# CABRO: Winner Determination Algorithm for Single-unit Combinatorial Auctions

Víctor MUÑOZ and Javier MURILLO
{vmunozs, jmurillo}@eia.udg.es
*University of Girona*

**Abstract.** In this paper we present CABRO, an algorithm for solving the winner determination problem related to single-unit combinatorial auctions. The algorithm is divided in three main phases. The first phase is a pre-processing step with some reduction techniques. The second phase calculates an upper and a lower bound based on a linear programming relaxation in order to delete more bids. Finally, the third phase is a branch and bound depth first search where the linear programming relaxation is used as upper bounding and sorting strategy. Experiments against specific solvers like CASS and general purpose MIP solvers as GLPK and CPLEX show that CABRO is in average the fastest free solver (CPLEX not included), and in some instances drastically faster than any other.

**Keywords.** winner determination problem, combinatorial auctions

## Introduction

Research in combinatorial auctions has grown rapidly in the recent years. This kind of auctions allow bidders to place bids on combinations (also named bundles, collections or packages) of items rather than just on individual items. This fact allows bidders to better express their interests as well as other restrictions such as complementarity or substitutability [9,4]. In a single-unit combinatorial auction the seller (auctioneer) is faced with a set of bids with different prices, and his aim is to select the best subset of them (maximizing the sum of its prices), the *winners*, so that there does not exist any pair of them sharing any item. The problem of selecting the winners in an auction is known as the *Winner Determination Problem* (WDP) and is particularly difficult in combinatorial auctions as it is equivalent to the weighted set-packing problem and the maximum weighted clique problem and therefore *NP-hard* [8,7]. Furthermore, it has been demonstrated that the WDP cannot even be approximated to a ratio of $n^{1-e}$ (any constant factor) in polynomial time, unless *P* = *NP* [9].

Since 1998 there has been a surge of research on designing efficient algorithms for the WDP (see [3,7] for a more extended survey). Given that the problem is *NP-Hard* in the strong sense, any optimal algorithm will be slow on some problem instances. However, in practice, modern search algorithms can optimally solve the WDP in a large variety of practical cases. There exist typically two different ways of solving it. On one hand there exist specific algorithms that have been created exclusively for this purpose, such as CASS [4] and CABOB [10]. On the other hand, the WDP can be modeled as a mixed integer linear problem (MIP) and solved using a generic MIP solver. Due to the efficiency of actual MIP solvers like GLPK (free) and specially CPLEX (commercial), the research community has nowadays mostly converged towards using MIP solvers as the default approach for solving the WDP. There also exist sub-optimal algorithms for solving the winner determination problem that find quick solutions to combinatorial auctions [11,12]. However we will focus only on optimal solutions.

An interesting thing to be noted about the modeling of the WDP as a MIP is that if bids were defined in such a way that they could be accepted partially, the problem would become a linear program (LP) which, unlike MIP, can be solved in polynomial time. We have kept this idea in mind to design a new algorithm, CABRO, which combines LP, search and several reduction techniques to obtain better results than other solvers, even CPLEX in some particular instances.

## 1. Notation

Here we introduce a few notation that is going to be used through this paper. In a single-unit combinatorial auction the auctioneer receives a set of bids $B = \{b_1, ..., b_n\}$, each of them composed by a price $p(b_i)$ and a subset of items $g(b_i)$ of size $n(b_i)$ (such that $n(b_i) = |g(b_i)|$). The complete set of items is $I = \{it_1, ..., it_m\}$.

Useful relations between bids include $b(it_i)$ as the set of bids that contain the item $it_i$, and $C(b_i)$ as the set of bids compatible with bid $b_i$ (i.e. the set of bids that do not contain any item in $g(b_i)$). Additionally, $C(b_i, b_j)$ and $\neg C(b_i, b_j)$ represent whether bids $b_i$ and $b_j$ are compatible or incompatible.

## 2. The Algorithm

CABRO (Combinatorial Auction BRanch and bound Optimizer) is mainly a branch and bound depth-first search algorithm with a specially significative procedure to reduce the size of the input problem. The algorithm is divided in three main phases:

- The first phase performs a fast preprocessing (polynomial time) with the aim of removing as many bids as possible. Bids removed in this phase may be either bids that are surely not in the optimal solution, or bids that surely are.
- The second phase consists in calculating upper and lower bounds for each bid. The upper bound of a bid is computed by formulating a relaxed linear programming problem (LP), while the lower bound is computed generating a solution quickly. This phase may also remove a notable amount of bids.
- The third phase completes the problem by means of search, concretely a branch and bound depth first search. In this phase the two previous phases are used also as heuristic and for pruning.

