Report: a lagrangian heuristic and a branch-and-cut approach to the combinatorial auction problem

Vitor Mazal Krauss

November 2021

1 Combinatorial Auctions

In this section, we present the *combinatorial auction problem* and formulate it as a *set packing problem*.

The combinatorial auction problem is described as follows: a supplir has a set I of exactly m *items* to auction. Bidders propose a total of n *bids*. For $1 \leq i \leq n$, each bid b_i *covers* a subset $B_i \subseteq I$ of items, and bid b_i has an associated *profit* w_i. The goal of the combinatorial auction problem is to select a set of bids in a way to maximize the sum of the profits of selected bids, and subject to the constraints that each item must be covered by at most one of the selected bids.

Let $A \in \mathbb{B}^{n \times m}, A = [a_{ij}]$ be a binary matrix such that $a_{ij} = 1$ if bid b_i covers item j, i.e $j \in B_i$, and $x_{ij} = 0$ otherwise, for each $1 \leq i \leq n$ and $1 \leq j \leq m$.

Let us consider $x \in \mathbb{B}^n$, where, for each $1 \leq i \leq n$, x_i is a binary variable such that $x_i = 1$ if bid b_i is selected and $x_i = 0$ otherwise. Then, a 0/1 integer programming formulation for the combinatorial auction problem is given by:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} w_i x_i & \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq 1 & j = 1, \ldots, m \\ & x_i \in \{0, 1\} & i = 1, \ldots, n \end{aligned} \tag{1}$$

Note that (1) models the combinatorial auction problem as the well-known set packing problem [3].

Since the set packing problem is known to be $\mathcal{NP}$-complete [3], we conclude the following result.

Theorem 1 *The combinatorial auction problem is $\mathcal{NP}$-complete.*

Given the computational difficulty of the combinatorial auction problem, many different exact and non-exact algorithms have been proposed to solve it. For a more detailed discussion and further references, see [3].

In this work, we propose two different approaches to solving the combinatorial auction problem. In Section 2 we present valid inequalities for the combinatorial auction problem based on the idea of *conflict graphs*. In Section 3 we present a non-exact heuristic method based on lagrangian relaxation. In Section 4 we present an exact branch-and-cut method. The separation problem solved on each step of this branch-and-cut algorithm is given by a *maximum weight clique problem* on the conflict graph. In Section 5 we present computational results for the two methods on instances of the well-established *CATS* test suite.

2 Valid Inequalities

In integer programming, including (strong) valid inequalities to an integer program (IP) are a well-established method for strengthening a given IP formulation [6]. In general integer programming, and more specifically for especialy hard integer programming problems, the inclusion of strong valid inequalities plays an important, and often crucial, role in the solution of the IP problem, in some cases even making the difference between success and failure.

In this section we present a method for finding valid inequalities for the combinatorial auction problems.

Consider an instance of the combinatorial auction problem with m items and n bids, where each bid b_i covers a set $B_i \subseteq I$ of bids and has associated profit w_i.

Note from the IP formulation given in (1) that, for each item $j \in I$, at most one of the selected bids can cover this item. Thus, for any two bids b_ℓ and b_k, if $B_\ell \cap B_k \neq \emptyset$, then at most one of b_ℓ or b_k can be selected in any feasible solution. And more generally, if bids $b_{i_1}, \ldots, b_{i_C}$ are such that $B_\ell \cap B_k \neq \emptyset$ for all $1 \leq \ell, k \leq C$, then at most one of $b_{i_1}, \ldots, b_{i_k}$ can be selected in any feasible solution.

With this idea in mind, we now define the *conflict graph* associated to this instance of the combinatorial auction problem.

We define the conflict graph $G = (V, E)$ where:

- $V = \{b_1, \ldots, b_n\}$, i.e, G has a vertex v_i for each bid b_i, for $1 \leq i \leq n$.
- $(v_\ell, v_k) \in E$ if and only if $B_\ell \cap B_k \neq \emptyset$.

