Tight Approximation Algorithms for Maximum Separable Assignment Problems

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 [20] for the details of many missing proofs.

Previous Work. For the special case of GAP, Shmoys and Tardos [28] 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 [28]. 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 [30], 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, 29], 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 [16] 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 [22] 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 [31]. In the conclusion of [3], they suggest considering the problem with bandwidth as an important extension.

Goemans, Li, Mirrokni, and Thottan [13], 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, [13] 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_S^i 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 $i \in U$, $\sum_{S \in \mathcal{I}_i} X_S^i = 1$. Moreover, we cannot assign each item to more than one bin: $\sum_{i, S \in \mathcal{I}_i : j \in S} X_i^S \leq 1$. Our objective is to find an assignment of items to bins to maximize the sum of profits, i.e., $\sum_{i, S \in \mathcal{I}_i} f_i^S X_i^S$ where $f_i^S = \sum_{j \in S} f_{ij}$. By relaxing the 0-1 variables to nonnegative real variables, we obtain the following linear programming relaxation:

$$
\begin{aligned}
(2.1) \quad \max \quad & \textstyle\sum_{i \in U, S \in \mathcal{I}_i} f_i^S X_i^S & \\
\text{s.t.} \quad & \textstyle\sum_{i \in U, S \in \mathcal{I}_i : j \in S} X_i^S \leq 1 & \forall j \in H \\
& \textstyle\sum_{S \in \mathcal{I}_i} X_i^S = 1 & \forall i \in U \\
& X_i^S \geq 0 & \forall i \in U, \forall S \in \mathcal{I}_i
\end{aligned}
$$

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_S^i. 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 [15]):

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

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.

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

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:

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

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 [17]. An approximate solution to the linear programs (2.1) and (2.2) can also be obtained via Lagrangian LP algorithms [11, 25, 33, 34].

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 [22]); 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 [27] 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 [12, 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.

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

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

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 (2.4) 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 [18] 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 5.3. 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 [24] 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 [13], 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 [20].

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 ($\sum_{t \in \{t_j | j \in D_i\}} a_t \leq A_i$). 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 [21]. 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 [32]. The results in [32] imply that the price of anarchy is 2. However, it is simpler to give a direct proof of Theorem 5.1. 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 [19, 23, 26].

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.

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 [26].

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.

[8] We can also say that finding a *sink equilibrium* [14, 20] 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 [14, 20].

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] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

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

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

[20] V. S. Mirrokni. *Approximation Algorithms for Distributed and Selfi sh Agents*. MIT Thesis, 2005.

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

[22] 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.

[23] 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.

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

[25] 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.

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

[27] 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.

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

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

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

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

[32] 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.

[33] N. Young. Randomized rounding without solving the linear program. In *7th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 170–178, 1996.

[34] 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.