In some instances it is not necessary to execute all the three phases of the algorithm, for example when the optimal solution is already found before the search phase (which happens more frequently than expected). The algorithm is able to end prematurely either

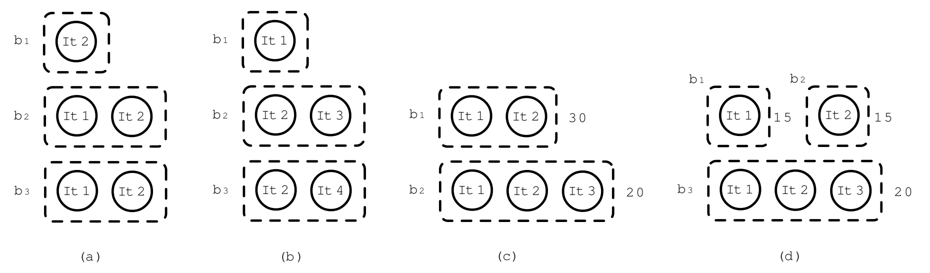

**Figure 1.** Examples of (a) dominated item ($it_1$), (b) solution bid ($b_1$), (c) dominated and (d) 2-dominated bids.

when all of the bids have been removed or when at some point of the execution the global lower bound reaches the global upper bound.

This algorithm also provides anytime performance, giving the possibility to be stopped at any time during the execution and providing the best solution found so far. In the following sections each of the three phases of the algorithm are explained in detail.

## *2.1. First phase: Pre-processing*

This phase uses fast algorithms (with polynomial-time complexity) to reduce the size of the problem by deleting bids and items that either cannot be present at the optimal solution or that surely belong to it. This phase consists of 8 separate strategies (steps), each of them using a different criteria to remove either bids or items.

- **Step 1: Bids with null compatibility.** In this step all the bids that do not have any compatible bid are deleted, except for the bid with the highest price $b_h$. These bids are surely not in the optimal solution since the maximum benefit of a solution containing any of them would be its own price, yet it still does not surpass the price of the bid $b_h$.

- **Step 2: Dependent items.** Items give information about incompatible bids. Still in some cases the information given by an item is already included into another's: the item is *dependent*. Then, the former can be removed without any loss of information. Hence, this step deletes (leaves out of consideration) dependent items. More formally, for each pair of items ($it_1$, $it_2$) such that $b(it_1) \subseteq b(it_2)$, $it_1$ may be deleted from the problem since the information given by $it_1$ is redundant. Figure 1 (a) shows an example of this situation; here item $it_1$ can be deleted given that the information given by $it_1$ ($\neg C(b_2, b_3)$) is already included in the information given by $it_2$ ($\neg C(b_1, b_2)$, $\neg C(b_2, b_3)$ and $\neg C(b_1, b_3)$).

- **Step 3: Bids of the solution.** In some instances there may exist bids such that all of its items are unique (the bid is the only one containing them), and therefore the bid does not have any incompatible bid. In such situations the bid is surely part of the optimal solution.
  This step finds all the bids complying with this condition, adding them to the optimal solution and being removed from the remaining set of bids. Figure 1 (b) shows an example of this situation, where bid $b_1$ is added to the optimal solution

