An Experimental Study of the Misdirection Algorithm for Combinatorial Auctions

Jörg Knoche* and Piotr Krysta**

Dept. of Computer Science, University of Dortmund

Abstract. Single-minded combinatorial auctions (CA) are auctions in which a seller wants to sell diverse kinds of goods and each of the potential buyers, also called bidders, places a bid on a combination, i.e., a subset of the goods. There is a severe computational limitation in CA, as the problem of computing the optimal allocation is NP-hard and even hard to approximate. There is thus interest in polynomial time approximation algorithms for this problem. Recently, many such approximation algorithms were designed, among them greedy and local search based algorithms. One of these is a so-called misdirection algorithm combining both approaches and using a non-standard, misdirected, local search approach with neighborhood of size 2. This algorithm has the best known provable approximation ratio for the problem in terms of the sizes of bids. Its analysis, however, is quite complicated. We study this algorithm and its variants on typical instances designed for CAs. On question is if larger neighborhood helps – the question that seems quite difficult to address theoretically at the moment, taking into account already complex analysis for size 2 neighborhood. We also study experimentally other aspects of the misdirection algorithm, and finally present a comparison to other approximation algorithms.

1 Introduction

There has been an increasing interest in the recent years in so-called combinatorial auctions (CA). These are auctions where a seller wants to sell diverse kinds of goods and the potential buyers, called bidders, place bids on the combinations, i.e., subsets of goods. Such auctions were suggested for auctioning, e.g., spectrum licenses, landing slots or computational resources, see [23] for a survey.

When the auction concerns many related kinds of goods, combinatorial auctions are particularly well suited as they allow buyers to express their valuations on combinations of goods, which should lead to more economically efficient allocations. One of the main obstacles of dealing with combinatorial auctions is the

* Department of Computer Science, Dortmund University, Baroper Str. 301, 44221 Dortmund, Germany. E-mail: joerg.knoche@cs.uni-dortmund.de. The author is partially supported by DFG grant Kr 2332/1-2 within Emmy Noether program.

** Department of Computer Science, Dortmund University, Baroper Str. 301, 44221 Dortmund, Germany. E-mail: piotr.krysta@cs.uni-dortmund.de. The author is supported by DFG grant Kr 2332/1-2 within Emmy Noether program.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 265–278, 2006.

computational hardness of the problem of determining the optimal allocation for a given collection of buyers. This problem, called also winner determination, is known to be NP-hard and even hard to approximate [15].

For simplicity, we assume that each bidder desires only a single subset of goods and places a positive valuation only on this particular set of goods (any superset obviously values the same for this bidder). Such kind of restricted bidders, called single-minded, has been introduced by Lehmann et al. in their seminal paper [15], and has since been intensively investigated, e.g., [2,17,7,14].

In this paper we are interested in the winner determination problem in single-minded CAs, well known in discrete optimization as the set packing problem. Given a family of subsets of a given universe, each subset with a prescribed value, this problem asks for a maximum value set packing, i.e., a pairwise-disjoint subfamily of the subsets. Here, each single subset models a single bidder desiring that subset and its value is the bidder's valuation. The disjointness constraints corresponds to the fact that the seller cannot sell the same good to two bidders.

Since the set packing problem is NP-hard and even hard to approximate, polynomial time approximation algorithms are of interest. In fact, many approximation algorithms have been designed for this problem, see, e.g., [11] for a survey. We will be interested in approximating this problem in terms of d, which is the size of the largest set in the input family. Having the value of d small in terms of combinatorial auctions (CA) means that the bidder's preference sets are of size at most d, which clearly is the natural assumption for the bidders.

It is known that even unweighted, i.e., when values of the sets are unit, set packing problem with sets of size at most d is NP-hard to approximate to within $O(d/\log d)$ [12]. We cannot, thus, expect a much better than merely d approximation factor for this problem in polynomial time. It is quite easy to obtain a precisely d-approximation algorithm via the greedy method. In fact, a d-approximation with a running time $O(|\mathcal{S}|^2)$ was given by Hochbaum [13], and a $|\mathcal{S}|^{O(1/\epsilon)}$-time $(d-1+\epsilon)$-approximation for any fixed $\epsilon > 0$, by Bafna et al. [4] and Arkin & Hassin [3], where $\mathcal{S}$ is the given family of sets.

It turns out that even reducing the constant in front of d is a challenging problem. Chandra and Halldórsson [8] succeed to give a $\frac{2}{3}(d+1)$-approximation for this problem, at the expense of running time of $\Omega(|\mathcal{S}|^d)$, which is not polynomial if d is not a fixed constant. Berman [5] has improved this ratio to $\frac{1}{2}(d+1)$ but the running time is also $\Omega(|\mathcal{S}|^d)$. The first known better than d approximation in polynomial time is due to Berman and Krysta [6], who gave a factor $\frac{2}{3}d$ approximation for this problem in time roughly $O(|\mathcal{S}|^2)$. This last result is also the best known to date approximation ratio for the considered problem.

