as desired.

3.2 k-Median with Hard Capacities and Packing Constraints. We can extend the local search algorithm for SAP to the k-median with hard capacities and packing constraints(KMed). KMed is as follows: Given a set of bins and a single-bin subproblem for each bin i, and also a set of items that have different value for each bin, choose at most K bins and pack a set of items in each selected bin to maximize the total value packed. To the best of our knowledge, this is the first constant-factor approximation algorithm for the the k-median problem with hard capacity constraints.

The local search algorithm for the KMed problem is very similar to the local search algorithm for SAP. At each step of the algorithm, we try to unpack a used bin, and pack a (possibly different) bin to increase the total value. The formal description of the algorithm and the analysis is omitted here. Thus, we get:

THEOREM 3.2. *Given a polynomial-time α-approximation algorithm for the single-bin subproblem, there is a polynomial-time $(\frac{\alpha}{\alpha+1} - \epsilon)$-approximation algorithm for* KMed.

4 A Hardness Result

In this section, we show a hardness result for the CapDC problem and special cases of SAP. We prove that the CapDC problem is not approximable better than a factor of $1 - \frac{1}{e}$ unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$ showing that the $1 - \frac{1}{e}$-approximation algorithm for CapDC is tight. This hardness result uses a hardness result by Feige, Halldorson, Kortsarz, and Srinivasan [8] for the domatic number of graphs. The *domatic number* of a graph is the maximum number of disjoint dominating sets in the graph. A subset S of vertices of a graph $G(V,E)$ is a *dominating set* if for any vertex $v \notin S$, there exists a vertex $u \in S$ which is connected to v, i.e., $(u,v) \in E(G)$. We first define a set of problems that are used in the reduction and restate the result of Feige et al. [8].

The Max 3-colorability problem is as follows: Given a graph $G = (V,E)$ color the vertices of G with 3 colors to maximize the number of legally colored edges (edges whose endpoints are colored differently). The Max 3-colorability-5 problem is the Max 3-colorability problem for 5-regular graphs. Petrank [29] proved that the Max 3-colorability problem is APX-hard. Using this proof, Feige et al. [8] proved that the Max 3-colorability-5 problem is APX-hard. Formally, they showed that for some number $\delta < 1$, it is NP-hard to distinguish between 5-regular graphs that have a legal 3-coloring, and 5-regular graphs in which every 3-coloring legally colors at most δ fraction of the edges. The following claim is implicit in the hardness result of Feige et al. [8]: Given an instance $G = (V,E)$ of the Max 3-colorability-5 problem, we can construct an instance of a set cover problem with $m = O(|V|^{O(\log\log|V|)}|E|^{O(\log\log|V|)})$

elements and $n = O(|V|^{O(\log\log|V|)}|E|^{O(\log\log|V|)})$ sets of size $\frac{m}{p}$ where $\frac{m}{p} = O(|V|^{O(\log\log|V|)}|E|^{O(\log\log|V|)})$ such that: (i) If the vertices of graph G are (legally) 3-colorable, then there exist $\frac{n}{p}$ disjoint set covers, each with p sets[6] in the set cover instance; and (ii) If any 3-coloring of G has less than $\delta|E|$ legally colored edges then any collection of βp sets cover at most $(1-(1-\frac{1}{p})^{\beta p})m$ elements[7].

From an instance of the set cover problem with n sets and m elements, we construct an instance of CapDC with $\frac{n}{p}$ types, $m \cdot \frac{n}{p}$ requests, and n cache locations as follows: For each element j in the ground set of the set cover, we put $\frac{n}{p}$ requests $j_1, j_2, \ldots, j_{\frac{n}{p}}$ of different types in the CapDC instance. For each set i in the set cover instance, we put a cache location i in the CapDC instance. The capacity A_i for cache location i is $A_i = 1$ and the size of each request type is equal to 1. Thus, we can locate at most one type in each cache location. The profit of assigning request j_e to cache location i is 1 if the corresponding element j is in the corresponding set in the set cover instance. If the set cover instance has $\frac{n}{p}$ disjoint set covers then in the instance of the CapDC problem, we can satisfy all requests of a particular type using one set cover and thus, we can find a solution to the instance of the CapDC problem with a total profit of $\frac{mn}{p}$. Moreover, we claim that if any collection of βp sets in the set cover problem, cover at most $(1-(1-\frac{1}{p})^{\beta p})m$ of elements then the profit of any assignment to the CapDC problem is at most $(1-(1-\frac{1}{p})^{p})\frac{mn}{p}$. Assume that in the set cover instance, any collection of βp sets cover at most $(1-(1-\frac{1}{p})^{\beta p})m$ of the elements. Consider a solution $\mathcal{S}$ with the maximum profit for the CapDC problem. For $1 \le t \le \frac{n}{p}$, let $\alpha_t p$ be the number of cache locations that keep the request type t in solution $\mathcal{S}$. We know that $\sum_{t=1}^{\frac{n}{p}} \alpha_t p = n$. Also from the inequality $(1-(1-\frac{1}{p})^{xp}) + (1-(1-\frac{1}{p})^{tp}) \le (1-(1-\frac{1}{p})^{yp}) + (1-(1-\frac{1}{p})^{zp})$ where $x \le y \le z \le t$ and $x+t = y+z$, it follows that the profit of $\mathcal{S}$ maximizes when all α_t for $1 \le t \le \frac{n}{p}$ are the same, i.e., $\alpha_t = 1$. By setting $\alpha_t = 1$, we have that the profit of $\mathcal{S}$ is at most $\sum_{t=1}^{\frac{n}{p}}(1-(1-\frac{1}{p})^{\alpha_t p})m \le \sum_{t=1}^{\frac{n}{p}}(1-(1-\frac{1}{p})^{p})m \le (1-(1-\frac{1}{p})^{p})\frac{mn}{p}$.

Therefore, if we apply the Feige et al. reduction from Max 3-colorobality-5 to the set cover and the above reduction from the set cover to the CapDC problem, we have the following: Given an instance of Max 3-colorability-5 problem, we can construct an instance of the CapDC problem with $\frac{n}{p}$ types, $m \cdot \frac{n}{p}$ requests, and n cache locations such that: (i) If vertices of graph G are (legally) 3-colorable, then there exists a solution with profit $\frac{mn}{p}$ for the CapDC instance; and (ii) If any 3-coloring of G has less than $\delta|E|$ legally colored edges, the maximum possible profit of the CapDC instance is at most $1-(1-\frac{1}{p})^p$ of the number of requests, i.e., $(1-(1-\frac{1}{p})^p)\frac{mn}{p}$.

Note that $(1-(1-\frac{1}{p})^p)$ tends to $1-\frac{1}{e}$ as p tends to ∞. Therefore, for any $\epsilon > 0$, there exists a sufficiently large p such that $(1-(1-\frac{1}{p})^p) \le (1-\frac{1}{e}) + \epsilon$. Hence, for any $\epsilon > 0$, for any sufficiently large instance of Max 3-colorability-5 problem, we can construct an instance of the CapDC problem with $\frac{n}{p}$ types, $m \cdot \frac{n}{p}$ requests, and n cache locations such that:

- If vertices of graph G are (legally) 3-colorable, then there exists a solution with profit $\frac{mn}{p}$ for the CapDC instance.

- If any 3-coloring of G has less than $\delta|E|$ legally colored edges, the maximum possible profit of the CapDC instance is at most $1-\frac{1}{e}+\epsilon$ of the number of requests, i.e., $(1-\frac{1}{e}+\epsilon)\frac{mn}{p}$.

This shows that for any $\epsilon > 0$, if we can approximate the CapDC problem within a factor better than $1-\frac{1}{e}+\epsilon$ then we can distinguish between the aforementioned cases of the sufficiently large instances of the Max 3-colorability-5 problem in time $O(V^{O(\log\log V)}E^{O(\log\log V)})$. Since distinguishing between these two cases of the Max 3-colorability-5 problem is NP-hard, if we can approximate the CapDC problem in polynomial time within a factor of $1-\frac{1}{e}+\epsilon'$ for $\epsilon' < \epsilon$, then NP$\subseteq$ DTIME$(n^{O(\log\log n)})$. Note that in the above reduction, we only used instances of the CapDC problem with uniform sizes and uniform capacities, this shows that even the uniform CapDC problem is not approximable within a factor better than $1-\frac{1}{e}$ unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$. In particular, it means that there are instances of SAP in which the single-bin subproblem is solvable in polynomial time, but the multiple-bin SAP problem is not approximable within a factor better than $1-\frac{1}{e}$ unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$.

5 A Decentralized Mechanism

In this section, we explore methods to obtain decentralized algorithms with a good performance. Motivated from service provider cellular networks [15], we may assume that bins or cache locations are selfish agents (resident subscribers) who want to maximize their own profit. We state the results of this section in terms of DCP terminology, but all the results hold

[6] See Lemma 17 of [8]. In fact, Feige et al. [8] present their result in terms of the dominating set and the domatic number problem. We restate their result for the set cover problem.

[7] The proof of this claim comes from the proof of Lemma 18 of [8], in which authors refer to the hardness result for the set cover problem by Feige (e.g. see Proposition 4.3 of [7]). Essentially, the proof of our claim comes from the fact that for the construction in [7] applied to our problem, any collection of βp sets cover at most $(1-(1-\frac{1}{p})^{\beta p})m$ elements [9]. The reason is that the number of elements that βp sets cover is less than the expected number of elements that βp random sets of size p cover where a random set is a set in which each element is picked uniformly at random and independent of other elements. We also note that p is not a constant. In particular, as $|V|$ tends to ∞, p also tends to ∞.

for separable assignment problems. Proofs of this section are omitted here and can be found in Mirrokni [23].