From this definition, it follows that any edge $(v_\ell, v_k) \in E$ defines a (redundant) valid inequality $x_\ell + x_k \leq 1$; any clique $C = \{v_{i_1}, \ldots, v_{i_C}\}$ in G defines a valid inequality $x_{i_1} + \cdots + x_{i_C} \leq 1$, since $B_\ell \cap B_k \neq \emptyset$ for all $1 \leq \ell, k \leq C$; and any maximal clique $C = \{v_{i_1}, \ldots, v_{i_C}\}$ in G defines a strong valid inequality $x_{i_1} + \cdots + x_{i_C} \leq 1$.

We conclude that given an instance of the combinatorial auction problem, the formulation presented in (1) can be strengthened by finding cliques and (even better) maximal cliques in the conflict graph G associated to this instance. In the case of cutting plane algorithms, such as the branch-and-bound algorithm presented in Section 4, the conflict graph can also be used in the definition of a separation problem which, in each node of the enumeration tree, possibly cuts off a fractional solution by including a valid inequality to the problems original formulation. This is presented in detail in Section 4.

3 Lagrangian Heuristic

In this section we present a non-exact heuristic method based on lagrangian relaxation for the combinatorial auction problem. From this point forward, this algorithm will be refered to as *LRH*.

Let us recall from Section 2 that the integer programming formulation of the combinatorial auction problem can be strengthened by including valid inequalities by finding cliques in the conflict graph associated with each instance.

Given an instance of the combinatorial auction problem, let $C = \{C_1, \ldots, C_k\}$ be a set of cliques in the associated conflict graph. We define a binary matrix $\mathbb{B}^{k \times n}, B = [b_{ci}]$, where $b_{ci} = 1$ if vertex v_i is in the clique C_c, and $x_{ci} = 0$ otherwise.

Then, we obtain the strengthened formulation for the combinatorial auction problem:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq 1 \quad j = 1, \ldots, m \\ & \sum_{i=1}^{n} b_{ci} x_i \leq 1 \quad c = 1, \ldots, k \\ & x_i \in \{0, 1\} \qquad i = 1, \ldots, n \end{aligned} \tag{2}$$

From the formulation presented in (2), we now present the procedure we applied in order to arrive at a lagrangian relaxation formulation for the combinatorial auction problem. To do this, we dualize the two kinds of constraints in (2):

- For each item $j = 1, \ldots, m$, we dualize the item cover constraint $\sum_{i=1}^{n} a_{ij} x_i \leq 1$.
- For each clique $C_c \in C$ we dualize the bid clique constraint $\sum_{i=1}^{n} b_{ci} x_i \leq 1$.

By performing these dualizations and including each one as a term in the objetive function, we arrive at the following associated lagrangian sub-problem for the combinatorial auction problem:

$$\begin{aligned}\max Z(u,v) &= \sum_{i=1}^{n} w_i x_i + \sum_{j=1}^{m} u_j \cdot \Big[1 - \sum_{i=1}^{n} a_{ij} x_i\Big] + \sum_{c=1}^{k} v_c \cdot \Big[1 - \sum_{i=1}^{n} b_{ci} x_i\Big] \\ &= \sum_{j=1}^{m} u_j + \sum_{c=1}^{k} v_c + \sum_{i=1}^{n} x_i \cdot \Big[w_i - \sum_{j=1}^{m} u_j a_{ij} - \sum_{c=1}^{k} v_c b_{ci}\Big]\end{aligned} \tag{3}$$

Note from the expression in (3) that, for fixed u and v, the lagrangian sub-problem $\max Z(u,v)$ can be solved in linear time.

For each $i = 1, \ldots, n$, let $\alpha_i = w_i - \sum_{j=1}^{m} u_j a_{ij} - \sum_{c=1}^{k} v_c b_{ci}$. Then, there is an optimal solution x^{LR} of $\max Z(u,v)$ satisfying:

- $\alpha_i < 0 \longrightarrow x_i^{LR} = 0$
- $\alpha_i > 0 \longrightarrow x_i^{LR} = 1$

At each step of the LRH algorithm, we begin by solving the lagrangian sub-problem $\max Z(u,v)$ for the current values of u and v by the linear procedure described above. Once a solution x^{LR} to the lagrangian sub-problem has been found, we apply a heuristic algorithm in order to obtain a feasbible solution to the original IP. This feasible solution then provides a lower bound on the value of the objetive function of the original IP.