Berman and Krysta [6] consider in fact a slightly more general problem, that is, a maximum weighted independent set problem in a $(d+1)$-claw-free graph. Their algorithm is based on the local search method with a local misdirected [1]

[1] Misdirection is meant to mean here that while performing one step of the local search, we locally optimize a different (misdirected) objective than the original one.

objective and neighborhood of size 2 [2]. The theoretical analysis of this algorithm in [6] is quite technical and complex. Thus, for instance, a natural question of extending this analysis to neighborhood of size 3 seems quite challenging. This is the place where experimental analysis may help. This, and other questions and properties of the misdirection algorithm that seem difficult, or even impossible to address theoretically, are the subject of this experimental paper. We also study other greedy algorithms and an LP-based randomized rounding algorithm.

Outline. The rest of this paper is organized as follows. Section 2 has formal definitions, description of the algorithms and instances. Section 3 includes the experimental analysis of the misdirection algorithm. Section 4 compares all algorithms by running time and approximation factor.

2 Problem Definition, Algorithms and Instances

We formally define the set packing problem. We are given a finite set of goods U with $|U| = m$, and a family of subsets of U denoted by $\mathcal{S} \subseteq 2^U$. A given set $S \in \mathcal{S}$ models a bidder, and thus we identify set $\mathcal{S}$ with the set of bidders, and $|\mathcal{S}| = n$. Let also each $S \in \mathcal{S}$ have an associated weight $w(S) \in \mathbb{R}_+$, modeling the valuation of bidder S. The set packing problem (winner determination in CAs) asks for finding a packing, that is, a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ such that any two distinct sets $S, T \in \mathcal{S}'$ are disjoint, $S \cap T = \emptyset$, and the total weight of $\mathcal{S}'$, $w(\mathcal{S}') = \sum_{S \in \mathcal{S}'} w(S)$, is maximized. We will assume in this paper that $d \in \mathbb{N}_+$ is the maximum size of any bid, i.e., $\max\{|S| : S \in \mathcal{S}\} = d$. Set packing problem is known to be NP-hard to approximate to within a factor of $O(d/\log d)$ [12].

2.1 Description of the Algorithms

We describe below the approximation algorithms for set packing we will use.

Greedy-1: This is algorithm Greedy-1 from paper [14]. It first sorts the sets in $\mathcal{S}$ by non-increasing values of $w(S)/\sqrt{|S|}$, and then goes through all sets in this order and puts them into the solution maintaining the feasibility of the packing. Since our input data is represented as a 0/1 matrix A such that $A_{e,S} = 1$ iff $e \in S$, implementing Greedy-1, the time to compute $|S|$ for each $S \in \mathcal{S}$ is taken into account. Greedy-1 is know to be $\sqrt{m}$-approximate for set packing [15].

Greedy-2: The same as Greedy-1 above, but the sorting is with respect to non-increasing values of $w(S)$. Note, that this algorithm is a bit faster than Greedy-1, since we do not need to compute $|S|$ for $S \in \mathcal{S}$. Hochbaum [13] shows that it is d-approximate for set packing.

Greedy-3: The same as Greedy-1 above, but the sorting is with respect to non-increasing values of $w(S)/|S|$. We did not put this algorithm into our

[2] We use the term neighborhood to denote a parameter ℓ in the misdirection algorithm, but in fact the "real" neighborhood size is roughly $O(|\mathcal{S}|^\ell)$.

diagrams – see further explanations below. In theory Greedy-3 has an approximation ratio of d (in fact [14] shows ratio $d+1$ for Greedy-3 on a more general problem, but the analysis of [14] slightly modified shows ratio d for set packing).

Misdirect-noGreed: This is the basic misdirection algorithm of Berman and Krysta [6]. It starts with an empty solution and performs all possible local exchanges. While this algorithm is defined in [6] for a slightly more general than set packing problem we redefine it here for the latter problem. Before we describe it we need some notation. Let $\mathcal{P}, \mathcal{R} \subseteq 2^U$ be two given set families. We define $N(\mathcal{P}, \mathcal{R}) = \{R \in \mathcal{R} : \ \exists P \in \mathcal{P} \text{ such that } P \cap R \neq \emptyset\}$. We also define $\mathcal{P} \triangleleft \mathcal{R} = (\mathcal{P} \setminus N(\mathcal{R}, \mathcal{P})) \cup \mathcal{R}$. Observe that if family $\mathcal{P}$ and family $\mathcal{R}$ is a packing then so is family $\mathcal{P} \triangleleft \mathcal{R}$. Let also $w^\alpha(\mathcal{P}) = \sum_{P \in \mathcal{P}} (w(P))^\alpha$, for a given $\alpha > 1$. Algorithm Misdirect-noGreed, also called $\ell - Imp^\alpha$, with neighborhood size $\ell = 2$ and value of $\alpha = 1.71$ (from [6]), is as follows.

Algorithm $2 - Imp^\alpha$
  $\mathcal{P} \leftarrow \emptyset$
  while there exists pair of sets $\{S, T\} \subseteq \mathcal{S}$ that improves $w^\alpha(\mathcal{P})$ **do**
    $\mathcal{P} \leftarrow \mathcal{P} \triangleleft \{S, T\}$

This algorithm can be described in words simply as follows: start from an empty packing, and as long as there is a pair of sets that on adding to the current packing (and removing all conflicting sets) improves the misdirected objective $w^\alpha(\cdot)$, perform such a local exchange. This algorithm may not have polynomial running time, and it is shown in [6] how to make it polynomial – see algorithm Misdirect below. Berman and Krysta prove an upper bound of (roughly) $\frac{2}{3}d$ on the approximation ratio of algorithm $2 - Imp^{1.71}$. More precisely, they prove that $\alpha = 1.71$ is the value of α giving the best approximation ratio of $\frac{2}{3}d$.