We define the *distributed caching game* in the setting of CapIBDC - with both capacity and bandwidth constraints. Each cache location is a player of this game. The strategy of a player i is to choose a subset of requests D_i such that the total bandwidth of these requests is within the bandwidth of the player ($\sum_{j \in D_i} b_j \leq B_i$) and the total capacity of types of these requests is less than the capacity of the player ($\sum_{t \in \{t_j \mid j \in D_i\}} a_t \leq A_i$). A strategy profile $\mathcal{S}$ is a vector of strategies for each player. Given $\mathcal{S}$, each request goes to a cache location with the minimum connection cost among the set of cache locations that provide this request. The profit of this request is evenly divided between the cache locations with the minimum connection cost. The reward of a cache location is the sum of profits of the requests it actually serves. The *social value* of strategy profile $\mathcal{S}$, denoted by $v(\mathcal{S})$, is the sum of profits of all players. This value $v(\mathcal{S})$ is a measure of the efficiency of the assignment of requests and request types to cache locations. We assume that cache locations are selfish and choose strategies to maximize their own profit. Since the distributed caching game is a strategic game, it has (mixed) Nash equilibria [24]. We prove that in a Nash equilibrium of this game, the social value is at least $\frac{1}{2}$ of the optimal social value.

THEOREM 5.1. *The price of anarchy of the distributed caching game for a mixed Nash equilibrium is at most 2.*

We can show that this game is a valid-utility game. These games are defined by Vetta [38]. The results in [38] imply that the price of anarchy is 2. However, it is simpler to give a direct proof of Theorem 6.2. We further investigate existence of pure strategy Nash equilibria in these games and prove the following:

THEOREM 5.2. *There are instances of* CapDC *game that have no pure Nash equilibria. Moreover, there exist cycles of best responses in the uniform* CapDC *game.*

Finally, we prove that there are instances of the uniform CapDC game in which finding a pure Nash equilibrium is PLS-complete [21, 28, 32].

THEOREM 5.3. *There are instances of the uniform* CapDC *game with pure Nash equilibria*[8] *for which finding a pure Nash equilibrium is PLS-complete.*

Proof. We give a reduction from the local search Max-Cut problem with swapping neighborhood to the uniform CapDC game.

[8] We can also say that finding a *sink equilibrium* [16, 23] is PLS-complete. A sink equilibrium is a set of strategy profiles that is closed under best-response moves. A pure equilibrium is a sink equilibrium with exactly one profile [16, 23].

Consider an instance $G = (V, E)$ of the Max-Cut problem with weights $w : E \to N$ on edges. We construct an instance $\mathcal{R}(G)$ of the CapDC game as follows: Each player corresponds to a vertex of graph G. There are two types of requests: p and q. The size of each request type is equal to one and the capacity of each cache location is one. For each edge, we define one request each of type p and q: request p_{uv} and request q_{uv}. The connection costs of both p_{uv} and q_{uv} to either u or v is zero and to any other vertex is $w_{uv} + 1$. The reward of each of these two requests is w_{uv}.

Each player caches either request type p or q. For each edge (u, v), if both u and v cache requests of the same type then they each get profit $\frac{w_{uv}}{2}$. If they cache different types then they each get profit w_{uv}. Thus, given a strategy profile, the total profit obtained is $w(E)+w(C)$ where C is the set of edges with one player caching type p and the other caching type q. In the Max-Cut problem, C corresponds to the cutset defined by the cut with all vertices that cache type p on one side and all vertices that cache type q on the other. Call this cut $\mathcal{M}(\mathcal{S}, G)$.

Under this profit sharing scheme, it is clearly in player u's best interest to include all requests of the type it caches that correspond to edges incident to u in G. Let $\mathcal{L}$ be the set of strategy profiles $\mathcal{S}$ of the instance $\mathcal{R}(G)$ that have this form.

In a strategy profile $\mathcal{S} \in \mathcal{L}$ of game $\mathcal{R}(G)$, player u can strictly improve his payoff by playing switching from request type p to request type q if and only if the value of the cutset $\mathcal{M}(\mathcal{S}, G)$ strictly improves by moving u to the other side of the cut. Thus, if a strategy profile $\mathcal{S} \in \mathcal{L}$ is a pure Nash equilibrium in game $\mathcal{R}(G)$, then $\mathcal{M}(\mathcal{S}, G)$ is a local optimal solution of the Max-Cut local search problem for graph G.

Thus, if we have a polynomial-time algorithm for finding a pure Nash equilibrium $\mathcal{S}$ (or a sink equilibrium) in any instance of the uniform CapDC game, since this pure Nash equilibrium is in $\mathcal{L}$, this implies a polynomial-time algorithm for finding a local optimum $\mathcal{M}(\mathcal{S}, G)$ of any instance of the Max-Cut local search problem with swapping neighborhood. The PLS-hardness follows from a PLS-completeness result of Schaffer and Yannakakis [32].

Acknowledgements. We thank Li Li for helpful discussions on the problem formulation and motivations from cellular networks. We thank Raphael Yuster for pointing out the stronger version of the inequality of Lemma 2.1 that lead to Theorem 2.4. And we thank Chandra Chekuri for pointing out the reduction of SAP to SFMM described in Section 3.

References

[1] A. A. Ageev and M. I. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. of Combinatorial Optimization*, 8:307–328, 2004.

[2] N. Andelman and Y. Mansour. Auctions with budget constraints. In *SWAT*, 2004.

[3] I. D. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *12th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 661–670, 2001.

[4] B. Carr and S. Vempala. Randomized meta-rounding. *Random Structure and Algorithms*, 20(3):343–352, 2002.

[5] C. Chekuri. Personal communication. 2005.

[6] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *11th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 213–222, 2000.

[7] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634 – 652, 1998.

[8] U. Feige, M. M. Halldorsson, G. Kortsarz, and A. Srinivasan. Approximating the domatic number. *SIAM Journal of Computing*, 32(1):172–195, 2002.

[9] U. Feige and G. Kortsarz. Personal communication. 2005.

[10] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of the approximations for maximizing submodular set functions II. *Mathematical Programming Study*, 8:73–87, 1978.

[11] T. Fleiner. A fixed-point approach to stable matchings and some applications. *Mathematics of Operations Research*, 28(1):103–126, 2003.

[12] S. Fujishige. *Submodular Functions and Optimization*. North-Holland, 1991.

[13] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[14] R. Garg, V. Kumar, and V. Pandit. Approximation algorithms for budget-constrained auctions. In *4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2001.

[15] M. Goemans, L. Li, V. S. Mirrokni, and M. Thottan. Market sharing games applied to content distribution in ad-hoc networks. In *5th ACM International Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2004.

[16] M. Goemans, V. S. Mirrokni, and A. Vetta. Sink equilibria and convergence. In *FOCS*, 2005.

[17] M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal of Discrete Mathematics*, 7:656–666, 1994.

[18] C. Gomes, R. Regis, and D. Shmoys. An improved approximation algorithm for the partial Latin square extension problem. In *14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2003.

[19] K. Jain, M. Mahdian, and M. Salavatipour. Packing stiner trees. In *14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 266–274, 2003.

[20] K. Jansen and G. Zhang. On rectangle packing: maximizing benefits. In *15th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 204 – 213, 2004.

[21] D. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[22] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *16th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 404–413, 1999.

[23] V. S. Mirrokni. *Approximation Algorithms for Distributed and Selfish Agents*. MIT Thesis, 2005.

[24] J. F. Nash. Equilibrium points in N-person games. In *Proceedings of NAS*, 1950.

[25] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:268–295, 1951.

[26] Z. Nutov, I. Beniaminy, and R. Yuster. A (1-1/e)-approximation algorithm for the maximum profit generalized assignment problem with fixed profits. *Operations Research Letters*. To appear.

[27] C. Papadimitriou. Algorithms, games, and the internet. In *33rd ACM Symp. on Theory of Computing (STOC)*, pages 749–753, 2001.

[28] C.H. Papadimitriou, A. Schaffer, and M. Yannakakis. On the complexity of local search. In *22nd Symp. on Theory of Computing (STOC)*, pages 438 – 445, 1990.

[29] E. Petrank. The hardness of approximation: Gap location. In *Computational Complexity*, pages 133–137, 1994.

[30] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[31] A.E. Roth and J. H. Vande Vate. Random paths to stability in two-sided matchings. *Econometrica*, 58(1):1475–1480, 1990.

[32] A. Schaffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM journal on Computing*, 20(1):56–87, 1991.

[33] H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 167–177, 2003.

[34] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3):461–474, 1993.

[35] A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *44th Symp. on Foundations of Computer Science (FOCS)*, pages 588–597, 2001.

[36] M. Sviridenko. A note on maximizing a submodular set function subject to knapsack constraint. *Operations Research Letters*, 32:41–43, 2004.

[37] C. Swamy. Algorithms for the data placement problem. Unpublished, 2004.

[38] A. Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In *43rd Symp. on Foundations of Computer Science (FOCS)*, pages 416–425, 2002.

[39] N. Young. Randomized rounding without solving the linear program. In *7th ACM-SIAM Symp. on Discrete Algorithms*

(SODA), pages 170–178, 1996.
[40] N. Young. Sequential and parallel algorithms for mixed packing and covering. In *42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 538–546, 2001.

6 Appendix

6.1 Proof of a lemma

Proof. **Proof of Lemma 2.2**

We run the ellipsoid algorithm on the linear program (2.2) using a β-approximate separation algorithm. More precisely, we move the objective into the set of constraints by adding the constraint $\sum_{i\in U} q_i^* + \sum_{j\in H}\lambda_j^* \le v^*$ to the current linear program. For a given v^*, we use the ellipsoid algorithm to determine if this LP is feasible; and use binary search to find the smallest feasible value v^*. Using the ellipsoid algorithm in this binary search framework with a β-approximate separation algorithm, suppose that the process of the algorithm terminates with a solution $(q_i^*, \lambda_j^* | j \in H)$ such that $v^* = \sum_{i\in U} q_i^* + \sum_{j\in H}\lambda_j^*$. Thus, we know that the linear program (2.2) with the new constraint is infeasible for $v^* - \delta'$ where δ' depends on the precision of the binary search[9]. Thus, the optimal solution to the LP (2.2) is at least $v^* - \delta'$. Since we use a β-approximate separation algorithm, we are not guaranteed that this solution is feasible. However, we know that $(\frac{q_i^*}{\beta}, \lambda_j^* | j \in H)$ is feasible. Thus, the optimal solution to the LP (2.2) is at most $\frac{v^*}{\beta}$. Thus the optimal solution to (2.2) is between $v^* - \delta'$ and $\frac{v^*}{\beta}$.