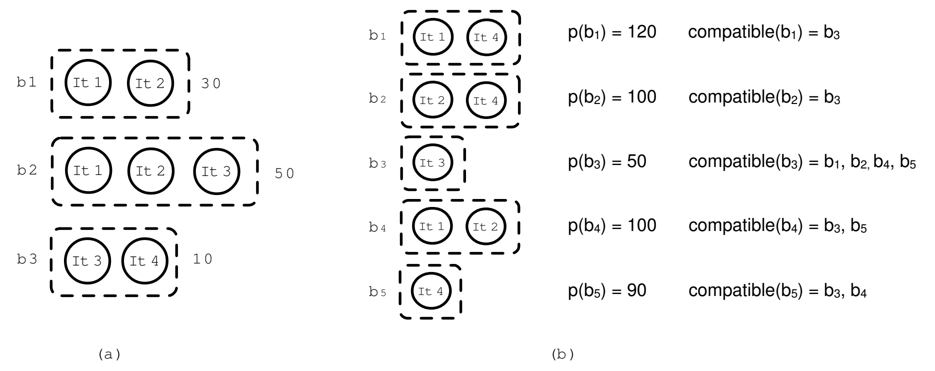

**Figure 2.** Left: Example of pseudo-dominated bid ($b_1$ is pseudo-dominated). Right: Example of compatibility-dominated bid ($b_2$ is compatibility-dominated by $b_1$).

given that its item $i_1$ is unique.

- **Step 4: Dominated bids.** This is the same pre-processing step that CASS [4] and CABOB [10] perform: the elimination of dominated bids. A bid is dominated by another when its set of items includes another bid's items and its price is lower. More formally, for each pair of bids $(b_i, b_j)$ where $g(b_i) \subseteq g(b_j)$ and $p(b_i) \geq p(b_j)$, $b_j$ may be removed as it is never preferable to $b_i$. Figure 1 (c) shows an example of a dominated bid ($b_1$ dominates $b_2$).

- **Step 5: 2-Dominated bids.** This is an extension of the previous technique (also noticed in [9]), checking whether a bid is dominated by a pair of bids. In some cases a bid is not dominated by any single bid separately, but the union of two bids together (joining items and adding prices) may dominate it. Figure 1 (d) shows an example of a 2-dominated bid (the union of $b_1$ and $b_2$ dominates $b_3$). This step can be easily generalized to check *n-dominated* bids. However, for many reasonable distributions, the probability of a bid being dominated by $n$ bids is very low for higher values of $n$, still requiring much more processing (finding all subsets of size $n$), so this generalization is not useful at all for $n > 2$.

- **Step 6: Pseudo-dominated bids.** This step is an even more complex generalization of the dominating techniques. Here we deal again with pairs of bids ($b_i$, $b_j$) such that not all of the items in $b_i$ are contained in $b_j$, but there is one single item $it_k$ not included. In this situation the bid $b_i$ can be removed only if adding to its price the price of its best (highest price) compatible bid containing item $it_k$ is not higher than the price of the bid $b_j$. In such a situation $b_j$ is always preferable to $b_i$ even when taking $b_i$ together with its best compatible bid; therefore $b_i$ does definitely not belong to the optimal solution and might be removed. Figure 2 (a) illustrates this situation: here $b_2$ pseudo-dominates $b_1$ since its price (50) is higher than the sum of bid $b_1$'s price (30) plus the price of its best compatible bid containing the item $it_3$, in this case $b_3$ (10), therefore $b_1$ can be removed.

- **Step 7: Upper and lower bound values.** In this step, fast upper and a lower bounds are assigned to each bid with the aim of deleting bids with its upper bound lower than a *global lower bound* (GLB)[1], since they cannot improve the best solution already found.
  The upper bound $u$ of a bid $b_x$ is calculated according to Equation 1 where $C'(b_x, it_k)$ is the set of compatible bids of $b_x$ including item $it_k$. Roughly, it computes the upper bound of a bid $b_i$ by adding to its price the best possible prices of the bids containing the items not included in $g(b_i)$.
  After that, the lower bound of the bids is then calculated constructing a solution of a bid by iteratively attempting to add all of its compatible bids to the solution. Its compatible bids are ordered in descending order according to the upper bound previously calculated. All the solutions obtained with this algorithm are valid solutions and update the GLB accordingly. Note that GLB actually stores the best solution to the problem found so far (although it may not be the optimal one), therefore it can be returned immediately if the user decides to stop de execution, thus providing anytime performance.

$$u(b_x) = p(b_x) + \sum_{\forall i \notin g(b_x)} \max_{\forall j \in C'(b_x, it_k)} \frac{p(b_j)}{n(b_j)} \tag{1}$$

- **Step 8: Compatibility-Dominated bids.** This step is another generalization of dominated bids. A bid $b_i$ is compatibility-dominated by another bid $b_j$ if the set of compatible bids of $b_i$ is a subset of the set of compatible bids of $b_j$ and its price is lower. More formally, for each pair of bids $(b_i, b_j)$ where $C(b_i) \subseteq C(b_j)$ and $p(b_i) \geq p(b_j)$, $b_j$ may be removed as it is never preferable to $b_i$. Figure 2 (b) shows an example where $b_2$ is not dominated by $b_1$ but it is compatibility-dominated.