Misdirect: This is the original misdirection algorithm described in paper [6]. First it runs algorithm Greedy-2. Let $\mathcal{P}$ be the output greedy packing. Then, rescale the weights so that $w(\mathcal{P}) = k \cdot |\mathcal{S}| = k \cdot n$, for some fixed $k \in \mathbb{N}_+$, and run algorithm Misdirect-noGreed after replacing the function w with $\overline{w}(S) = \lfloor w(S)^\alpha \rfloor^{1/\alpha}$, starting with initial solution $\mathcal{P}$. Berman and Krysta show that the running time of this modified algorithm is bounded by $O(k^\alpha (dn)^{2+\alpha})$, and its approximation ratio is at most $\frac{k}{k-1} \cdot \frac{2}{3} \cdot d$, for any choice of $k \in \mathbb{N}_+$.

RandRound: This is the most typical approximation algorithm, see [18,19,22], for packing problems like set packing. It first solves the linear programming relaxation of the set packing problem and then performs the standard randomized rounding. An iteration may not produce a feasible packing and that is why we repeat the randomized rounding step 750 times and take the best output solution. Please note that we do not optimize the number of iterations of RandRound and take it into account just to compare with the above (combinatorial) algorithms, and our comparison is fair – see further sections. Srinivasan [21] proved that RandRound has an $O(d)$-approximation for set packing, where, in particular the constant in the ratio $O(d)$ is larger than 1.

2.2 Description of the Instances

Vohra / de Vries: These instances are described by Zurel and Nisan [24], and by de Vries and Vohra [23]. The description below follows [24]. These instances are called in our experiments *prob.i.m.n.d*, where $i = 1, 2, 3, 4$ according to the definitions below, and numbers m, n and d are as defined previously. (Some of the instances may not have the last part d in *prob.i.m.n.d* defined.)

1. Random: For each bid, pick the number of goods randomly from $\{1, 2, \ldots, m\}$. Randomly choose that many goods without replacement. Pick the bid weight (valuation) randomly from $[0, 1]$. Then, $m \in \{100, \ldots, 400\}$, $n \in \{500, \ldots, 1000\}$.
2. Weighted Random: The same as for Random, but the bid weight is picked from [0, number of goods in bid]. Then, $m \in \{100, \ldots, 400\}$, and $n \in \{500, \ldots, 2000\}$.
3. Uniform: For each bid, pick a constant number of goods randomly from $\{1, 2, \ldots, m\}$. Randomly choose that many goods without replacement. Pick the bid weights randomly from $[0, 1]$. Then, $m \in \{25, \ldots, 100\}$, $n \in \{50, \ldots, 1100\}$, and the bid size $d \in \{3, 8, 11\}$.
4. Decay: For each bid, give it one random good. Then repeatedly add a new random good with probability γ until that good was not added or the bid contains all m goods. Pick the bid valuations randomly from [0, number of goods in bid]. Also, $m \in \{50, \ldots, 200\}$, $n \in \{50, \ldots, 200\}$, and the probability $\gamma \in [5\%, 95\%]$.

CATS: These are instances generated by the CATS program described in paper [16] by Leyton-Brown, Pearson and Shoham. The used distributions are arbitrary, paths, regions and scheduling. We have used the standard parameters to generate these instances, and only the number of bids and goods was varying. For more precise description of this instance generator see [16] and the web page `http://cats.stanford.edu/`. These instances are referred in our paper to as *name.m.n*, where $name \in \{arb, paths, reg, sched\}$ and the names $\{arb, paths, reg, sched\}$ correspond to the ones above.

Fujishima / Sandholm: These are the instances described in [1], which can be found at web page `http://user.it.uu.se/~tein/cmb/index.html`. They contain instances generated according to random, uniform, decay, binomial and exponential distribution. The distributions used in our experiments are uniform, binomial with 1500 bids and exponential. From each distribution the first 5 instances have been taken. We refer to those instances as *name.nr*, where $name \in \{exp, uni, bin\}$ and nr is the number of the instance. Following the cited paper, we keep m, n fixed to some specific values.

Uniform ([20]): Draw the same number of randomly chosen items for each bid. Pick an integer valuation from $[500, 1500]$ and multiply by the number of commodities. The number of goods and bids are fixed to $m = 100$ and $n = 500$.

Binomial ([9]): The probability distribution for a bid requesting j goods out of m goods in the market is $f(j) = p^j \cdot (1-p)^{m-j} \cdot \binom{m}{j}$ with $p = 0.2$. An integer

valuation is drawn from 500 to 1500 and multiplied by j. The number of goods and bids here are $m = 150$ and $n = 1500$.

Exponential ([9]): The probability distribution is defined as $f_e(j) = c \cdot e^{-j/5}$ (c is implicitly defined by $\sum_{j=1}^{m} f_e(j) = 1$, where m as before is the number of goods). The valuation is an integer, rectangularly drawn from $[500, 1500]$ and multiplied by the number of requested goods j. Again, we fix $m = 30$ and $n = 3000$.