In the execution of the ellipsoid algorithm for $v^* - \delta'$, we check a polynomial number of constraints. This set of constraints is enough to show that the value of the dual is greater than $v^* - \delta'$. The dual of this restricted LP is equivalent to the linear program (2.1) restricted to the variables corresponding to this set of constraints (by setting all other variables to zero). By LP-duality, the cost of the solution to this program is at least $v^* - \delta'$. Thus, the solution to this polynomial-sized LP has value at least $v^* - \delta'$. By setting δ' sufficiently small, this is an $(\beta - \delta)$-approximation algorithm for the primal linear program, since $\mathsf{LP(SAP)} \le \mathsf{LP(Dual\ SAP)} \le \frac{v^*}{\beta}$.

6.2 General budget-constrained combinatorial auctions: Adwords.

We write an exponential-size linear program for AAP. Let $\mathcal{J}_j$ be the family of the feasible sets of bidders for the AdWord j. In the linear program, for each AdWord j and each bidder i, there exists a variable z_{ij}: $z_{ij} = 1$ if bidder i is assigned to AdWord j and $z_{ij} = 0$ otherwise. For each AdWord j and each subset $T \subseteq \mathcal{J}_j$, there exists a variable Y_j^T: $Y_j^T = 1$ if set T of bidders is assigned to AdWord j and $Y_j^T = 0$ otherwise. We relax the 0-1 variables to real variables. The following are the primal and dual linear programs.

$$\text{(6.6)}\quad \begin{array}{lll} \max & \sum_{i\in U, j\in H} v_{ij} z_{ij} & \\ \text{s.t.} & \sum_{T\in\mathcal{J}_j : i\in T} Y_j^T = z_{ij} & \forall i\in U, j\in H \\ & \sum_{T\in\mathcal{J}_j} Y_j^T \le 1 & \forall j\in H \\ & \sum_{j\in H} v_{ij} z_{ij} \le B_i & \forall i \in U \\ & Y_j^T \ge 0 & \forall j\in H, \forall T\in\mathcal{J}_j \\ & z_{ij}\ge 0 & \forall i\in U, j\in H \end{array}$$

$$\text{(6.7)}\quad \begin{array}{lll} \min & \sum_{i\in U} B_i q_i + \sum_{j\in H}\lambda_j & \\ \text{s.t.} & \lambda_j + \sum_{i\in T} y_{ij} \ge 0 & \forall j\in H, \forall T\in\mathcal{J}_j \\ & v_{ij} q_i - y_{ij} \ge v_{ij} & \forall j\in H, \forall i\in U \\ & \lambda_j \ge 0 & \forall j \in H \\ & q_i \ge 0 & \forall i\in U. \end{array}$$

Similar to solving linear program (2.1), and using a β-approximation polynomial-time algorithm for the rectangle packing problem for AdWord j, we can solve the primal linear program (6.6) within a factor β in polynomial time. Now, given a fractional solution, we round the solution as follows: for each AdWord j, we assign exactly one set of bidders to j and we assign a set T of bidders to j with probability Y_j^T. The following lemma proves that in the rounding process, we do not lose more than a factor $1 - \frac{1}{e}$.

LEMMA 6.1. *Let B'_i be the revenue of bidder i in the fractional solution and E_i be the expected revenue from bidder i in the rounded solution. Then $E_i \ge B'_i(1-\frac{1}{e})$.*

Proof. We know that $B'_i = \sum_{j\in H} v_{ij} z_{ij}$. Let X_{ij} be the indicator random variable that indicates if AdWord j is assigned to bidder i in the rounded solution. Therefore, $E_i = \mathbf{E}[min(B_i, \sum_{j\in H} X_{ij} v_{ij})]$. We know that $\mathbf{Pr}[X_{ij} = 1] = z_{ij}$. Therefore, similar to the proof of Theorem 4 in [2], we can show that $E_i \ge B'_i(1-\frac{1}{e})$.

The above lemma along with the $(\frac{1}{2}-\epsilon)$-approximation algorithm for the rectangle packing problem [20] give a 0.31-approximation algorithm for AAP. If the width of all rectangles are the same, then the packing problem for each AdWord is a knapsack problem, therefore, our algorithm is a 0.63-approximation algorithm for this special case of AAP.

6.3 Fractional DCP

For the fractional case, CapFBDC, we may consider $\mathcal{I}_i$ as the family of fractional subsets of requests that are feasible for cache location i and write similar linear programs, but the number of variables (or constrains) of these linear programs are infinite. Instead we modify the LP formulation. Let $\mathcal{Z}_i$ be the set of all vectors in $\{0,1\}^k$ such that for $Z^{(i)} \in \mathcal{Z}_i$, $\sum_{1\le t\le k} Z_t^{(i)} s_t \le A_i$. For a vector $Z^{(i)} \in \mathcal{Z}_i$, let $\mathcal{P}(Z^{(i)})$ be a polytope defined as follows: $\mathcal{P}(Z^{(i)}) = \{y^{(i)} \in R^m | \sum_{j\in H} y^{(i)} r_j \le$

[9] We can set δ' to be $\frac{1}{2^{cn}}$ for any constant $c > 0$.

$B_i, y_j^{(i)} \leq Z_{t_j}^{(i)}, 0 \leq y_j^{(i))} \leq 1\}$. Let $S(\mathcal{P}(Z^{(i)}))$ be the set of extreme points of polytope $\mathcal{P}(Z^{(i)})\}$ and let $\mathcal{S}_i = \cup_{Z^{(i)} \in \mathcal{Z}_i} S(\mathcal{P}(Z^{(i)}))$. For simplicity let $\mathcal{S}_i = \{y_1^{(i)}, \ldots, y_{p(i)}^{(i)}\}$ where $p(i) = |\mathcal{S}_i|$. Every point $y_i \in \mathcal{S}$ can be described as a linear combination of $\{y_l^{(i)} | 1 \leq l \leq p(i)\}$, thus $y_i = \sum_{l=1}^{p(i)} \lambda_l^{(i)} y_l^{(i)}$. Therefore, we can formalize the following linear program:

$$\begin{array}{lll} \text{maximize} \quad (6.8) & \sum_{i \in U} \sum_{l=1}^{p(i)} f_{ij} y_{l,j}^{(i)} \lambda_l^{(i)} & \\ \text{subject to} & \sum_{l=1}^{p(i)} \lambda_l^{(i)} = 1 & \forall i \in U \\ & \sum_{i \in U} \sum_{l=1}^{p(i)} y_{l,j}^{(i)} \lambda_l^{(i)} \leq 1 & \forall j \in H \\ & \lambda_l^{(i)} \geq 0 & \forall i \in U, 1 \leq l \leq p(i) \end{array}$$

The dual of the linear program 6.8 is as follows:

$$\begin{array}{lll} \text{minimize} \quad (6.9) & \sum_{i \in U} q_i + \sum_{j \in H} w_j & \\ \text{subject to} & q_i + \sum_{j \in H} y_{l,j}^{(i)} w_j \geq \sum_{j \in H} y_{l,j}^{(i)} f_{i,j} & \forall i \in U, 1 \leq l \leq p(i) \\ & w_j \geq 0 & \forall j \in H \end{array}$$

In order to find an approximate separation oracle, given a set of variables $(w_j, q_i | i \in U, j \in H)$, we need to check if there is a violated constraint in the linear program 6.9. Fixing a cache location i, we need to check if there exists $y_l^{(i)}$ for which $\sum_{j \in U} y_{l,j}^{(i)} (f_{ij} - w_j)$ is greater than q_i. By setting $v_j = f_{i,j} - w_j$ as the value of request j, we need to check if there exists a point $y^{(i)} \in \cup_{Z^{(i)} \in \mathcal{Z}_i} S(\mathcal{P}_i(Z^{(i)}))$ such that $\sum_{j \in H} v_j y^{(i)}$ is greater than q_i or not. Thus, the separation oracle is to maximize $\sum_{i \in U} y_j^{(i)} v_j$ for $y^{(i)} \in \cup_{Z^{(i)} \in \mathcal{Z}_i} S(\mathcal{P}_i(Z^{(i)}))$. We may relax this condition to $y^{(i)} \in \cup_{Z^{(i)} \in \mathcal{Z}_i} \mathcal{P}_i(Z^{(i)})$. This corresponds to the following 2-dimensional knapsack problem. We are given a bin with A_i available space and B_i available bandwidth. We are also given a set S of items with a value $v_j = f_{i,j} - w_j$, a type t_j, and a bandwidth consumption r_j. For each type t, we are given a size s_t. We want to pack a set of items with the most value. We may pack items fractionally. The bandwidth consumption of these items can be at most B_i and the sum of sizes of request types in this set is at most A_i. We can easily extend the PTAS for the separation oracle of CapIBDC to give a PTAS for the separation oracle for CapFBDC. This, in turn, gives an approximate solution to the primal linear program.

6.4 A PTAS for the single-bin subproblem for CapFBDC and CapIBDC A Polynomial-Time Approximation Scheme for the Separation Oracle.

Here, we develop a PTAS for the single-bin subproblem for CapFBDC and CapIBDC problems. Assume that the available storage and available bandwidth of cache location i are A_i and B_i. Each request j has a type t_j and the bandwidth consumption b_j. The size of type t is a_t and the value of request (item) i for cache location i is $v_j = f_{ij}$. The description of the PTAS with its analysis follows. The details of the analysis is left for the full version of the paper.