Algorithm 1 FSB Heuristic

Require: x^{LR} solution to lagrangian sub-problem
  $T = \{j \in I : \sum_{i=1}^{n} a_{ij} x_i^{LR} \geq 2\}$
  $x \leftarrow x^{LR}$
  while $T \neq \emptyset$ **do**
    $\ell = \min_{1 \leq i \leq n} \left\{ \frac{w_i}{|T \cap B_i|} \right\}$
    $x_\ell \leftarrow 0$
    $T \leftarrow \{j \in I : \sum_{i=1}^{n} a_{ij} x_i \geq 2\}$
  end while
  return x

Also, note from expression (3) that a subgradient vector $\partial_u Z(u,v)$ of $Z(u,v)$ with respect to u is given by:

$$\partial_u Z(u,v) = \vec{1} - A^T \vec{x} \tag{4}$$

i.e, for each coordinate u_j of $\vec{u}$:

$$\partial_{u_j} Z(u,v) = 1 - \sum_{i=1}^{n} a_{ij} x_i \tag{5}$$

And that a subgradient vector $\partial_v Z(u,v)$ of $Z(u,v)$ with respect to v is given by:

$$\partial_v Z(u,v) = \vec{1} - B\vec{x} \tag{6}$$

i.e, for each coordinate v_j of $\vec{v}$:

$$\partial_{v_j} Z(u,v) = 1 - \sum_{i=1}^{n} b_{ci} x_i \tag{7}$$

With these two subgradients $\partial_u Z(u,v)$ and $\partial_v Z(u,v)$, we define a subgradient $\partial Z(u,v)$ of $Z(u,v)$ with respect to (u,v) as $\partial Z(u,v) = (\partial_u Z(u,v), \partial_v Z(u,v))$.

We note that, for any (u,v), the value $\max Z(u,v)$ is an upper bound on the value of the objetive function of the original IP. Hence $\min_{u,v\geq 0}\{\max Z(u,v)\}$ is the best–as in tightest– upper bound on the value of the objetive function of the original IP that one can obtain from (3). The problem of finding $\min_{u,v\geq 0}\{\max Z(u,v)\}$ is called the *lagrangian dual*.

By knowing $\partial Z(u,v)$, which is a subgradient vector of $Z(u,v)$, we can perform a subgradient method in the hopes of finding a minimum–possibly local– of $Z(u,v)$.

Now we present the LRH algorithm. In short, the LRH begins by specifing initial values for u^0, v^0 and by initializing the value of $\overline{Z}$, an upper bound on the value of the objetive function of the original IP, and $\underline{Z}$, a lower bound, and x^*, the best known feasible solution.

The values of u^0 and v^0 are given by:

$$u_j^0 = \frac{\sum_{i=1}^{n} \left[a_{ij} c_i \big(\sum_{k=1}^{m} a_{ik} \big)^{-1} \right]}{\sum_{i=1}^{n} a_{ij}}$$

and

$$v_j^0 = 0$$

And initially $x^* \leftarrow \vec{0}$.

At step t of the LRH algorithm, we begin by solving–in linear time–the lagrangian sub-problem $\max Z(u^t, v^t)$ with solution x_{LR}. We denote the value of $Z(u^t, v^t)$ at point x_{LR} by $Z(u^t, v^t, x^{LR})$. This value provides an upper bound on the value of the objetive function of the original IP. We then apply the FSB Heuristic on x^{LR} described in Algorithm 2 in order to find a feasible solution x' to the original IP and, hence, to find a lower bound on the value of its objective function. We then compute the subgradient $\partial Z(u^t, v^t)$ and set

$$u_j^{t+1} = \max\left\{0, u_j^t - F \cdot \frac{\left(\overline{Z} - Z(u_t, v_t, x_t^{LR})\right)}{||(u^t, v^t)||^2} \cdot \left(1 - \sum_{i=1}^{n} a_{ij} x_i^{LR}\right)\right\}$$

$$v_j^{t+1} = \max\left\{0, v_j^t - F \cdot \frac{\left(\overline{Z} - Z(u_t, v_t, x_t^{LR})\right)}{||(u^t, v^t)||^2} \cdot \left(1 - \sum_{i=1}^{n} b_{ci} x_i^{LR}\right)\right\}$$

Finally, we update the values of $\overline{Z}, \underline{Z}$ and x^*. The LRH also keeps track of an integer variable k, which counts the number of consecutive iterations with no improvement on x^*. The algorithm continues until k reaches a specified value.