Random: These instances were randomly generated by us with a fixed number of goods m, bids n, and goods per bid (the d value). We choose randomly n subsets (bids) of size d out of m goods (possibly with repetitions). Then for each generated bid its weight is randomly chosen from $[0, 1]$ and multiplied by d. We call these instances $Randomx$, where x is the serial number of the instance.

Test-Setup: All algorithms are implemented in Java 1.4.2 and run on AthlonXP 1900 MHz machine with 768 MB RAM under WindowsXP with Service Pack 1. In all tested instances the optimal solution was found by using CPLEX 6.5.2.

3 Analyzing the Misdirection Algorithm

3.1 Proved Versus Achieved Approximation

A natural question after one succeeds to prove a bound on the approximation ratio is how rough this bound is as compared to one obtained on typical instances. Indeed, also in the case of the misdirection algorithm $2 - Imp^{1.71}$ with the starting greedy solution this bound turns out to be rough. This, of course, is not surprising, but just confirms the known phenomenon that most likely there are only few worst-case, untypical instances. For some data, see Figure 1.

instance	prob.1.100.1050.3	prob.3.75.50.8	prob.3.100.50.3	Random1	bin150	bin1506	uni31
d	3	8	3	5	47	48	3
proved-apx.	2,000	5,166	2,000	3,229	30,351	30,996	2,000
achieved-apx.	1,086	1,352	1,037	1,000	1,007	1,0	1,091

instance	prob.3.100.100.3	prob.3.100.1300.3	prob.3.75.100.13	exp50	exp51	exp52	exp53
d	3	3	13	4	4	4	4
proved-apx.	2,000	2,000	8,395	2,606	2,606	2,606	2,606
achieved-apx.	1,000	1,096	1,214	1,055	1,038	1,026	1,056

Fig. 1. The proved approximation ratio is just $\frac{2}{3}d$, and the achieved ratio is calculated by comparing to the optimal solution.

3.2 Larger Neighborhood

We investigate here one of our main questions concerning algorithm Misdirect, namely if it is worth investing time to find theoretical analysis showing better factors when $\ell \geq 3$. Let us first consider a tight example in [6] for the ratio of $\frac{2}{3}d$

of $2 - Imp^{\alpha}$ when $d = 3$. This is given by two examples in Lemma 2.1 and 2.2 in [6]. These two lemmas together imply Lemma 2.3 in [6] stating that for $d = 3$ the ratio is at least $(\sqrt{5d^2 - 8d + 4} + 2 - d)/2 = \frac{2}{3}d = \beta \cdot d = 2$, where $\beta^{\alpha} = \frac{1}{2}$, which implies $\beta = \frac{2}{3}$ and $\alpha \approx 1.71$, which is the best theoretical value of α for $d = 3$ found in [6]. (Note, taking $\alpha \approx 1.71$ gives approximation factor $\frac{2}{3}d$ for *all* values of d. For other values of d slightly better values of α are found [6].)

It can, however, easily be checked that the locally optimal solutions w.r.t. $\ell = 2$ in these two examples from [6] are not locally optimal anymore when $\ell = 3$ and parameter β from [6] fulfills $\frac{1}{3} < \beta^{\alpha} < 1$. We, therefore, see that the size 3 neighborhood indeed helps, but certainly this is not enough evidence. What about typical instances ?

We judge here the improvement in the weight of the solution when we change the neighborhood size ℓ in algorithm Misdirect from 2 to 4 on CA instances. We have selected the instances in Figure 2 (a) only from type *prob.i.m.n.d* (additional number in the brackets is just the serial number). The reason being that these are quite small instances and the running time increases rapidly when ℓ is raised. Obviously, we did not run the tests for larger ℓ if we reached optimum, i.e., 100% earlier. For each value of $\ell \in \{3, 4\}$ we used one value of α for all tested instances, namely the best found.

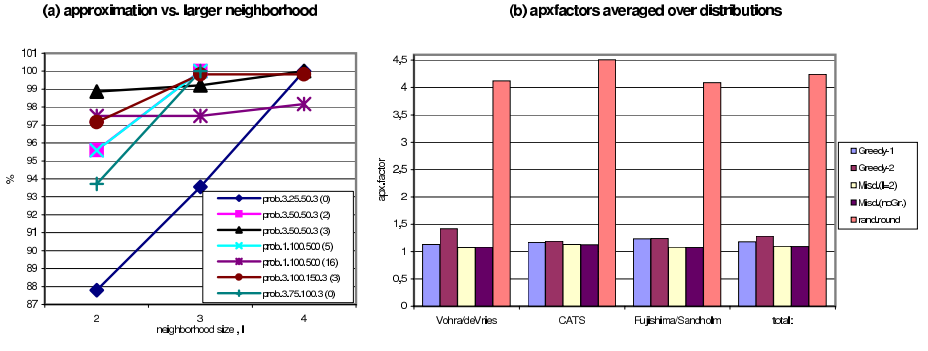


Fig. 2. (a) 100% on the vertical axis corresponds to the weight of an optimal solution. (b) shows approximation factors of all algorithms averaged over all instances.