1. Let $L = \frac{1}{\epsilon}$.
2. If an optimal solution has $M \leq L$ types, then we can guess those M types. In the CapIBDC problem, given a set of types, the problem of finding a subset of requests with the maximum value is a knapsack problem and we can the FPTAS for this problem. In the CapFBDC problem, this problem is a simple maximum fractional knapsack problem that can be solved in polynomial time.
3. Scale all numbers v_j down to the nearest multiple of $\epsilon v_{max}/n$. Using the standard argument, any optimal solution for the new instance corresponds to a feasible $(1 - \varepsilon)$-approximate solution for the original problem.
4. If an optimal solution has more than L types in it, then we can guess L most profitable types, i.e., types such that requests of those types bring most profit to the objective. We also guess the total profit V_L achieved by the requests of those L types in the optimal solution. We can guess V_L exactly for the CapIBDC problem, since there is only a $\frac{n^2}{\epsilon}$ number of choices. For the CapFBDC problem, we guess the closest multiple of ϵ/n from below to the scaled profit achieved by the requests of L types in the optimal solution.
5. Solve the corresponding knapsack problem restricted to requests of those L fixed types, i.e., find the set of requests which achieve the guessed profit V_L using the smallest amount of bandwidth B_L. For the CapFBDC problem, this can be done by a simple greedy algorithm. For the CapIBDC problem, this can be done using a dynamic programming approach, since the number of possible values is a polynomial. Let B and A be the remaining bandwidth and capacity of the bin after packing requests of the above L most profitable types. We need to pack the remaining requests and types.
6. We write the straightforward linear program (6.10) with bandwidth B and capacity A. For the CapIBDC problem, we write this LP only for requests j such that $v_j \leq \frac{V_L}{L}$, since we only consider the types whose profit is less than the L types with the maximum profit. Also, we set $v'_j = v_j$ and $b'_j = b_j$ for all j such that $v_j \leq \frac{V_L}{L}$. For the CapFBDC problem, we set $v'_j = v_j$ and $b'_j = b_j$ if $v_j \leq \frac{V_L}{L}$ and $v'_j = \frac{V_L}{L}$ if $v_j > \frac{V_L}{L}$ and set $b'_j = \frac{v'_j b_j}{v_j}$. This implies that if the value of a request is greater than $\frac{V_L}{L}$, then we use at most a fraction $\frac{V_L}{L}$

of this request. We say that a request j is tight if either $k_j = 0$ or $k_j = p_{t_j}$, otherwise we say that the request j is loose. WLOG, in an optimal basic feasible solution (p^*, k^*) of this LP, for each type t, there are at most two requests that are not not tight, i.e, two requests j such that $p^*_{t_j} > k^*_j$ and $k^*_j > 0$. The reason is that there are two constraints on variables k_j of a a type t.

$$
\begin{array}{lll}
\text{maximize} & \sum_{j \in S} v'_j k_j & \\
\text{subject to} & \sum_{j \in S} k_j b'_j \le B & \\
 & p_{t_j} \ge k_j & \forall j \in S \\
 & \sum_{1 \le t \le k} p_t a_t \le A & \\
 & \sum_{j: t_j = t} v'_j k_j \le \frac{V_L}{L} & \forall 1 \le t \le k \\
 & 0 \le p_t \le 1 & \forall 1 \le t \le k \\
 & 0 \le k_j \le 1 & \forall j \in S
\end{array}
$$

(6.10)

7. Change the above solution (p^*, k^*) to linear program (6.10) as follows: For any two requests j and j' (not necessarily of the same type) that are not tight, do simple transformation add (or subtract) ϵ from one k^*_j and subtract (or add) value linearly dependent on ϵ from the second variable $k^*_{j'}$ such that the total bandwidth is the same and change ϵ in the direction of the increase of objective function until either one of $k_j = p_{t_j}$ or $k_j = 0$ or $k_{j'} = p_{t_{j'}}$ or $k_{j'} = 0$. After doing this process on all pairs of loose requests j and j', we get a vector (p, k) which has at most one loose request j. This solution may not be feasible for linear program (6.10), but we claim that is feasible for linear program (6.11). The reason is that for each type t, there are at most two loose requests in the solution (p^*, k^*). For each loose request j, $v'_j \le \frac{V_L}{L}$. Hence, for each type t the increase in the summation $\sum_{j: t_j = t} v'_j k_j$ is at most $\frac{2V_L}{L}$. As a result, in the resulting solution after this process, $\sum_{j: t_j = t} v'_j k_j \le \frac{3V_L}{L}$. Therefore, this solution (p, k) is feasible for linear program (6.11) and has at most one loose variable j.

$$
\begin{array}{lll}
\text{maximize} & \sum_{j \in S} v'_j k_j & \\
\text{subject to} & \sum_{j \in S} k_j b'_j \le B & \\
 & p_{t_j} \ge k_j & \forall j \in S \\
 & \sum_{1 \le t \le k} p_t a_t \le A & \\
 & \sum_{j: t_j = t} v'_j k_j \le \frac{3V_L}{L} & \forall 1 \le t \le k \\
 & 0 \le p_t \le 1 & \forall 1 \le t \le k \\
 & 0 \le k_j \le 1 & \forall j \in S
\end{array}
$$

(6.11)

8. Throw away the loose request j from the above linear program. By throwing away this request, we decreased the objective function by a small amount, since we have L types which have profit V_L and the profit of this request is at most $\frac{V_L}{L}$.

9. We now have a feasible solution of linear program (6.11) without any loose request j and with total value at least $1 - \epsilon$ times the optimal value of linear program (6.10). Since $k_j = p_{t_j}$ for any $k_j > 0$, we can throw away all requests j such that $k_j = 0$ and write the remaining problem as a 2-dimentional knapsack problem on the set of types. The reason is that by substituting p_{t_j} instead of k_j both capacity and bandwidth constraints are over types. Moreover, the constraint $\sum_{j: t_j = t} v'_j k_j \le \frac{3V_L}{L}$ is implied from the constraint $p_t \le 1$. Thus, we can remove this constraint.

10. Find a basic optimal solution of the resulting two-dimentional knapsack linear program. This solution has at most two fractional variables for types, since there are only two constraints. (Note that we removed the trivial constraints $\sum_{j: t_j = t} v'_j k_j \le \frac{3V_L}{L}$.) Throw away all variables associated with these two types. By throwing away these two requests, we decreased the objective function by at most $\frac{6V_L}{L}$ which is a small amount compared to V_L that we have already packed.

From the above algorithm and analysis, we can prove that for any constant $\epsilon > 0$, the above algorithm is a $1 - 10\epsilon$-approximation algorithm for the single-bin subproblem. Thus, there exists a PTAS for the single-bin subproblem of CapIBDC and CapFBDC.

Proof. **Proof of Lemma 3.1**

Let R be the set of items that are better served in the optimal solution than in $\mathcal{S}$ and L be the rest of the items. Let R_i be the set of items in R served by bin i in the optimum Ω. For any set T of items, let $o(T)$ be the value of the items in T in Ω and $l(T)$ be the value of items of T in $\mathcal{S}$. For all items $j \in R_i$, the marginal value for bin i is positive ($w_j > 0$), since the value of j for bin i is greater than the current value of j in $\mathcal{S}$. For each item $j \in R_i$, the marginal value of j for solution $\mathcal{S}$ is $f_{ij} - \mathsf{value}_j(\mathcal{S}) \ge f_{ij} - v_j(\mathcal{S})$. Thus, the total marginal value of items in set R_i for bin i is at least $\sum_{j \in R_i}(f_{ij} - v_j(\mathcal{S})) = o(R_i) - l(R_i)$. Since $\mathcal{S}$ is a β-approximate local solution, the operation $\mathsf{Local}(i)$ cannot find a solution with marginal value greater than $\alpha_i(\mathcal{S})$, otherwise this operation could increase the total value. Since we use a β-approximation algorithm for the single-bin subproblem and there exists a solution with marginal value $o(R_i) - l(R_i)$, it follows that $\alpha_i(\mathcal{S}) \ge \beta(o(R_i) - l(R_i))$. Therefore, $o(R_i) \le l(R_i) + \frac{1}{\beta}\alpha_i(\mathcal{S})$. Furthermore, for items in set L, $o(L) \le l(L) \le v(\mathcal{S})$ by definition. Therefore,

$$\begin{aligned}\mathrm{OPT} &= v(\Omega) \\ &= o(L) + o(R) \\ &= o(L) + \sum_{i \in U} o(R_i) \\ &\le l(L) + \sum_{i \in U} (l(R_i) + \frac{1}{\beta}\alpha_i(\mathcal{S})) \\ &\le (1 + \frac{1}{\beta}) v(\mathcal{S}).\end{aligned}$$

Thus, $v(\mathcal{S}) \geq \frac{\beta}{\beta+1} v(\Omega)$.

6.5 k-median with hard capacities We can extend the local search algorithm for SAP to the k-median problem with hard capacities and packing constraints(KMed). Recall that KMed is as follows: Given a set U of n bins, a set H of m items with a value f_{ij} for each item j and each bin i, and also a single-bin subproblem for each bin i, i.e., a lower-ideal family $\mathcal{I}_i$ of subsets for bin i, choose at most K bins and pack a set of items in each selected bin to maximize the total value packed. To the best of our knowledge, this is the first constant-factor approximation algorithm for the the k-median problem with hard capacity constraints.

The local search algorithm for the KMed problem is very similar to the local search algorithm for SAP. At each step of the algorithm, we try to unpack a used bin, and pack a (possibly different) bin to increase the total value. The formal description of the algorithm is as follows:

Local Search Algorithm for KMed.

1. Start with the empty solution, i.e., $\mathcal{S} = (S_1, \ldots, S_n)$ and $S_i = \emptyset$ for all $i \in U$.

2. Let $P = \{1, 2, \ldots, K\}$.

3. For an appropriate ϵ', run the following loop for $\frac{1}{\beta} K \ln(\frac{1}{\epsilon'})$ times:

   (a) Let the current assignment be $\mathcal{S} = (S_1, \ldots, S_n)$ where $S_i = \emptyset$ for $i \notin P$.

   (b) For each bin $i_1 \in P$ and bin $i_2 \in (U - P) \cup \{i_1\}$ do

      i. For each item j, let $\mathsf{value}_j(\mathcal{S})$ be $f_{i'j}$ if j is assigned to a bin $i' \neq i_1$ in $\mathcal{S}$, and be equal to zero if j is unassigned or is assigned to bin i_1 in $\mathcal{S}$.

      ii. For each item j, let the *marginal value* of j (with respect to bins i_1 and i_2) be $w_j = f_{i_2 j} - \mathsf{value}_j(\mathcal{S})$.

      iii. Use the β-approximation algorithm for the single-bin subproblem for bin i_2 to pack a subset of items in bin i_2 with the maximum *marginal value*. Let the marginal value of this solution for bin i_2 be $W_{i_1 i_2}$ and let $S'_{i_1 i_2}$ be the set of items with marginal value $W_{i_1 i_2}$.

