Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

A Faster Core Constraint Generation Algorithm for Combinatorial Auctions

Benedikt Bünz
Department of Computer Science
Stanford University
buenz@cs.stanford.edu

Sven Seuken
Department of Informatics
University of Zurich
seuken@ifi.uzh.ch

Benjamin Lubin
Information Systems Department
Boston University School of Management
blubin@bu.edu

Abstract

Computing prices in core-selecting combinatorial auctions is a computationally hard problem. Auctions with many bids can only be solved using a recently proposed core constraint generation (CCG) algorithm, which may still take days on hard instances. In this paper, we present a new algorithm that significantly outperforms the current state of the art. Towards this end, we first provide an alternative definition of the set of core constraints, where each constraint is weakly stronger, and prove that together these constraints define the identical polytope to the previous definition. Using these new theoretical insights we develop two new algorithmic techniques which generate additional constraints in each iteration of the CCG algorithm by 1) exploiting separability in allocative conflicts between participants in the auction, and 2) by leveraging non-optimal solutions. We show experimentally that our new algorithm leads to significant speed-ups on a variety of large combinatorial auction problems. Our work provides new insights into the structure of core constraints and advances the state of the art in fast algorithms for computing core prices in large combinatorial auctions.

1 Introduction

Combinatorial auctions (CAs) have found application in many real-world domains, including procurement auctions (Sandholm 2007), TV advertising auctions (Goetzendorff et al. 2014) and government spectrum auctions (Cramton 2013; Ausubel and Baranov 2014). CAs are attractive, as they can produce efficient outcomes even when bidders have complex preferences on bundles of heterogeneous items. However, the construction of such auctions requires myriad design decisions, even if we limit the scope to sealed bid mechanisms. First, a potential design must include a reasonable bidding language for participants to use; many have been proposed in the literature, e.g., XOR, OR*, etc. (Nisan 2006). Next, a means for solving the NP-hard winner-determination problem must be obtained (Sandholm 2002). Finally, the design must specify a payment mechanism. The classic answer to this latter question is the well-known VCG mechanism, where agents pay the externality they impose on all other agents (Vickrey 1961; Clarke 1971; Groves 1973).

However, as has been pointed out in the literature (Ausubel and Milgrom 2006), there are numerous issues with VCG, most notably that it may result in arbitrarily low revenue to the seller. This not only creates a strong disincentive for sellers to use VCG, but also opens the possibility that collusive collections of bidders may be able to come together with an outside offer that is more attractive than that in VCG, undermining the VCG-based regime. This observation has led to considerable interest in *core-selecting combinatorial auctions* (Ausubel and Milgrom 2002; Milgrom 2007; Day and Milgrom 2008), which offer a principled way to ensure that revenue in the auction is guaranteed to be high enough to avoid such incentives towards collusion.

However, solving this economic problem has created a computational one, as the naïve formulation of the pricing problem requires computing the optimal allocation for all 2^n coalitions to describe the core polytope. The state-of-the art approach is to use *constraint generation* to consider only the most valuable coalitions, which leads to a moderate number of constraints in expectation (Day and Raghavan 2007). Nevertheless, run-time is still a limiting factor in practice.

1.1 Overview of Results

In this paper, we propose several novel methods for speeding up the core pricing algorithm by reducing the number of CCG iterations in the algorithm, and thus the number of NP-hard problems that need to be solved. We offer:

1. Limits on the effectiveness of an existing heuristic.
2. Theoretical results that enable stronger and more precise core constraints.
3. Two algorithmic ideas that generate additional constraints in each CCG iteration by exploiting separability in allocative conflicts between participants and by leveraging non-optimal solutions.
4. Experimental results, showing that our main algorithm significantly outperforms the current state of the art.

1.2 Related Work

This paper directly extends the earlier work of Day and Raghavan (2007), where they propose to use *constraint generation* to concisely codify the pricing problem in core-selecting CAs. We also include more recent advances, proposed by Day and Cramton (2012), where core prices are chosen that minimize the L_2 metric to VCG.

The constraint generation method dates back to the 50s, as a way to solve math programs that contain too many constraints to capture directly (Dantzig, Fulkerson, and Johnson 1954; Balinski 1965). Instead of solving the full program at once, the primary problem is solved with only a subset of its constraints yielding a provisional solution. Then a secondary problem is formulated using this provisional solution, the result of which yields either a new constraint for the primary, or a proof that the primary is already correct. The algorithm iterates between the two problems, until such a proof is obtained (Nemhauser and Wolsey 1988). Such methods have found wide applicability in the operations literature in areas such as airline scheduling (Hoffman and Padberg 1993) and portfolio optimization (Künzi-Bay and Mayer 2006).

2 Preliminaries

In a *combinatorial auction (CA)*, there is a set G of m distinct, indivisible items, and a set N of n bidders. Each bidder i has a *valuation function* v_i which, for every bundle of items $S \subseteq G$, defines bidder i's value $v_i(S) \in \mathbb{R}$, i.e., the maximum amount that bidder i would be willing to pay for S. To simplify notation, we assume that the seller has zero value for the items. However, all of our results also hold for settings where the seller has non-zero value for the items (see Day and Cramton (2012) for a discussion on how to handle reserve prices in core-selecting payment rules).