Algorithm 2 LRH

Require: $A \in \mathbb{B}^{n \times m}$
Require: $B \in \mathbb{B}^{c \times n}$
Require: $\vec{w} \in \mathbb{R}^n$
  Initialize:
    u^0, v^0, x^*
  while $k \leq k_{\max}$ **do**
    Solve $Z(u^t, v^t)$ with solution x^{LR}
    $x' \leftarrow$ FSB Heuristic on x^{LR}
    Compute $\partial Z(u^t, v^t)$
    Compute u^{t+1}
    Compute v^{t+1}
    Update $\overline{Z}, \underline{Z}$ and x^*
  end while
  return $\overline{Z}, \underline{Z}$ and x^*

4 Branch-and-Cut

In this section we present an exact branch-and-cut algorithm for the combinatorial auction problem.

In short, the branch-and-cut algorithm follows a similar procedure of a branch-and-bound algorithm, with the addition that, at each node in the enumeration tree, a *separation problem* is solved and and a valid inequality (cutting plane) is included to the IP formulation in order to strengthen/tighten the IP formulation. A detailed explanation of both the branch-and-bound and the branch-and-cut algorithm is presented in [6].

At each node in the branch-and-cut enumeration tree, a solution x^{LP} to the linear programming relaxation of the current sub-problem is found. Since the linear programming relaxation is defined by dropping the integer constraints on the decision variables, x^{LP} may have fractional(non-integer) values. In the case that x^{LP} is fractional, we aim to find a valid inequality to the IP that "cuts off" this fractional solution, i.e, we aim to find a valid inequality which is violated by x^{LP}. We call the problem of finding such an inequality as the *separation problem*. If such an inequality is found, we then include this inequality as a constraint in the IP, hence strengthening the original IP formulation.

The branch-and-cut algorithm for the combinatorial auction problem presented here is based on the idea of the conflict graph, as presented in Section 2.

We now define the separation problem solved at each node of the branch-and-cut algorithm for the combinatorial auction problem.

Given an instance of the combinatorial auction problem, we begin by constructing the conflict graph associated to this instance, as defined in Section 2. For the separation problem, we shall make use of a weighted version of this clonflict graph. At each node in the branch-and-cut enumeration tree the vertex weights will change, but the vertex set and edge set of the conflict graph are constant and, hence, need only be constructed once.

4.1 Separation Problem

On any node of the branch-and-cut enumeration tree, let x^{LP} be the solution to the linear programming relaxation associated with that node. We define a weighted conflict graph $G(x^{LP})$ such that, for $i = 1, \ldots, m$, each vertex v_i has associated weight x_i^{LP}. Then, the separation problem for this node is to find a maximum weight clique on this weighted conflict graph. From Section 2 we know that any clique in the conflict graph defines a valid inequality for the problem. Let $C = \{i_1, \ldots, i_k\}$ be a clique on the weighted clique graph and $W = \sum_{\ell=1}^{k} x_{i_\ell}^{LP}$ be this clique total weight. Then, if $W > 1$, the inequality $x_{i_1} + \cdots + x_{i_k} \leq 1$ "cuts off" the current LP solution x^{LP}. Hence this inequality is added to the problems formulation.

Algorithm 3 Separation Problem

Require: x^{LP} solution of LP relaxation
Require: $G = (V, E)$ unweighted conflict graph
  Initialize:
    $G(x^{LP})$ weighted conflict graph
  Find maximum weight clique $C = \{i_1, \ldots, i_k\}$ on $G(x^{LP})$
  if W **then** ¿ 1
    Add $x_{i_1} + \cdots + x_{i_k} \leq 1$ to the formulation
  end if

In this work, in order to find a maximum weight clique on the conflict graph, we make use of the *FastWClq* algorithm presented in [2]. This is a non-exact heuristic algorithm which aims to find a maximum weight clique in a weighted graph. In this work, we implemented the FastWClq algorithm in the Julia programming language [1]. Some results of our implementation on reasonably large network graphs are presented in the Section 5.