Once all of these steps have been executed, since the problem has changed, it may be the case that some bids and items previously undeleted can now be removed. For example the deletion of a bid may cause the appearance of dominated items and vice-versa. Therefore phase 1 is repeated until it does not remove any more bid or item.

## 2.2. *Second phase: Upper and Lower Bounding*

In the second phase, the algorithm calculates improved upper and lower bounds for each bid. In order to compute the upper bound for a given bid $b_i$, a relaxed linear programming (LP) problem is formulated. This relaxed formulation defines the bids in such a way that they can be accepted partially (a real number in the interval [0, 1]), therefore it can be solved using the well-known simplex algorithm [2], which solves most of the instances in polynomial-time. The relaxed version does not contains neither the current bid $b_i$ nor none of the bids with items included in $b_i$ (i.e. its incompatible bids). Adding the price of the bid $b_i$ to the solution of the relaxed LP problem gives a new upper bound that is usually much more precise than the one obtained in step 7 of phase 1.

This step firstly performs an ordering of the bids according to the upper bound value calculated in step 7 of phase 1 in ascending order. Then the process of calculating new

[1]The global lower bound (GLB) is the best (maximum) lower bound found, associated to a valid solution.

upper bounds using the simplex method starts with the bid with the lower upper bound, and each time a bid's upper bound is lower that the GLB, it is deleted, thus decreasing the size of the subsequent bids' simplex. Note that the chosen ordering, beginning with the "worst" bids, may seem inappropriate, but this is in fact a good strategy since the worst bids' upper bounds are usually much faster to compute than the "best", hence we quickly obtain accurate upper and lower bounds that may allow to remove lots of bids rapidly, thus decreasing the size of the problem and making "best" bids also faster to be computed. This fact has been verified experimentally.

Regarding the lower bound for each bid $b_i$, it is computed using the values returned by the LP solver, and updates the GLB accordingly. The solution is constructed by firstly considering any value greater than 0.5 to be actually 1; that is, part of the (partial) solution. This assumption is not inconsistent (it does not produce solutions containing incompatible bids) because compatible bids are restricted to sum at most 1, therefore two incompatible bids cannot have both values larger than 0.5. After that, the remaining bids (with values smaller or equal to 0.5) are attempted to be put into the solution in descending order. Of course if the solution of the LP was integer this process is not required, as it is the optimal solution for that bid.

## 2.3. *Third phase: Search*

The third phase (*iCabro*) performs a branch-and-bound depth-first search with the remaining bids of the previous phases ($L$). The full algorithm can be seen in Figure 3. The value of the best solution found so far (GLB) is stored in the global variable *bSolution*. Initially *bSolution*=0, and the search starts by calling *iCabro*($L$,0).

```
procedure iCabro($L$,cSolution)
    for each element $b$ of $L$
        $L2 \leftarrow L \bigcap compatible(b)$
        $cSolution2 \leftarrow cSolution \bigcup b$
        $LPSol \leftarrow simplex(cSolution2)$
        if $LPSol$ is integer then
            $cSolution2 \leftarrow cSolution2 \bigcup LPSol$
            $L2 \leftarrow \emptyset$
        end-if
        if $v(LPSol) > v(bSolution)$ then
            if $v(cSolution2) > v(bSolution)$ then
                $bSolution \leftarrow cSolution2$
            end-if
            if $L2$ is not empty then
                $sort(L2)$
                $iCabro(L2, cSolution2)$
            end-if
        end-if
    end-for
end-procedure
```

**Figure 3.** Pseudo-code algorithm of iCabro procedure

The *iCabro* procedure processes the incoming list of bids $L$ performing the following steps:

- The algorithm begins getting the first bid $b$ of the list $L$ (recall that $L$ is sorted according to the upper bound computed in phase 2). A new list $L2$ is created as the intersection between $L$ and $C(b)$ (compatible bids of $b$). In deeper nodes (as it is a recursive function) the set $L2$ represents the compatible bids with the current solution.
- After that, the algorithm formulates and solves the Linear Programming (LP) problem related to the current solution. If the result of the LP problem is integer then the algorithm finishes (prunes) the current branch, as the optimal solution of the branch has been found.
- At line 10 the algorithm verifies if the upper bound of the current solution is greater than the GLB (the best solution found so far). If this is the case the search continues through this branch updating the best current solution if necessary. Otherwise, the branch is pruned.
- At line 14 the algorithm verifies that the $L2$ set is not empty, given that if it is empty then it means that the current solution does not have any more compatible bids and consequently the branch is finished. Alternatively, if this condition does not apply, then the following action is to sort the list $L2$ according to the upper bound of each bid, in order to perform a recursive call to *iCabro* with the list $L2$.

## 3. Results

To evaluate the CABRO algorithm we have compared it against both specific algorithms and general purpose MIP solvers. We have chosen CASS for the specific solver instead of CABOB because although their authors claim that it outperforms CASS, there is no implementation of it available publicly. For the MIP solver, both GLPK (free) and CPLEX 10.1 (commercial) have been tested.

Test examples have been generated using the popular benchmark for combinatorial auctions CATS (Combinatorial Auctions Test Suite) [6], which creates realistic auction instances. Since its first release in 2000, CATS has became the standard benchmark to evaluate and compare WDP algorithms [10,7]. The CATS suite generates instances following five real-world situations and seven previously published distributions by different authors (called legacy). Given a required number of goods and bids, all the distributions select which goods to include in each bid uniformly at randomly without replacement.

For most of the five real-world distributions a graph is generated representing adjacency relationships between goods, and it is used to derive complementarity properties between goods and substitutability properties for bids. Some of the real-world situations concern complementarity based on adjacency in (physical or conceptual) space, while the remaining concern complementarity based on correlation time. The characteristics of each distribution are as follows:

- Paths (PATHS). This distribution models shipping, rail and bandwidth auctions. Goods are represented as edges in a nearly planar graph, with agents submitting a set of bids for paths connecting two nodes.
- Arbitrary (ARB). In this distribution the planarity assumption is relaxed from the previous one in order to model arbitrary complementarities between discrete goods such as electronics parts or colectables.

- Matching (MAT). This distribution concerns the matching of time-slots for a fixed number of different goods; this case applies to airline take-off and landing rights auctions.
- Scheduling (SCH). This distribution generates bids for a distributed job-shop scheduling domain, and also its application to power generation auctions.

The seven legacy distributions are the following:

- L1, the *Random* distribution from [9], chooses a number of items uniformly at random from [1,$m$], and assigns the bid a price drawn uniformly from [0, 1].
- L2, the *Weighted Random* distribution from [9], chooses a number of items $g$ uniformly at random from [1, $m$] and assigns a price drawn uniformly from [0, $g$].
- L3, the *Uniform* distribution from [9], sets the number of items to some constant $c$ and draws the price offer from [0, 1].
- L4, the *Decay* distribution from [9] starts with a bundle size of 1, and increments the bundle size until a uniform random drawn from [0, 1] exceeds a parameter $\alpha$.
- L5, the *Normal* distribution from [12], draws both the number of items and the price offer from normal distributions.
- L6, the *Exponential* distribution from [4], requests $g$ items with probability $C_{e^{-g/q}}$, and assigns a price offer drawn uniformly at random from [0.5$g$, 1.5$g$].
- L7, the *Binomial* distribution from [4], gives each item an independent probability of $p$ of being included in a bundle, and assigns a price offer drawn uniformly at random from [0.5$g$, 1.5$g$] where $g$ is the number of items selected.

We have also created a new distribution called transports (TRANS) based on a real problem: the road transportation problem. The problem roughly consists of finding the best assignment of available drivers to a set of requested services given a cost function and subject to a set of constraints (see [1] for more details). To model this problem as an auction the bids represent journeys (a set of services) associated with a driver, therefore its items represent the services performed as well as the driver used. Note that the original problem consists in minimizing the final cost of doing all the services, while an auction is concerned on maximizing. Therefore, the costs associated to the bids are appropriately transformed so that the maximized solution corresponds to the real (minimizing) solution.