   (c) For every two bins $i_1 \in P$ and $i_2 \in (U - P) \cup \{i_1\}$, let $\Delta_{i_1 i_2} = W_{i_1 i_2} - \alpha_{i_1}(\mathcal{S})$.

   (d) Let bins i_1^* and i_2^* be the bins with the maximum $\Delta_{i_1 i_2}$, i.e., $\Delta_{i_1^* i_2^*} \geq \Delta_{i_1 i_2}$ for any bin i_1 and i_2.

   (e) If $\Delta_{i_1^* i_2^*} > 0$, unpack bin i_1^* and pack the set $S'_{i_1^* i_2^*}$ of items in bin i_2^* (i.e., set $P = P - \{i_1^*\} \cup \{i_2^*\}$ and set $S_{i_1^*} = \emptyset$ and $S_{i_2^*} = S'_{i_1^* i_2^*}$).

THEOREM 6.1. *For any $\epsilon > 0$, the above local search algorithm for* KMed *is a polynomial-time $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm for* SAP.

Proof. The output of the local search algorithm is a feasible solution of the KMed problem, since the number of nonempty bins in the output is at most K. Let Ω be an optimal assignment. Consider the current assignment $\mathcal{S}$ in the process of the algorithm. Let P_Ω and $P_\mathcal{S}$ be the set of used bins in Ω and $\mathcal{S}$ respectively. The set of used bins in Ω and $\mathcal{S}$ are not necessarily the same. Let $\pi : U \to U$ be a permutation function such that if $i \in U$ is used in both Ω and $\mathcal{S}$, then $\pi(i) = i$ and if i is used in Ω, but not in $\mathcal{S}$ then $\pi(i) = i'$ where i' is used in $\mathcal{S}$, but not in Ω. As the number of used bins in Ω is K, such a permutation exists. Let R be the set of items that are better served in Ω than in $\mathcal{S}$ and L be the rest of the items. Let R_i be the set of items in R satisfied by bin i in Ω. For any set T of items, let $o(T)$ be the value of the items in T in Ω and $l(T)$ be the value of items of T in $\mathcal{S}$. Thus, we have $\mathrm{OPT} = o(R) + o(L)$ and $v(\mathcal{S}) = l(R) + l(L)$. For an item j, let $v_j(\mathcal{S})$ the value of item j in $\mathcal{S}$. Consider two bins $\pi(i)$ and $i \in P_\Omega$. For each item $j \in R_i$, the marginal value of j with respect to bins $\pi(i)$ and i is $f_{ij} - \mathsf{value}_j(\mathcal{S}) \geq f_{ij} - v_j(\mathcal{S})$. Thus, the marginal value of set R_i with respect to bins $\pi(i)$ and i is at least $\sum_{i \in R_i} (f_{ij} - v_j(\mathcal{S})) = o(R_i) - l(R_i)$ and R_i is a feasible solution for bin i. Since we use a β-approximation algorithm to find $W_{\pi(i)i}$, $W_{\pi(i)i} \geq \beta(o(R_i) - l(R_i))$. Therefore, $\sum_{i \in P_\Omega} W_{\pi(i)i} \geq \beta \sum_{i \in P_\Omega} (o(R_i) - l(R_i)) = \beta(o(R) - l(R))$. Since $o(L) \leq l(L)$, $\sum_{i \in P_\Omega} W_{\pi(i)i} \geq \beta(o(R) - l(R) + o(L) - l(L)) = \beta(\mathrm{OPT} - v(\mathcal{S}))$. Thus, $\sum_{i \in P_\Omega} \Delta_{\pi(i)i} = \sum_{i \in P_\Omega} W_{\pi(i)i} - \sum_{i \in P_\mathcal{S}} \alpha_i(S) \geq \beta(\mathrm{OPT} - v(\mathcal{S})) - v(\mathcal{S}) = \beta\mathrm{OPT} - (1+\beta)v(\mathcal{S})$. In particular, $\Delta_{i_1^* i_2^*} \geq \frac{\beta}{K}\mathrm{OPT} - \frac{1+\beta}{K} v(\mathcal{S})$. Let $\mathcal{S}'$ be the assignment after setting $S_{i_1^*}$ to $\emptyset$ and changing the set of items of i_2^* to $S'_{i_1^* i_2^*}$. As a result,

$$\begin{aligned}v(\mathcal{S}') &= v(\mathcal{S}) + \Delta_{i_1^* i_2^*} \\ &\geq v(\mathcal{S}) + \frac{\beta}{K}\mathrm{OPT} - \frac{1+\beta}{K} v(\mathcal{S}) \\ &= v(\mathcal{S})(1 - \frac{1+\beta}{K}) + \frac{\beta}{K}\mathrm{OPT}.\end{aligned}$$

Now, let y_k be the total value of the assignment after the k'th execution of Loop 3b. From above discussion, $y_k \geq (1-\frac{1+\beta}{K})y_{k-1} + \frac{\beta}{K}\mathrm{OPT}$ and $y_0 = 0$. Similar to the proof of Theorem 3.1 and solving this recurrence relation, we get $y_k \geq \frac{\beta}{1+\beta}(1-(1-\frac{\beta+1}{K})^k)\mathrm{OPT}$. By setting $k = \frac{K\ln(\frac{1}{\epsilon'})}{\beta+1}$, we get $y_k \geq \frac{\beta}{1+\beta}(1-\frac{1}{e^{\ln(\frac{1}{\epsilon'})}})\mathrm{OPT} = \frac{\beta}{1+\beta}(1-\epsilon')\mathrm{OPT}$. Therefore, the value of the output of the above algorithm is at least $(\frac{\beta}{1+\beta}-\epsilon)\mathrm{OPT}$ as desired.

6.6 Decentralized Mechanisms In this section, we explore methods to obtain decentralized algorithms for all variants of the DCP problem with a good performance. Before stating the formal definitions and results, we give an introduction on the applicability of the decentralized mechanisms in distributed caching in cellular networks.

The 3G subscriber market can be categorized into groups with shared interest in location-based services, e.g., the preview of movies in a theater or the scene of the beach nearby. Since the 3G radio resources are limited, it is expensive to repeatedly transmit large quantities of data over the air interface from the base station (BS). It is more economical for the service provider to offload such repeated requests on to the ad-hoc network comprised of its subscribers where some of them recently acquired a copy of the data. In this scenario the goal for the service provider is to give incentives for peer subscribers in the system to cache and forward the data to the requesting subscribers. Since each data item is large in size and transit subscribers are mobile, we assume that the data transfer occurs in a close range of a few hops.

In this system, we envision a system consisting of two groups of subscribers: resident and transit subscribers. Resident subscribers are less mobile and mostly confined to a certain geographical area. Resident subscribers have incentives to cache data items that are specific to this geographical region since the service provider gives monetary rewards for satisfying the queries of transit subscribers. Transit subscribers request their favorite data items when they visit a particular region. Since the service provider does not have knowledge of the spatial and temporal distribution of requests, it is difficult if not impossible for the provider to stipulate which subscriber should cache which set of data items. Therefore, the decision of what to cache is left to each individual subscriber. The realization of this content distribution system depends on two main issues. First, since subscribers are selfish agents, they may act to increase their individual payoff and decrease the performance of the system. Here, we provide a framework for which we can prove that in an equilibrium situation of this framework, we use the performance of the system efficiently (below, we will describe the efficiency formally in terms of the price of anarchy of a game). The second issue is that the payoff of each request for each agent must be a function the set of agents that have this request in their strategy, since these agents compete on this request and the profit of this request should be divided among these agents in an appropriate way. Therefore, each selfish agent may change the set of items it cached in response to the set of items cached by others. This leads to a non-cooperative caching scenario which we model as the distributed caching game. Motivated from service provider cellular networks, we assume that cache locations are selfish agents (resident subscribers) who want to maximize their own profit.

6.7 Preliminaries Here, we define some game theoretic notions that are necessary for defining the distributed caching game.

A *strategic game* $\mathcal{G}$ is defined as a tuple $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha_i() | i \in U\})$ where (i) U is the set of n players or agents, (ii) F_i is a family of feasible *(pure) strategies* or *actions* for player i and (iii) $\alpha_i : \Pi_{i\in U} F_i \to \mathbb{R}^+ \cup \{0\}$ is the (private) *payoff* or *utility* function for agent i, given the set of strategies of all players. Player i's strategy is denoted by $s_i \in F_i$. A *strategy profile* or a (strategy) *state*, denoted by $S = (s_1, s_2, \ldots, s_n)$, is a vector of strategies of players. $\emptyset_i$ corresponds to a null or empty strategy for player i. Also let $S \oplus s'_i := (s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_k)$, i.e., the strategy profile obtained from S if agent i changes its strategy from s_i to s'_i. The vector of strategies of players except player i is denoted by by $S_{-i} := (s_1, \ldots, s_{i-1}, \cdot, s_{i+1}, \ldots, s_n)$. Also let $\mathcal{F} := \Pi_{i\in U} F_i$ be the set of all possible strategy profiles.

In a *non-cooperative* game, it is assumed that each agent wishes to maximize its own payoff. In other words, agents are selfish and given the strategy profile for other players, they play a strategy that maximizes their own utility function. For a strategy profile $S = (s_1, s_2, \ldots, s_n)$, an *improvement move* of player i is a strategy s'_i such that $\alpha_i(S \oplus s'_i) \geq \alpha_i(S)$. For a strategy profile $S = (s_1, s_2, \ldots, s_n)$, a *strict improvement move* of player i is a strategy s'_i such that $\alpha_i(S \oplus s'_i) > \alpha_i(S)$. Also, for a strategy profile $S = (s_1, s_2, \ldots, s_n)$ a *best response* of player i in S is a strategy $s^*_i \in F_i$ such that for any strategy $s_i \in F_i$, $\alpha_i(S \oplus s^*_i) \geq \alpha_i(S \oplus s_i)$. Note that a best response is an improvement move.