In short, the branch-and-cut algorithm follows a procedure similar to a branch-and bound algorithm, with the addition that at each node of the enumeration tree a separation problem is solved and possibly a valid inequality is added to strengthen the IP formulation. At each node, the algorithm possibly

generates two new nodes by setting the value of a certain variable x_i to be $x_i = 0$ on one branch and $x_i = 1$ on the other. Although this branching strategy leads to a state space of size 2^n–i.e the number of possible 0/1 sequences of size n– the number of nodes actually explored by the branch-and-cut algorithm is possibly–and hopefully–reduced–sometimes drastically–by determining upper and lower bounds on the objective function value during the enumeration, and also by including the valid inequalities found by solving the separation problem.

5 Results

In this section, we present some computational results of the proposed algorithms.

5.1 FastWClq

In Table 1 we present the results of our implementation in Julia of the FastWClq [2] algorithm for finding a maximum weight clique in a graph.

The instances here considered are graphs from the *Network Repository* [5]. Since the graphs provided by the Netwrok Repository are originally unweighted, in order to obtain a weighted graph, we associate with each node i of the graph the weight $(i \mod 200)+1$, exactly as suggested in [2]. The instances considered here are a part of the ones considered in [2].

In Table 1, each line corresponds to a single instance and its solution, column *Instance* refers to the Network Repository instance name, column *# nodes* refers to the number of the nodes in the graph, column *# edges* refers to the number of edges in the graph, column $z_{FastWClq}$ refers to the weight of clique of largest weight found by the FastWClq algorithm, column $t_{FastWClq}$ refers to the time, in seconds, taken by the algorithm to finish, column $z_{MaxWClq}$ refers to the optimal or best known maximum weight clique–as presented in [2]– value for that instance and the column *% gap* refers to the percentage difference between the found solution and the optimal/best known solution.

Instance	# nodes	# edges	$z_{FastWClq}$	$t_{FastWClq}$	$z_{MaxWClq}$	% gap
bio-dmela	7393	25569	805	218ms	805	0
bio-yeast	1458	1948	578	665 μ s	629	8.1
ca-AstroPh	17903	196972	5320	8.5 s	5338	0.33
ca-CondMat	21363	91286	2887	715 ms	2887	0
ca-Erdos992	6100	7515	958	4.3 ms	958	0
ca-citeseer	227320	814134	8838	5.4 s	8838	0
ca-dblp-2010	226413	716460	7456	3.9 s	7575	1.5
ca-dblp-2012	317080	1049866	14108	7 s	14108	0
ia-fb-messages	1266	6451	791	61 ms	791	0
ia-reality	6809	7680	333	30 ms	374	10
rec-amazon	91813	125704	888	26 ms	942	5.7
socfb-Berkeley13	22900	852419	3565	53 s	4906	27
socfb-CMU	6621	249959	3796	19s	4141	8.3
socfb-Duke14	9885	506437	3694	21s	3694	0

Table 1: Results of our implementation of the FastWClq algorithm for instances of the Network Repository.

5.2 Lagrangian Heuristic

For the lagrangian heuristic and the branch-and-cut for the combinatorial auction problem, we consider instances from the Combinatorial Auction Test Suite, or simply *CATS* [4]. The CATS is a standard suite for generating instances of the combinatorial auction problem. Besides providing a wide variety of distributions and instance profiles, the CATS addresses many important aspects of generating realistic combinatorial auction instances, such as the distribution and relationships between items, bids and prices.

In Tables 2 and 3 we present results of the lagrangian heuristic algorithm presented in Section 3 for instances of the CATS Suite. In Table 2 we consider instances with smaller number of items and bids, while in Table 3 we consider instances with larger number of items and bids.