Conclusions: It seems it is worth to try to analyze $\ell - Imp^{\alpha}$ theoretically for larger values of ℓ. Even in the typical instances we selected we observe an improvement in the approximation ratio ranging from 1% to about 10%. This seems not much, but note, that our instances are average "typical" ones. On the other hand it is plausible that there are better improvements possible on other larger instances as well, but so far our limiting factor for such tests was rapidly increasing running time of our implementations.

Testing which α is best for $\ell \geq 3$ we observed that smaller values give better results for larger neighborhoods. Our diagram in Fig. 2 (a) shows the best solution found over all α tested (between 1 and 2 with step size 0.01), but we

found one value of $\alpha = 1.05$ which gives these best results for $\ell = 3$. Same value of α gives best results for most of the instances when $\ell = 4$, but we could not test many of them because of high run time. For a majority of instances an α smaller than 1.1 seemed to be best. But α-values up to 1.2 may also give better results depending on the instance. For $\ell = 2$ we found that on roughly 30% of the instances smaller values of α in $(1.0, 1.2]$ were better than 1.71.

3.3 Misdirected Versus Standard Local Search

In this section we compare the standard local search, that is with $\alpha = 1$, with the misdirected local search, that is with $\alpha > 1$, both with neighborhood of size 2. More precisely, we compare $2 - Imp^1$ with $2 - Imp^{1.71}$. Both used algorithms start with an empty initial solution (Misdirect-noGreed).

The diagrams in Fig. 3 show the increase of the solution's original weight, $w(S)$, at every local exchange for the respective algorithms. The weight of the final solution output by $2 - Imp^{1.71}$ algorithm is 100% on the vertical axis.

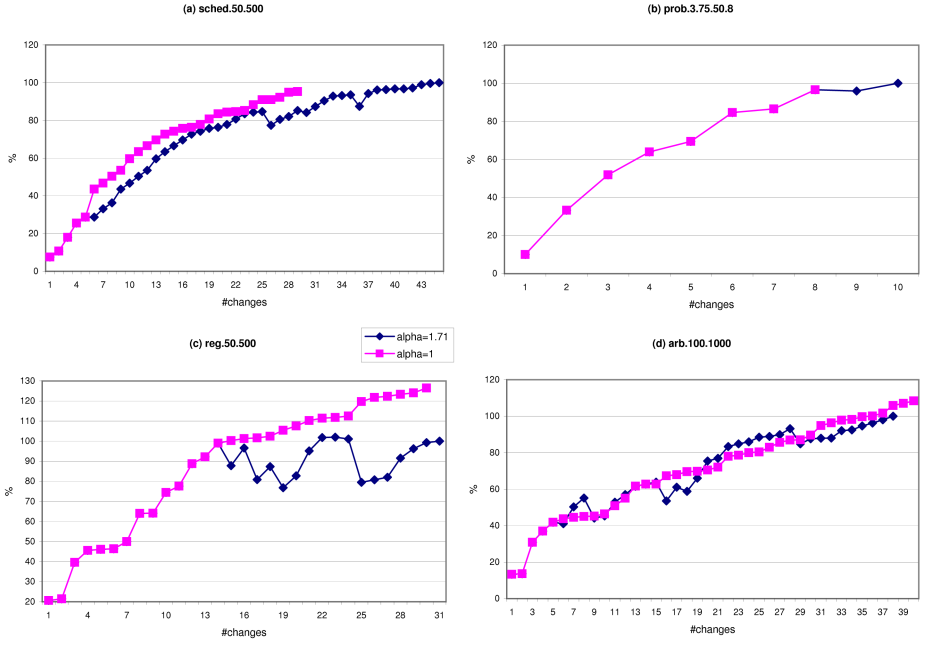


Fig. 3. Misdirected vs. standard local search

Conclusions: One can observe that of course in case of $2 - Imp^{1.71}$ there are many jumps down, which in many cases lead to better locally optimal solutions in the future exchanges.

Berman and Krysta [6] show an example on which the misdirected algorithm avoids some bad local optima that would lead to an approximation factor of d,

which is the case for the standard local search, instead of factor of $\frac{2}{3}d$ for the misdirected algorithm. In fact, this example is quite specific one. A question is if such a behavior also occurs for typical instances. Indeed, we found some instances where this is the case, see Figure 3 (a) and (b). There are also instances in which the misdirected solution was worse – see Figure 3 (c) and (d).

The major kind of behavior that we observed is, however, the fact that in about 80% of all the instances we tested the two curves for $2 - Imp^{1.71}$ and for $2 - Imp^{1}$ split at some exchange, earlier or later, and at the very point of the split, the weight of the solution of $2 - Imp^{1.71}$ jumps down. As examples, see Figure 3. Also in these majority of instances the $2 - Imp^{1.71}$-curve is below the $2 - Imp^{1}$-curve. We also observe that the remaining 20% of the instances where this behavior does not occur, do not come from one specific type of instances. We have not observed a clear correlation between the jump down and the fact that the final Misdirect solution was better than that of standard local search.

3.4 How Fast Is the Local Optimum Reached?

We have taken into account here how many local exchanges are needed to reach the final locally optimal solution for the Misdirect-noGreed algorithm.