A pure strategy Nash equilibrium (PSNE) of a strategic game is a strategy profile in which each player plays his best response. More formally, $S = (s_1, s_2, \ldots, s_n)$ is a pure strategy Nash equilibrium (PSNE) if for all $i \in U$ and for any strategy $s'_i \in F_i$, $\alpha_i(S) \geq \alpha_i(S \oplus s'_i)$.

A *mixed (randomized) strategy* for player i is a probability distribution $\pi_i : F_i \to \mathbb{R}^+ \cup \{0\}$ over pure strategies where $\sum_{s_i\in F_i} \pi_i(s_i) = 1$. A *mixed strategy profile*, denoted by $P = (\pi_1, \pi_2, \ldots, \pi_n)$, is a vector of mixed strategies of all players. For a mixed strategy profile P the probability of realizing a pure strategy profile $S = (s_1, \ldots, s_n)$ is $p^P(s_1, \ldots, s_n) := \Pi_{i\in U} \pi_i(s_i)$. The *expected payoff* of player i in the mixed strategy profile P is $\beta_i(P) := \sum_{s_1 \in F_1, \ldots, s_n \in F_n} p^P(s_1, \ldots, s_n)\alpha_i(s_1, s_2, \ldots, s_n)$. Similar to the above definition of pure Nash equilibrium, we can define a mixed Nash equilibrium. A mixed strategy Nash equilibrium is a mixed strategy profile in which each player does not have an incentive to play a different mixed strategy and improve her expected payoff. Nash [25] proved that any non-cooperative strategic game possesses a mixed Nash equilibrium. We denote Nash equilibrium by NE and the pure strategy Nash equilibrium by PSNE.

In order to model the selfish behavior of players, we use the underlying *state graph*. Each vertex in the state graph represents a strategy profile or a state $S = (s_1, s_2, \ldots, s_n)$. The arcs in the state graph correspond to best-response moves by the players. Formally, the state graph, $\mathcal{D} = (\mathcal{F}, \mathcal{E})$, of a strategic game $\mathcal{G}$, is an arc-labeled directed graph, where the vertex set $\mathcal{F}$ corresponds to the set of strategy profiles or states in $\mathcal{G}$, and there is an arc from state S to state S' with label i if the only difference between S and S' is in the strategy of player i; and player i plays his best response or one of his best responses in strategy profile S'[10].

Observe that the state graph will contain loops. A *best-response walk* is a directed walk in the state graph. We say that a player i plays in the best-response walk $\mathcal{P}$, if at least one of the edges of $\mathcal{P}$ is labeled i.

The major tool for analyzing the lack of coordination for some games is the notion of the *price of anarchy* in a game [22, 27]. Given a strategic game, $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha() | i \in U\})$, and a maximization social function $\gamma : \Pi_{i \in U} F_i \to \mathbb{R}$, the price of anarchy, denoted by $\mathsf{poa}(\mathcal{G}, \gamma)$, is the worst ratio between the optimum and the social value of a pure Nash equilibrium [27][11]. Formally, if $\mathcal{N}$ is the set of all pure Nash equilibria, then $\mathsf{poa}(\mathcal{G}, \gamma) := \frac{\mathrm{OPT}}{\min_{S \in \mathcal{N}} \gamma(S)}$. Similarly, if $\mathcal{P}$ is the set of all mixed Nash equilibria, the price of anarchy for mixed Nash equilibria is equal to $\frac{\mathrm{OPT}}{\min_{P \in \mathcal{P}} \gamma(P)}$. Also, the price of anarchy for a minimization social function γ is defined as $\frac{\max_{P \in \mathcal{P}} \gamma(P)}{\mathrm{OPT}}$.

Now, we are ready to define the distributed caching games. Consider a distributed caching setting in which selfish cache locations decide which types and which requests to provide, based on their limited space and bandwidth. The service provider let cache locations decide on the set of requests they want to satisfy, and run the following rewarding scheme: the profit of each request will go to the cache location that provides this request with the maximum profit. The *payoff* of a cache location is the total profit of the requests that are assigned to this cache location. Cache locations compete with each other on getting more profit from satisfying these requests. This defines a game - the $\mathsf{CapIBDC}$ game which is formally defined below. The service provider is interested in maximizing the total profit, i.e., the social function is the total profit of the cache locations.

Formally, we define the *distributed caching game* in the setting of $\mathsf{CapIBDC}$ - with both capacity and bandwidth constraints. Given an instance of the $\mathsf{CapIBDC}$ problem, we define a strategic game $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha_i | i \in U\})$ as follows. The set of players U is the set of cache locations. The family of feasible strategies F_i of a cache location i is the family of subsets s_i of requests such that $\sum_{j \in s_i} b_j \leq B_i$ and $\sum_{t \in \{t_j | j \in s_i\}} a_t \leq A_i$. Given a vector $S = (s_1, s_2, \ldots, s_n)$ of strategies of cache locations, the *favorite* cache locations for request j, denoted by $\mathsf{FAV}(j)$, is the set of cache locations i such that $j \in s_i$ and f_{ij} has the maximum profit among the cache locations that have request j in their strategy set, i.e., $f_{ij} \geq f_{i'j}$ for any i' such that $j \in s_{i'}$. For a strategy profile $S = (s_1, \ldots, s_n)$ $\alpha_i(S) = \sum_{j : i \in \mathsf{FAV}(j)} \frac{f_{ij}}{|\mathsf{FAV}(j)|}$. Intuitively, the above definition implies that the profit of each request goes to the cache locations with the minimum connection cost (or equivalently with the maximum profit) among the set of cache locations that provide this request. If more than one cache location have the maximum profit (or minimum connection cost) for a request j, the profit of this request is divided equally between these cache locations. The payoff of a cache location is the sum of profits from the requests it actually serves. We say that a player i serves a request j if $i \in \mathsf{FAV}(j)$. The social value of strategy profile S, denoted by $\gamma(S)$, is the sum of profits of all players. This value $\gamma(S)$ is a measure of the efficiency of the assignment of requests and request types to cache locations.

We similarly define games for the other versions of the distributed caching problem. For problems, such as CapDC and uniform CapDC, that just have capacity constraints, the strategy of player i is simply a subset of requests s_i and a strategy s_i is a feasible strategy for player i if $\sum_{t \in \{t_j | j \in s_i\}} a_t \leq A_i$. For IBDC, the strategy of player i is a subset of requests s_i and the set s_i is a feasible strategy for player i if $\sum_{j \in s_i} b_j \leq B_i$.

In this section, we bound the price of anarchy for these games (Section 6.8), study the existence of pure equilibria (Section 6.9) and demonstrate convergence results for the more general versions of the game (Section 6.10). Picture **??** depicts the games that we consider in this paper and other games and their relation to each other. Some of these games are defined in [15, 23].

[10] In the definition of the best-response moves and the state graph, we only consider the myopic best responses. Other models can be considering nonmyopic players who can predict the responses of other players and play a strategy that gives more payoff in the long-term and not only at the current strategy profile.

[11] In this paper, we use the term, price of anarchy, for pure Nash equilibria. For mixed Nash equilibria, we use the term, price of anarchy for mixed Nash equilibria. In some other places, the price of anarchy is defined for mixed Nash equilibria.

6.8 CapIBDC **Game: Price of Anarchy** Since the CapIBDC game is a strategic game, it has mixed Nash equilibria [25]. We prove that in a mixed Nash equilibrium of this game, the expected social value is at least $\frac{1}{2}$ of the optimal social value. We can prove this by showing that the CapIBDC game is a valid-utility game. First, we give a direct proof of this fact.

THEOREM 6.2. *The price of anarchy of the* CapIBDC *game for a mixed Nash equilibrium is at most* 2.

Proof. In a mixed Nash equilibrium, each player chooses a probability distribution over its feasible pure strategies. We prove that the expected social value of this equilibrium is at least $\frac{1}{2}$ of the optimum solution. Consider the optimum solution $\Omega = (\sigma_1, \ldots, \sigma_n)$. Let the profit of request j in Ω be ρ_j. Consider a mixed Nash equilibrium $\mathcal{P}$. Let the expected payoff of player i in $\mathcal{P}$ be $\alpha_i(\mathcal{P})$. For any request j, let p_j be the probability that j's profit in $\mathcal{P}$ is at least ρ_j. It is easy to see that $\mathbf{E}[\gamma(\mathcal{P})] \geq \sum_{j\in H} p_j\rho_j$, since with probability p_j the profit of j is at least ρ_j. We know that the expected social value of $\mathcal{P}$ is equal to $\mathbf{E}[\gamma(\mathcal{P})] = \sum_{i\in U}\alpha_i(\mathcal{P})$. Let player i change his strategy in $\mathcal{P}$ to the pure strategy σ_i. Consider a request $j \in \sigma_i$. With probability $1 - p_j$ the profit of request j is less than ρ_j. When player i provides this request, the profit is ρ_j. Thus, if player i changes his strategy to σ_i, his expected payoff is at least $\sum_{j\in\sigma_i}(1 - p_j)\rho_j$. $\mathcal{P}$ is a mixed Nash equilibrium, thus player i's expected payoff in $\mathcal{P}$ is at least his expected payoff when he plays σ_i in $\mathcal{P}$. Therefore, $\alpha_i(\mathcal{P}) \geq \sum_{j\in\sigma_i}(1 - p_j)\rho_j$. Therefore,

$$
\begin{aligned}
\text{OPT} &= \gamma(\Omega) \\
&= \sum_{j\in H}\rho_j \\
&= \sum_{j\in H} p_j\rho_j + \sum_{j\in H}(1-p_j)\rho_j \\
&\leq \mathbf{E}[\gamma(\mathcal{P})] + \sum_{i\in U}\sum_{j\in\sigma_i}(1-p_j)\rho_j \\
&\leq \mathbf{E}[\gamma(\mathcal{P})] + \sum_{i\in U}\alpha_i(\mathcal{P}) \\
&= 2\mathbf{E}[\gamma(\mathcal{P})]
\end{aligned}
$$

as desired.

In the following, we show that the CapIBDC game is a valid-utility game. By a result of Vetta [38], this gives an alternative proof for the price of anarchy for mixed Nash equilibria in CapIBDC game.