In Tables 2 and 3, each line corresponds to a single instance and its solution, column *# bids* refers to the number of bids of the instance, column *# items* refers to the number of items of the instance, the column $\overline{z}_{LR}$ refers to the lowest upper bound found by lagrangian relaxation, column $\underline{z}_{LR}$ refers to the largest lower bound found by primal solutions during the lagrangian heuristic algorithm, column t_{LR} refers to the time, in seconds, taken by the algorithm to finish, column z_{gurobi} refers to the optimal or best value found by Gurobi, column t_{gurobi} refers to the time taken by Gurobi–which is interrupted after 100 seconds– and column *% gap* refers to the percentage difference between $\underline{z}_{LR}$ and z_{gurobi}. In Table 3, column *speedup* refers to how many times faster the lagrangian heuristic was compared to Gurobi.

# bids	# items	$\overline{z}_{LR}$	$\underline{z}_{LR}$	t_{LR}	z_{gurobi}	t_{gurobi}	% gap
41	79	4226	2997	17 ms	3136	15 ms	4.4
42	58	2986	2084	11 ms	2179	5 ms	4.3
44	101	4668	3220	16 ms	3272	2 ms	1.6
25	51	3109	2320	7 ms	2320	3 ms	0
31	77	3958	2984	12ms	3017	9ms	1.1
20	60	2992	1840	4ms	1840	1ms	0

Table 2: Results of the lagrangian heuristic for instances of CATS.

#bids	# items	$\overline{z}_{LR}$	$\underline{z}_{LR}$	t_{LR}	z_{gurobi}	t_{gurobi}	% gap	speedup
290	1508	55432	31475	6.79s	39186	$\overline{100}$s	19	14.72
127	1159	25884	18377	3.1	20451	0.33s	10	0.10
388	1542	54754	27592	16s	39299	$\overline{100}$s	29	6.25
452	1416	57871	28536	12s	39176	$\overline{100}$s	28	8.33
234	1463	49180	25460	5s	33272	$\overline{100}$s	23	20

Table 3: Results of the lagrangian heuristic for larger instances of CATS.

5.3 Branch-and-Cut

For the branch-and-cut, we consider the same five instances as the ones considered in Table 3.

In Table 4, we compare results obtained for a branch-and-cut algorithm which only adds conflict graph clique inequalities–as presented in Section 2–against a standard call to Gurobi. In Table 4, each line corresponds to an instance and its solution, column z_{BC} refers to the objective function value found by this branch-and-cut algorithm, column *#user-cuts* refers to the number of conflict graph clique inequalities included during the branch-and-cut, column *# nodes* refers to the number of nodes explored during the branch-and-cut, column t_{BC} refers to the time taken for the branch-and-cut to finish, column $\%t_{UC}$ refers to the percentage of the execution time that is spent on solving the separation problem on each node, column z_{gurobi} refers to the objetive function value found by the standard Gurobi call, column t_{gurobi} refers to the time taken by the standard Gurobi call to finish and column *% gap* refers to the percentage difference between z_{BC} and z_{gurobi}.

z_{BC}	#user-cuts	# nodes	t_{BC}	$\%t_{UC}$	z_{gurobi}	t_{gurobi}	% gap
38212	955	1983	$\overline{100}$s	91	39186	$\overline{100}$s	2.5
20451	44	168	2.51s	93	20451	0.33s	0
38846	476	761	$\overline{100}$s	96	39299	$\overline{100}$s	1
38097	380	538	$\overline{100}$s	97	39176	$\overline{100}$s	2.7
33272	741	2896	99s	96	33272	$\overline{100}$s	0

Table 4: Comparison of a branch-and-cut based on conflict graph clique inequalities compared to a standard call to Gurobi.

In Table 5, we present details on the results obtained for the standard calls to Gurobi. In Table 5, column *# nodes* refers to number of nodes explored and column *# cuts* refers to the number of cuts included by this standard Gurobi call, without including any conflict graph clique inequality.

# nodes	# cuts	t_{gurobi}	z_{gurobi}
6254	55	$\overline{100}$	39186
101	20	0.16	20451
2460	56	$\overline{100}$	39299
1045	100	$\overline{100}$	39176
5118	38	$\overline{100}$	33272

Table 5: Results of a standard call to Gurobi for instances of CATS.