In Figure 4 we draw the quality of the solution (w.r.t. the original weight $w(\cdot)$) as a function on the number of local exchanges. We see that within one class of instances there are easier instances, where there are small number of high jumps, and there are harder instances, where we have many small jumps. One striking observation is that for Fujishima/Sandholm instances, see Fig. 4 (a) the instances have been clearly divided into easy, medium, and hard. In particular easy are instances generated w.r.t. binomial distribution, medium – ones generated with exponential distribution, and hard – generated with uniform distribution. Observe, also, that in the case of "medium", exponential instances there are many jumps down, which may suggest that in those cases there are many bad local optima and they are avoided by Misdirect-noGreed.

A similar picture can be obtained for Misdirect-noGreed with neighborhood of size 3 in Figure 4 (d), where the easiest instances are the ones of Vohra/de Vries – see also Fig. 4 (c) for those instances and neighborhood of size 2.

Considering Misdirect-noGreed with $\ell = 2$ on Vohra/de Vries instances we found that there are also all levels of difficulty, see Fig. 4 (c).

Finally, we observed that in the case of Misdirect, increasing k which is used in scaling the weights does not increase its running time on tested instances (though a bound of $O(k^{\alpha}(dn)^{2+\alpha})$ on the running time in [6] suggests the opposite). To explain this we found instances where the value of $\overline{w}(S)^{\alpha}$ increases by more than 1 in the exchanges, and not just by 1 as assumed in the $O(k^{\alpha}(dn)^{2+\alpha})$ bound.

4 Comparing All the Algorithms

This section is devoted to the comparison of all the algorithms that we tested, that is the two greedy algorithms and two misdirection algorithms, and the randomized rounding algorithm.

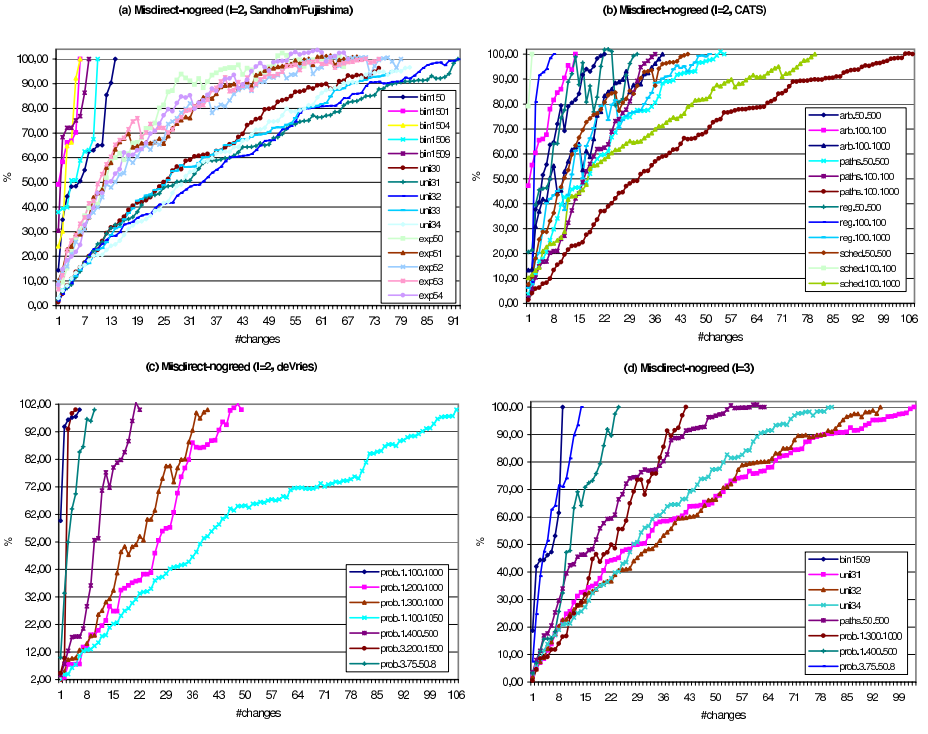


Fig. 4. How fast is the local optimum reached by Misdirect-noGreed for all distributions?

4.1 Running Times

In terms of running time it is no surprise that the greedy algorithms are much faster than the misdirection ones. For a comparison see Fig. 5, where we averaged over 13 instances for the Vohra/de Vries distributions, 12 instances of the CATS distributions and 15 instances of the Fujishima/Sandholm distributions.

Conclusions: We see that typically, the greedy algorithms are faster than the misdirection ones by at least a factor of 100. We also see that the Fujishima/Sandholm instances are most time demanding for all algorithms, and the CATS instances are somehow least time demanding for all algorithms.

Note that when calculating the running time of RandRound in Fig. 5 we are fair and only take into account the randomized rounding iterations and disregard the time for solving the LP relaxation. The reason for this is that we solve the LPs exactly by CPLEX, but for such packing LPs there are faster (approximate) LP solvers, e.g., [10].

For RandRound we observe that our 750 iterations of randomized rounding phase lead to running time higher than that of greedy algorithms (Fig. 5), but the approximation factors achieved are much worse (Fig. 6, Fig. 2 (b)).