First, we define some preliminaries. A function f of the form $2^V \to \mathbb{R}^+ \cup \{0\}$ is called a set function on the ground set V. A set function $f : 2^V \to \mathbb{R}^+ \cup \{0\}$ is submodular if for any two sets $A, B \subseteq V$, $f(A) + f(B) \geq f(A\cap B) + f(A\cup B)$. A set function f, is non-decreasing if $f(X) \leq f(Y)$ for any $X \subseteq Y \subseteq V$.

Here, we describe an abstract definition of valid-utility games. In these games, for each player i, there exists a ground set V_i. Note that V_i's may intersect. We denote by V the union of ground sets of all players, i.e., $V = \cup_{i\in U}V_i$[12]. The feasible strategy set F_i of player i is a subset of the power set of V_i, 2^{V_i} that contains the empty set $\emptyset$ which corresponds to having no action. Thus, the strategy s_i of player i is a subset of V_i ($s_i \subseteq V_i$). In order to define these games, we need the following definitions.
Given a vector $S = (s_1, \ldots, s_n)$, where s_i is a subset of the ground set V_i ($s_i \subseteq V_i$), the set $\mathcal{H}_S = \{(i, j) : i \in U, j \in s_i\}$ is called *the pair set* for vector S. Note that S may or may not be a feasible strategy profile. Given a function $f : \Pi_{i\in U}2^V \to \mathbb{R}^+ \cup \{0\}$, the *corresponding set function* f^s of f is a set function of the form $2^{\mathcal{H}} \to \mathbb{R}^+ \cup \{0\}$ where $\mathcal{H} = \{(i, j) : i \in U, j \in V\}$ and $f^s(\mathcal{H}_S) = f(S)$. In other words, for a set $A \subseteq \mathcal{H}$, $f^s(A) = f((a_1, a_2, \ldots, a_n))$ if $a_i = \{j : (i, j) \in A\}$. Here, we assume that the social function γ is of the form $\Pi_{i\in U}2^V \to \mathbb{R}^+ \cup \{0\}$ instead of the form $\Pi_{i\in U}F_i \to \mathbb{R}^+ \cup \{0\}$.

Let $\mathcal{G}(U, \{F_i|i \in U\}, \{\alpha_i()|i \in U\})$ be a non-cooperative strategic game where $F_i \subseteq 2^{V_i}$ is a family of feasible strategies for player i. Let $V = \cup_{i\in U}V_i$ and let the social function be $\gamma : \Pi_{i\in U}2^V \to \mathbb{R}^+ \cup \{0\}$. $\mathcal{G}$ is a *valid-utility game* if it satisfies the following properties:

1) Submodular and Non-decreasing Social Function: γ^s, the corresponding set function of γ over the set $\mathcal{H} = \{(i, j) : i \in U, j \in V\}$, is submodular and non-decreasing.

2) Vickrey Condition: The payoff of a player is at least the difference in the social function when the player participates versus when it does not participate, i.e., $\alpha_i(S) \geq \gamma'_{s_i}(S \oplus \emptyset_i)$. In *basic-utility* games, this is an equality, i.e., $\alpha_i(S) = \gamma'_{s_i}(S \oplus \emptyset_i)$.

3) Cake Condition: For any strategy profile, the sum of the payoffs of players should be less than or equal to the social function for that strategy profile, i.e., for any strategy profile S, $\sum_{i\in U}\alpha_i(S) \leq \gamma(S)$.

Vetta [38] proved that the price of anarchy for mixed Nash equilibria in valid-utility games is at most 2. This framework encompasses a wide range of games including the facility location games, the traffic routing games, auctions [38], market sharing games [15]. Here, we prove that this set of games include the distributed caching games.

THEOREM 6.3. *The* CapIBDC *game is a valid-utility game.*

[12] We can define the valid-utility games by setting all ground sets $V_i = V$. But for later convenience, we let the ground set of player i be V_i.

Proof. We need to show the following three properties:

1) Nondecreasing and Submodular Social Function: First, it is clear that γ^s is non-decreasing. To show its submodularity, we use an equivalent definition of submodular functions: A set function f is submodular if for any two subsets A and B such that $A \subset B$ and for any element $i \notin B$, $f(A \cup \{i\}) - f(A) \geq f(B \cup \{i\}) - f(B)$ [12]. Thus, in order to prove that γ^s is submodular, it is enough to prove that for two (possibly infeasible) strategy profiles $S = (s_1, \ldots, s_n)$ and $S' = (s'_1, \ldots, s'_n)$ such that $s_i \subseteq s'_i$ for all $i \in U$, by adding a new request j to the strategy set of any player i the increase in γ^s for S is not less than the increase for S'. Let v_j and v'_j be the profit of request j in S and S', respectively ($v_j = 0$ if $j \notin \cup_{i \in U} s_i$ and $v'_j = 0$ if $j \notin \cup_{i \in U} s'_i$). As a result,

$$\begin{aligned} v'_j &= \max_{i: j \in s'_i} f_{ij} \\ &\geq \max_{i: j \in s_i} f_{ij} \\ &= v_j. \end{aligned}$$

Adding j to a strategy s_h and s'_h increases γ^s for S and S' by $\max(0, f_{hj} - v_j)$ and $\max(0, f_{hj} - v'_j)$, respectively. From $v'_j \geq v_j$, it follows that $\max(0, f_{h_j} - v_j) \geq \max(0, f_{hj} - v'_j)$. Hence, the increase in γ^s for S is greater than or equal to the increase for S'. Thus in any case, the increase for S is not less than the increase for S'.

2) Vickrey Condition: The difference in the social function when i plays s_i or empty (does not play at all) is equal to $\sum_{j: i \in \mathsf{FAV}(j), |\mathsf{FAV}(j)|=1} f_{ij}$ and this is indeed less than or equal to $\alpha_i(S) = \sum_{j: i \in \mathsf{FAV}(j)} \frac{f_{ij}}{|\mathsf{FAV}(j)|}$.

3) Cake Condition: By the definition of the social function, we have $\sum_{i \in U} \alpha_i(S) = \gamma(S)$ and therefore the third property is satisfied as well.

The above theorems extend to the general setting for all separable assignment problems. We can define the SAP game similar to the CapIBDC game: bins are selfish agents and their strategy is to keep a feasible subset of items according to the lower-ideal $\mathcal{I}_i$. The profit of an item j goes to a player i that have this item in his strategy and has the maximum profit for item j among all bins that have this item in their strategy. The above proofs show that the SAP game is a valid-utility game and thus, the price of anarchy for a mixed NE for this game is at most 2.

6.9 DCP **Games: Pure Nash Equilibria** In this section, we address various problems related to pure strategies in the DCP games. We show that on the one hand, there are instances of CapDC and IBDC that have no pure Nash equilibria; while the uniform IBDC games in which there are no ties among the profit of the requests to cache locations have pure Nash equilibria. For the uniform CapDC game, we demonstrate a cycle of strict best responses. A *strict best-response move* for a player i is a best-response move in which player i strictly increases his payoff.

THEOREM 6.4. *There are instances of the* IBDC *game that have no pure Nash equilibrium.*

Proof. Consider the following instance with 2 players and 4 requests. The bandwidth consumption of requests are 8, 3, 5 and 6 respectively. The profit of requests to player 1 are 10, 2, 5.5 and 5.5 respectively. The profit of requests to player 2 are 2, 3, 3 and 5.6 respectively. The available bandwidth of player 1 and 2 are 11 and 8 respectively. The example is depicted in Table 1. The only possible best responses of player 1 to any strategy of player 2 are sets $\{1, 2\}$ and $\{3, 4\}$. The only possible best responses of player 2 are subsets $\{4\}$ and $\{2, 3\}$. None of the 4 pairs of best responses is a pure Nash equilibrium, thus this game does not have a pure Nash equilibrium.

Since, IBDC is a special case of CapDC, the above theorem implies that there are instances of the CapDC game that have no pure Nash equilibrium. In the above example the bandwidth consumption of requests are not uniform, and this was essential in finding the example. In the following, we study the uniform variant of these games. Note that we can easily change the above example to an example of the IBDC game without tie among the profit of the requests and with no PSNE. In the following, we prove that the uniform IBDC game in which there are no ties among the profit of requests does not contain any cycle of strict best-response moves. As a result, the subgraph of the state graph with strict best-response moves as arcs does not contain any cycle. Therefore, this graph has a vertex without any outgoing arcs, and this vertex is a PSNE.

THEOREM 6.5. *Any instance of the uniform* IBDC *game in which there are no ties among the profit of the requests does not have any cycle of strict best-response moves and thus, has a PSNE.*

Proof. Given a strategy profile $S = (s_1, \ldots, s_n)$, and the resulting assignment of requests to cache locations, order the set of all pairs (i, j) of (cache i, request j) such that cache i serves a request j in non-increasing order of their profits. Consider the vector $p(S)$ of the profit of these pairs in the above order. We claim that this vector is lexicographically increasing as players play a strict best-response move.

To see this, consider a player i that plays a strict best-response move from s_i to s'_i. Consider the first pair (i^*, j^*) of cache i^* and request j^* that disappears from the vector of

	The Profit of the Requests				
	1	2	3	4	Available Bandwidth
Cache Loc. 1	10	2	5.5	5.5	11
Cache Loc. 2	2	3	3	5.6	8

Table 1: The profit of request and available bandwidth for cache locations in the IBDC game without pure Nash equilibria.

profits (i.e., $f_{i^*j^*}$ appears in vector $p(S)$ and not in vector $p(S \oplus s'_i)$). Thus, the profit of all pairs in $p(S)$ with a profit larger than $f_{i^*j^*}$ appear in the vector $p(S \oplus s'_i)$. Since request j^* is not served by i^* in $S \oplus s'_i$, either j^* is served by player i in $S \oplus s'_i$, or $i^* = i$ and j is not in s'_i.