We have generated 100 instances of each distribution with different amounts of bids and items. Each instance has been solved using CABRO, CASS, GLPK 4.9 and CPLEX 10.1 with a timeout of 300 seconds. The first three methods have been run in a 2.4GHz Pentium IV with 2Gb of RAM running under Windows XP SP2, while CPLEX has been run on a 3.2GHz Dual-Core Intel Xeon 5060 machine with 2 Gb of RAM running under GNU/Linux 2.6.

Figure 4 (left) shows the average execution time (in seconds) required for each method to solve all the instances of all the distributions. Here we can observe that CPLEX is in average the fastest solver since it solves all the instances (1167 auctions) in considerably less time than the other solvers. Recall that the machine used for CPLEX is considerably faster than the one used for the others; however, we believe that the results on equal machines would not change significantly. Yet CABRO spends less time than the free solvers GLPK and CASS.

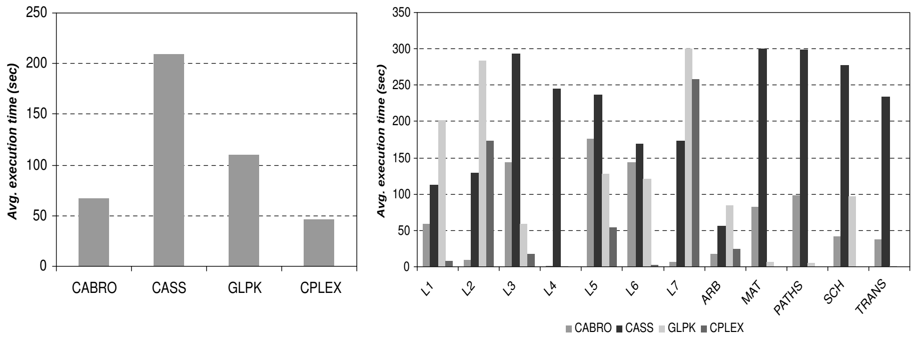

**Figure 4.** Left: Global comparative. Right: Comparative over distributions.

Figure 4 (right) shows the results in each of the distributions comparing the average time required (in seconds) to solve all the instances of each distribution with the four methods. Here we can observe that in two distributions (L2 and L7) CABRO is clearly the best algorithm and in other one (ARB) is also the best solver but CPLEX is very close. In the rest of distributions CPLEX is the best. Regarding the free solvers, GLPK is cleary the best solver in L3, L5, TRANS, MAT and PATHS while CABRO is cleary the best in L1, L2,L7, ARB and SCH. CASS is only rather competitive in L1, L2, L7 and ARB distributions.

| | $CABRO$ | | | $CASS$ | | | $GLPK$ | | | $CPLEX$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $F$ | $\neg F$ | % | $F$ | $\neg F$ | % | $F$ | $\neg F$ | % | $F$ | $\neg F$ | % |
| $L1$ | 80 | 15 | 84.2 | 67 | 28 | 70.5 | 39 | 56 | 41.1 | 95 | 0 | 100.0 |
| $L2$ | 100 | 0 | 100.0 | 90 | 10 | 90.0 | 7 | 93 | 7.0 | 50 | 50 | 50.0 |
| $L3$ | 54 | 46 | 54.0 | 3 | 97 | 3.0 | 84 | 16 | 84.0 | 98 | 2 | 98.0 |
| $L4$ | 100 | 0 | 100.0 | 22 | 78 | 22.0 | 100 | 0 | 100.0 | 100 | 0 | 100.0 |
| $L5$ | 44 | 56 | 44.0 | 23 | 77 | 23.0 | 61 | 39 | 61.0 | 90 | 10 | 90.0 |
| $L6$ | 53 | 47 | 53.0 | 46 | 54 | 46.0 | 70 | 30 | 70.0 | 100 | 0 | 100.0 |
| $L7$ | 100 | 0 | 100.0 | 68 | 32 | 68.0 | 0 | 100 | 0.0 | 15 | 85 | 15.0 |
| $ARB$ | 96 | 4 | 96.0 | 86 | 14 | 86.0 | 81 | 19 | 81.0 | 99 | 1 | 99.0 |
| $MAT$ | 81 | 19 | 81.0 | 0 | 100 | 0.0 | 100 | 0 | 100.0 | 100 | 0 | 100.0 |
| $PATHS$ | 55 | 17 | 76.4 | 1 | 71 | 1.4 | 72 | 0 | 100.0 | 72 | 0 | 100.0 |
| $SCH$ | 98 | 2 | 98.0 | 9 | 91 | 9.0 | 84 | 16 | 84.0 | 100 | 0 | 100.0 |
| $TRANS$ | 94 | 6 | 94.0 | 24 | 76 | 24.0 | 100 | 0 | 100.0 | 100 | 0 | 100.0 |
| $TOTAL$ | 955 | 212 | 81.8 | 439 | 728 | 37.6 | 798 | 369 | 68.4 | 1019 | 148 | 87.3 |