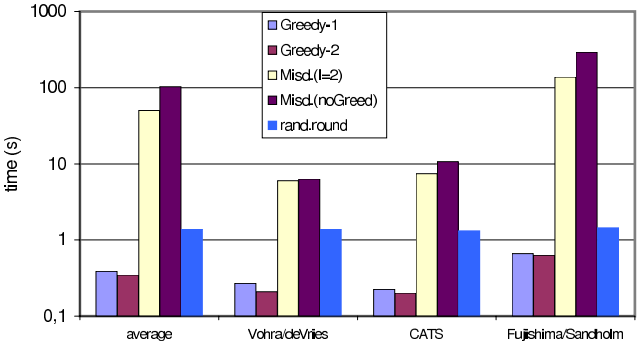


Fig. 5. Running times of all algorithms averaged over the instances

We have also found two instances from the Vohra/de Vries distribution where all the algorithms have a larger running time than on the other instances. Both have a small d value of 3. Thus, possibly, the running time increases, when the number of bids is the same but the d value is smaller. Finally we observed that obtaining optimal solution with CPLEX was 10 to 100 times slower than running a greedy algorithm. Except one type of the Fujishima/Sandholm instances, namely exponential, where CPLEX was about 4 times faster than greedy.

4.2 Approximation Factors

We first describe the diagrams for comparing the approximation factors. Fig. 6 (a), (b) shows the approximation factors for every algorithm depending on the number of goods. Each data point is an average over 3-10 instances. Fig. 6 (c), (d) shows the approximation factors for every algorithm depending on the number of bids, where each data point is an average over 3-5 instances. Finally, Fig. 2 (b) shows the approximation factors averaged over all instances of a distribution. The Vohra/de Vries value is averaged over 13 instances. The CATS value is averaged over 12 (3 of each kind of distribution), and the Fujishima/Sandholm value–over 15 instances (5 of each distribution type).

Conclusions: We see that Misdirect has the best approximation factors which as we know is also the case in the theoretically proven results. This, however, is with the expense of much higher running times – see the previous subsection. Also, we did not put here Greedy-3 into the diagrams, because we observed that Greedy-1 always had better approximation ratio than Greedy-3 (there are only very few instances of the Fujishima-Sandholm type where the ratio of Greedy-3 is better only by 0.02%). This is somehow interesting since in theory Greedy-1 has ratio roughly $\sqrt{m}$ (which is $\sqrt{30} \approx 5.47$ for those instances), but Greedy-3 has ratio d (which is 4 for those instances).

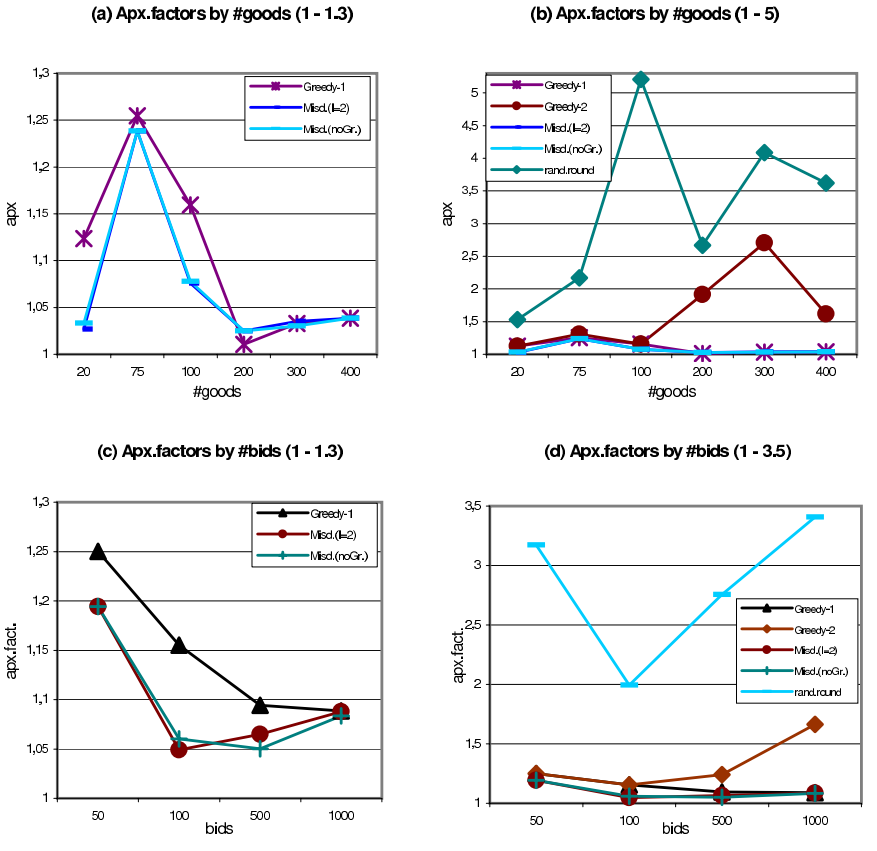
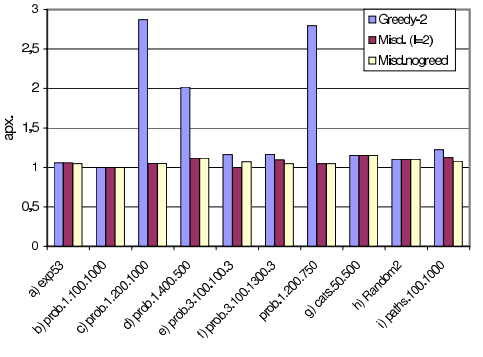


Fig. 6. Approximation factors of all algorithms averaged over the instances



	Greedy-2	Misd. (l=2)	Misd.noGreed
a)	1,281	420,672	820,016
b)	0,11	1,360	1,328
c)	0,187	2,500	2,300
d)	0,094	1,016	0,969
e)	0,015	0,078	0,031
f)	0,844	31,156	27,313
g)	0,047	0,625	0,609
h)	0,016	0,031	0,016
i)	0,765	54,766	97,672

Fig. 7. Approximation factors of Misdirect with/without Greedy and of Greedy itself. The table on the right is the running time (s) for those instances.