In the first case, player i serves request j^* instead of player i^* and thus the profit of i for j^* should be more than $f_{i^*j^*}$. Therefore, there exists the number f_{ij^*} which is strictly greater than $f_{i^*j^*}$ in vector $p(S \oplus s'_i)$ instead of $f_{i^*j^*}$; thus the vector is lexicographically increasing. In the latter case, since $(i, j^*) = (i^*, j^*)$ is the pair with the largest profit whose profit disappears from the vector $p(S)$ and player i increased his payoff playing his best response from s_i to s'_i and the bandwidth requirement of all requests are the same, there exists another request $j' \in s'_i$ with profit $f_{ij'}$ such that $f_{ij'}$ is greater than f_{ij^*}; and $f_{ij'}$ appears in vector $p(S \oplus s'_i)$ instead of $f_{i^*j^*}$. Therefore, in the latter case the vector is lexicographically increasing as well.

We note that the existence of pure Nash equilibria for the uniform IBDC game when there are no ties among profits can be derived from the existence of stable matchings in a general setting with social choice functions with substitutability property [11]. However, cycles of strict best-response moves are known even for stable matching games [31] [13]. Our proof for this variant of the uniform IBDC indicates that not only PSNE exists, but also any sequence of strict best-response moves of players converges to a PSNE.

For the uniform CapDC game, in the example below, we demonstrate a cycle of strict best-response moves. We do not know if the uniform CapDC game or the uniform IBDC game always have a PSNE or not.

THEOREM 6.6. *There exist cycles of strict best-response moves in the uniform* CapDC *game.*

Consider an instance of the distributed caching game with three players and three request types, two requests from each type. The available capacity of each player is one and the size of each request type is one. Let players be 1, 2, and 3. Let request types be A, B, and C. There are two requests of type A, i.e, A_1 and A_2, two requests of type B, i.e., B_1 and B_2, and two requests of type C, i.e., C_1 and C_2. In each strategy profile, each player can choose both requests of only one request type. The profit of all requests are depicted in Table 2. Since, each player can cache both requests of only one type, we can refer to the strategy of a player as one type. In the following, by a type as the strategy of a player, we mean the set of both requests of that type.

In this instance, starting from the strategy profile (C, C, B) for players, if we let players 2, 1, 3, 2, 1, and 3 play their best responses in this order, players will end up with the same configuration (C, C, B). The complete cycle is depicted in Table 3. In this table, we refer to the strategy of a player as a type and by that we mean both requests of that type.

6.10 CapDC **Game: Poor Convergence to Equilibria** In this section, we prove that there are instances of the uniform CapDC game in which finding a pure Nash equilibrium is PLS-hard [21]. The definition of PLS-hard problems can be found in papers by Yannakakis et. al [21, 32].

We give a reduction from the Max-Cut local search problem with swapping neighborhood to the problem of finding a PSNE in some instances of the uniform CapDC game. In turn, this implies that there are states of this game from which any path of best response moves to the equilibrium has exponential length.

Recall that the Max-Cut local search problem with swapping neighborhood is as follows: Given an edge-weighted graph G and a cut, the local operations are to switch one node from one side of the cut to the other side if it can increase the value of the cut. In this section, WLOG we assume that the graph G is connected.

THEOREM 6.7. *There are instances of the uniform* CapDC *game with pure Nash equilibria*[14] *for which finding a pure Nash equilibrium is PLS-hard.*

[13] In stable matching games, each player has an arbitrary preference list for each request, and each request has an arbitrary preference list for the players. Players offer to a request in their preference list, and each request j goes to a player with the highest priority among the players that offer to j. Each player likes to get a request with a higher priority in his preference list. A cycle of strict best-response moves is known for this game [31]. The IBDC game in which each cache location can cache only one request can be formalized as a special case of the stable matching game.

[14] We can also say that finding a *sink equilibrium* is PLS-hard. A sink equilibrium is a set of strategy profiles that is closed under best-response moves. A pure equilibrium is a sink equilibrium with exactly one profile. This equilibrium concept is formally defined in [16].

Proof.

	Profit of the Requests to Players		
	Player 1	Player 2	Player 3
A_1	2	0	1
A_2	3	0	4
B_1	0	6	5
B_2	0	2	1
C_1	1	7	0
C_2	3	2	0

Table 2: The profit of the requests for the example of the CapDC game with a cycle of strict best responses.

Players	Player's Type (Player's Payoff)					
1	C(3)	C(4) $\rightarrow$	A(5)	A(2)	A(2) $\rightarrow$	C(3)
2	C(7) $\rightarrow$	B(8)	B(8)	B(8) $\rightarrow$	C(9)	C(7)
3	B(6)	B(0)	B(0) $\rightarrow$	A(4)	A(4)	A(5) $\rightarrow$

Table 3: A cycle of size 6 of best-responses in the uniform CapDC game. Each column represents the vector of request types that the players provide. The numbers in parenthesis in each column are the payoffs of players. The arrow ($\rightarrow$) indicates that a player plays his best response and changes his strategy to the request type in the next column.

Proof. We give a reduction from the local search Max-Cut problem with swapping neighborhood to the uniform CapDC game.

Consider an instance $G = (V, E)$ of the Max-Cut problem with weights $w : E(G) \rightarrow N$ on edges. We construct an instance $\mathcal{R}(G)$ of the CapDC game as follows: Each player corresponds to a vertex of graph G. There are two types of requests. The size of each request type is equal to one and the capacity of each cache location is one. For each edge, we define two requests. Edge uv has a request p_{uv} of type one and a request q_{uv} of type two. We assume that the connection costs of both requests on edge uv to either u or v is zero. The connection costs of these two requests to any other vertex is greater than $w(uv)$. The reward of each of these two requests is $w(uv)$.

Each player caches either requests of type one or requests of type two. If both u and v cache requests of the same type then they each get profit $\frac{w(uv)}{2}$ from the requests from edge (u, v). If they cache different types then they each get profit $w(uv)$ for this edge. Thus, given a strategy profile, the total profit obtained is exactly $w(E) + w(C)$ where C is the set of edges with one player caching type 1 and the other caching type 2. In other words, C is the cut set defined by the cut where all vertices that cache type 1 are on one side and all vertices that cache type 2 are on the other.

From the definition of the game $\mathcal{R}(G)$, if s'_i is a best-response move of player i in strategy profile S, we claim that either s'_i contains all requests of type 1 on edges adjacent to i, or s'_i contains all requests of type 2 on edges adjacent to i. The reason is that if s'_i does not contain all requests of one type on edges adjacent to i then player i can strictly increase his payoff by including the rest of requests of the same type on edges adjacent to him in his strategy (here, we use the fact that G is connected.). We let $\mathcal{L}$ be the set of strategy profiles S of the instance $\mathcal{R}(G)$ in which each player i plays a set s_i such that s_i either contains all requests of type one on edges adjacent to player i, or s_i contains all requests of type two on edges adjacent to player i. From the above discussion, it follows that any PSNE of game $\mathcal{R}(G)$ is in $\mathcal{L}$.

From a feasible strategy profile $S \in \mathcal{L}$ of $\mathcal{R}(G)$, we construct a cut $\mathcal{M}(S, G)$ in the Max-Cut local search problem on G. For a player u in the game $\mathcal{R}(G)$ if u caches requests of type one, then we put u in side one of the cut $\mathcal{M}(S, G)$, and if u caches requests of type two, then we put u in side two of $\mathcal{M}(S, G)$.

In a strategy profile $S \in \mathcal{L}$ of game $\mathcal{R}(G)$, player u can strictly improve his payoff by playing switching from requests of type 1 to requests of type 2 (or vice-versa) if and only if the value of the cut set $\mathcal{M}(S, G)$ strictly improves by moving u to the other side of the cut. Thus, if a strategy profile $S \in \mathcal{L}$ is a pure Nash equilibrium in game $\mathcal{R}(G)$, then $\mathcal{M}(S, G)$ is a local optimal solution of the Max-Cut local search problem for graph G.

Thus, if we have a polynomial-time algorithm for finding a pure Nash equilibrium S (or a sink equilibrium) in any instance of the uniform CapDC game, since this pure Nash equilibrium is in $\mathcal{L}$, this implies a polynomial-time algorithm for finding a local optimum $\mathcal{M}(S, G)$ of any instance of the Max-Cut local search problem with swapping neighborhood. The PLS-hardness follows from a PLS-completeness result of Schaffer and Yannakakis [32].

Using the above proof and a result of Schaffer and Yannakakis [28, 32], we can show that in some instances of the uniform CapDC game there are states from which all paths of best responses have exponential length.

COROLLARY 6.1. *There are instances of the uniform* CapDC *game that have pure Nash equilibria with states from which any sequence of best-response moves to any pure Nash equilibrium (or sink equilibrium) has an exponential length.*

Proof. In [32], it is shown that there exists a weighted graph G, and an initial cut C, such that the length of any sequence of local operations for the Max-Cut local search problem from C to any local optimal solution is exponential. Consider the CapDC game $\mathcal{R}(G)$ defined in the previous proof and the set $\mathcal{L}$ of strategy profiles. From the cut C of G, we can easily construct a strategy profile $S \in \mathcal{L}$ such that $\mathcal{M}(S, G) = C$ by letting player u play all requests of type 1 or 2 in S, if vertex u is in the side 1 or 2 of the cut C, respectively. We claim that any sequence of best responses from S to any PSNE in $\mathcal{R}(G)$ has exponential length.

Assume for contradiction that there is a sequence of strategy profiles $S_0 = S$, S_1, …, S_t where $t \leq \text{poly}(n)$ and player u_i plays his best-response move from S_i to S_{i+1} and S_t is a PSNE. Since $S_0 = S \in \mathcal{L}$ and u_i plays his best-response move from S_i to S_{i+1}, by induction it follows that $S_i \in \mathcal{L}$ for all $0 \leq i \leq t$. It is easy to see that $C = \mathcal{M}(S_0, G)$, $C_1 = \mathcal{M}(S_1, G)$, $C_2 = \mathcal{M}(S_2, G)$, …, $C_t = \mathcal{M}(S_t, G)$ is a sequence of cuts such that cut C_{i+1} results from the cut C_i by the local operation of vertex u_i and C_t is a local optimal solution. Therefore, $t \leq \text{poly}(n)$ implies that there exists a sequence of local operations of polynomial length from cut C to a local optimal solution. This contradicts the assumption that no such sequence of polynomial length from cut C exists.