In Table 6 we present results of a branch-and-cut algorithm which includes all the standard valid inequalities from Gurobi–such as Gomory, Clique, MIR, Zero Half and RLT– in addition to the conflict graph clique inequalities, as presented in Section 2. In Table 6, each node corresponds to an instance and its solution, column *#user-cuts* refers to the number of conflict graph clique inequalities included during the branch-and-cut, column *# other cuts* refers to the number of standard Gurobi cuts included, column*# nodes* refers to the number of nodes explored during the branch-and-cut, column t_{BC} refers to the time taken for the branch-and-cut to finish, column $\%t_{UC}$ refers to the percentage of the execution time that is spent on solving the separation problem on each node and column z_{BC} refers to the objective function value found by this branch-and-cut algorithm.

# user cuts	# other cuts	# nodes	t_{BC}	$\%t_{UC}$	z_{BC}
742	53	2184	$\overline{100}$s	60	39397
24	27	80	0.42s	50	20451
600	59	1017	$\overline{100}$s	62	39299
19	98	278	$\overline{100}$s	28	38290
603	41	3716	88s	61	33272

Table 6: Results of a branch-and-cut algorithm including all kinds of inequalities for instances of CATS.

Note from Tables 5 and 6 that the addition of conflict graph clique inequalities to the branch-and-cut algorithm from Gurobi–which originally includes Gomory, Clique, MIR, Zero Half and RLT cuts– led to a significant decrease in the number of nodes explored during enumeration, and led to a slight improvement of the best objective function value found.

6 Conclusion and Future Work

In this work, we presented the combinatorial auction problem and proposed a heuristic algorithm based on lagrangian relaxation and a branch-and-cut algorithm based on conflict graph clique inequalities.

We believe that these two algorithms provide simple to understand approaches to the combinatorial auction problem.

On computational tests, the lagrangian heuristic proved to find near-optimal solution for small instances. And for the larger instances, the lagrangian heuristic proved to be a much faster way to obtain feasible solutions, with speed-ups up to 14 and 20 times the time taken by standard Gurobi, and within a 20% optimality gap.

The inequalities presented in Section 2 also proved to be a promising approach to this computationally hard problem. As presented in Section 5, the number of cuts of this kind included during the branch-and-cut algorithm proposed in Section 4 is significant. From Tables 5 and 6, note that in all of the instances considered, the inclusion of this kind of inequality led to a decrease in the number of nodes explored during the branch-and-cut and also led to an improvement in the best objective function value found.

While the idea of the conflict graph and the separation problem described in Section 4 lead to a significant number of included cuts and decrease in the number of nodes explored, we note that the amount of time necessary to solve the separation problem is significantly high. Since the node set and the edge set of the conflict graph associated with a given instance is constant–with only the weights changing for each node in the enumeration–all the maximal cliques in the conflict graph can be determined just once. A possibility of further investigation and improvement would be to determine, before enumeration, all the maximal

cliques in the conflict graph and then, on each node in the enumeration tree, define the separation problem as the problem of querying this set of maximal cliques in order to find a clique of maximum–or large enough–weight. We believe that an efficient querying strategy would lead to significant improvements to the overall performance of the branch-and-cut approach.

References

[1] Bezanson, J., Edelman, A., Karpinski, S. *Julia: A fresh approach to numerical computing*, 2017.

[2] Cai, S., Lin, J. *Fast Solving Maximum Weight Clique Problem in Massive Graphs*, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016.

[3] Guo, Y., Lim, A., Rodrigues, B., Tang, J. *Using a Lagrangian Heuristic for a Combinatorial Auction Problem*, International Journal on Artificial Intelligence Tools, 2006.

[4] Leyton-Brown, K., Pearson, M., Shoham, Y. *Towards a Universal Test Suite for Combinatorial Auction Algorithms*, Proceedings of the 2nd ACM conference on Electronic commerce, 2000.

[5] Rossi, R. A., Ahmed, N. K. *The Network Data Repository with Interactive Graph Analytics and Visualization*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[6] Wolsey, L. *Integer Programming, Second Edition.* Wiley, 2020.