**Table 1.** Finished auctions ($F$), not finished auctions ($\neg F$) and percentage of finished auctions ($\%$) before the timeout.

Table 1 shows the number of auctions finished ($F$), the number of auctions not finished ($\neg F$) and the percentage of finished auctions ($\%$) before the timeout, for each method and each distribution. The results are similar to the execution time results, with CPLEX being the best method in absolute results, as it solves up to 1019 instances (87%). However, there is not any method that can be claimed to be the best, since it depends on the kind of data that the auction is processing. Particularly, CABRO performs better for the weighted random and binomial distributions, solving 100% of the instances, while CPLEX only solves 15% in L7 and 50% in L2.

## 4. Conclusions

In this paper an algorithm for solving combinatorial auction problems has been presented. It uses many reduction techniques, together with an heuristic function based on linear programming techniques that provides more pruning. We have compared its performance with other existing algorithms obtaining encouraging results, particularly for weighted random and binomial distributions. In the future, we plan to hugely improve the algorithm with new reduction strategies as well as integrate it in the search phase. We also plan to improve the upper bound function used in the first phase and to test other sorting criteria to obtain better lower bounds. Also, we would focus on understanding the different characteristics of the domains and its influence in the solution time, and theoretically characterize domains where CABRO outperforms CPLEX and work in the domains where it does not. We expect that these changes would significantly improve the algorithm performance. Another interesting point would be to extend this algorithm to deal also with multi-unit combinatorial auctions, as there are not many specific algorithms for this kind of auctions. Finally, we will study another criteria to evaluate the algorithm as for example the anytime behavior and the memory consumption, as it is known to be a drawback of MIP solvers.

## References

[1] Javier Murillo and Beatriz Lopez, ‘An empirical study of planning and scheduling interactions in the road passenger transportation domain’, *Proceedings of PlanSIG 2006, 2006.*, 129–136, (2006).

[2] George B. Dantzig, ‘The simplex method’, *RAND Corp*, (1956).

[3] Sven de Vries and Rakesh V. Vohra, ‘Combinatorial auctions: A survey’, *INFORMS Journal on Computing*, (3), 284–309, (2003).

[4] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham, ‘Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches’, in *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 548–553, (1999).

[5] Holger H. Hoos and Craig Boutilier, ‘Solving combinatorial auctions using stochastic local search’, in *AAAI/IAAI*, pp. 22–29, (2000).

[6] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham, ‘Towards a universal test suite for combinatorial auction algorithms’, in *ACM Conference on Electronic Commerce*, pp. 66–76, (2000).

[7] Yoav Shoham Peter Cramton and Richard Steinberg, *Combinatorial Auctions*, MIT Press, 2006.

[8] M. H. Rothkopf, A. Pekec, and R. M. Harstad, ‘Computationally manageable combinatorial auctions’, Technical Report 95-09, (19, 1995).

[9] Tuomas Sandholm, ‘Algorithm for optimal winner determination in combinatorial auctions’, *Artificial Intelligence*, **135**(1-2), 1–54, (2002).

[10] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine, ‘CABOB: A fast optimal algorithm for combinatorial auctions’, in *IJCAI*, pp. 1102–1108, (2001).

[11] D. Schuurmans, F. Southey, and R. C. Holte, ‘The Exponential Subgradient Algorithm for Heuristic Boolean Programming’, in *IJCAI*, (2001).

[12] Holger H. Hoos, and Craig Boutilier, ‘Solving Combinatorial Auctions Using Stochastic Local Search’, in *AAAI/IAAI*, pp. 22-29, (2000).