4.3 How Profitable Is Using Greedy Inside Misdirect?

Fig. 7 shows that using greedy solution as the starting point for the misdirection algorithm does not lead to much better approximation factors. Also, the running times are not much different between Misdirect with and without Greedy.

Final conclusion: The best algorithm suggested by our experiments is Greedy-1 if we want a good trade-off for both running time and approximation quality. It also wins when the simplicity and easy implementation are the concerns. Greedy-1 is fastest, and gives in most instances the best ratio and in all instances ratios worse than slowest Misdirect, by at most 8%.

5 Conclusion and Future Work

Our experiments suggest that it should be interesting to try to theoretically prove better ratios for Misdirect with larger neighborhoods. The theoretical bounds on the running time of Misdirect appeared to be quite rough on typical instances. We also observed some interesting behavior on how Misdirect avoids some local optima. Among the tested algorithms Greedy-1 turns out to be best if we want a fast algorithm, good approximation ratios and simple implementation. For the future, we plan to also test other algorithms for multi-packing problems, and conducting experiments on larger instances.

Acknowledgment. We would like to thank Piotr Berman for some useful discussions on experimental analysis.

References

1. A. Andersson, M. Tenhunen, F. Ygge. Integer Programming for Combinatorial Auction Winner Determination. *Proc. 4th Int. Conf. on Multiagent Systems (ICMAS)*, 2000.
2. A. Archer, C.H. Papadimitriou, K. Talwar, and É. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. In *Proc. of 14th SODA*, 2003.
3. E. M. Arkin, and R. Hassin. On local search for weighted k-set packing. In the *Proc. ESA '97*, LNCS **1284**, Springer, 1997.
4. V. Bafna, B. Narayana, R. Ravi. Nonoverlapping local alignments (Weighted independent sets of axis-parallel rectangles). *Discr. Applied Math.*, **71**, 41–53, 1996.
5. P. Berman. A $d/2$ Approximation for Maximum Weight Independent Set in d-Claw Free Graphs. *Nordic J. Computing*, **7**(3), pp. 178–184, 2000.
6. P. Berman and P. Krysta. Optimizing misdirection. In *Proc. 14th SODA*, 2003.
7. P. Briest, P. Krysta, and B. Vöcking. Approximation Techniques for Utilitarian Mechanism Design. In *Proc. 37th ACM STOC*, 2005.
8. B. Chandra and M. M. Halldórsson. Greedy local improvement and weighted packing approximation. In *Proc. SODA*, 1999.
9. Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proc. 16th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 548–553, 1999.

10. N. Garg and J. Könemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. In *Proc. 39th IEEE FOCS*, 1998.
11. M.M. Halldórsson. A survey on independent set approximations. In *Proc. APPROX '98*, Springer LNCS 1444, pp. 1–14, 1998.
12. E. Hazan, S. Safra and O. Schwartz. On the Hardness of Approximating k-Dimensional Matching. In *APPROX*, 2003.
13. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discr. Applied Math.*, **6**, pp. 243–254, 1983.
14. P. Krysta. Greedy Approximation via Duality for Packing, Combinatorial Auctions and Routing. In *Proc. of 30th MFCS*, 2005.
15. D. Lehmann, L. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. In *Proc. 1st ACM Conference on Electronic Commerce (EC)*, 1999.
16. K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a Universal Test Suite for Combinatorial Auction Algorithms. In *Proc. 2nd ACM Conference on Electronic Commerce (EC)*, 2000.
17. A. Mu'alem and N. Nisan. Truthful Approximation Mechanisms for Restricted Combinatorial Auctions. In *Proc. 18th AAAI Conf. on Artificial Intelligence*, 2002.
18. P. Raghavan. Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs. *J. Comput. Syst. Sci.*, **37(2)**, 130-143, 1988.
19. P. Raghavan and C.D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, **7**: 365–374, 1987.
20. T. W. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 542–547, 1999.
21. A. Srinivasan. A extension of the Lovász Local Lemma and its applications to integer programming. *In Proc. 7th ACM-SIAM SODA*, 1996.
22. A. Srinivasan. Improved Approximation Guarantees for Packing and Covering Integer Programs, *SIAM J. Computing*, Vol. 29, 648–670, 1999.
23. S. de Vries and R. Vohra. Combinatorial Auctions: A Survey. *INFORMS J. Computing*, **15**(3), pp. 284–309, 2003.
24. E. Zurel and N. Nisan. An Efficient Approximate Allocation Algorithm for Combinatorial Auctions. In the *Proc. EC*, 2001.