We let $p_i \in \mathbf{p}$ denote bidder i's payment, and we assume that bidders have quasi-linear utility functions, i.e., $u_i(S, p_i) = v_i(S) - p_i$. Bidders may make a non-truthful report about their value function to the mechanism. However, in this paper we do not study the incentive properties of the payment rule, and thus we do not need to distinguish between the agents' true value and their value report. To simplify notion, we use v_i to denote an agent's report, as this is what the mechanism gets as input. Additionally, we will assume in our notation that bidders use the XOR bidding language, but all of our theoretical results in Section 4 will apply to any bidding langue.

We define an *allocation* $X = (X_1, \ldots, X_n) \subseteq G^n$ as a vector of bundles, with $X_i \subseteq G$ being the bundle that i gets allocated. An allocation X is *feasible* if $X_i \cap X_j = \emptyset \quad \forall i, j \in N, i \neq j$. We let $\mathbb{X}$ denote the set of feasible allocations. With each allocation X we associate a *coalition* $C(X) = \{i | i \in N, X_i \neq \emptyset\}$, i.e., those agents that get allocated under X. We define the total value of allocation X to agents C as $V_C(X) = \sum_{i \in C} v_i(X_i)$. If we omit the subscript C we assume it to be N, i.e., $V(X) = V_N(X)$.

A mechanism's *allocation rule* maps the bidders' reports to an allocation. We consider rules that maximize *social welfare*, which can be formalized as an integer program (IP) according to the details of the bidding language. We denote the solution to this as $wd(N) = \arg\max_X V(X)$, subject to $X \in \mathbb{X}$, when the bids of all bidders are considered in the maximization. When only the bids of the coalition C shall be considered, we write $wd(C) = \arg\max_X V_C(X)$.

A *mechanism* specifies an *allocation rule*, defining who gets which goods, and a *payment rule*, defining prices. Together, these define the auction's *outcome*, denoted O.

2.1 Core-selecting Combinatorial Auctions

With this background, we are ready to consider appropriate *payment rules* for CAs. The famous VCG mechanism (Vickrey 1961; Clarke 1971; Groves 1973) is an appealing candidate because it is strategyproof (i.e., no individual bidder can benefit from misreporting his value). Unfortunately, in CAs where some items are complements, using VCG may result in an outcome outside of the *core*. Informally, this means that a coalition of bidders is willing to pay more in total than what the seller receives from the current winners. To avoid such undesirable outcomes, recent auction designs have employed payment rules that restrict prices to be in the core (Day and Milgrom 2008; Day and Raghavan 2007).

Formally, given outcome O, we let the *coalition* C_O denote the set of bidders who are allocated under outcome O.

Definition 1 (Blocking Coalition). *An outcome $\overline{O}$ is* blocked*, if (a) it is not individually rational, or (b) there exists another outcome O which generates strictly higher revenue for the seller and for which every bidder in the corresponding coalition C_O weakly prefers O over outcome $\overline{O}$. The coalition C_O is called a* blocking coalition*.*

Definition 2 (Core). *An outcome that is not blocked by any coalition is called a* core outcome*.*

We can restrict our attention to allocation rules that are efficient because all inefficient outcomes are not in the core. Thus, it suffices to think about the payments, and the remaining challenge is to find *payments that lie in the core*. Intuitively, the payments for the winners must be *sufficiently large*, such that there exists no coalition that is willing to pay more to the seller than the current winners are paying.

Formally, we let W denote the set of winners. Given W and p, the opportunity cost of already-winning members of the coalition C is $V_C(wd(N)) - \sum_{i \in C} p_i$. Thus, the condition that p lies in the core can be expressed as follows (Day and Raghavan 2007):

$$\sum_{i \in W} p_i \geq V(wd(C)) - V_C(wd(N)) + \sum_{i \in C} p_i \quad \forall C \subseteq N \quad (1)$$

This means that the core polytope can be defined by having one constraint for each possible coalition $C \subseteq N$. The left-hand side (LHS) of each constraint is the sum of the winning payments; the right-hand side (RHS) is the value of the agents in coalition C for the allocation chosen if only their bids are considered, i.e., $V(wd(C))$, minus the opportunity cost of already-winning members of the coalition. Because $p_i : i \in W \cap C$ appears on both sides, and because $p_i = 0$ for $i \notin W$, the core constraints can be simplified to:

$$\sum_{i \in W \setminus C} p_i \geq V(wd(C)) - V_C(wd(N)) \quad \forall C \subseteq N \quad (2)$$

Note that the core is defined in terms of bidders' true values. However, given that no strategyproof core-selecting CA exists, we must expect that bidders will be non-truthful. Goeree and Lien (2014) have recently shown via a Bayes-Nash equilibrium analysis that the outcome of a core-selecting CA can be outside the true core. Thus, core-selecting CAs only guarantee to produce outcomes in the *revealed core*, i.e., in the core with respect to *reported values.*

2.2 Core-constraint Generation (CCG)

Because the number of core constraints is exponential in n, it is impossible to enumerate them for even medium-sized CAs; fortunately we can often consider only a small fraction of them by using the method of *constraint generation* as described by Day and Raghavan (2007). This iterative method applies the following two steps in each round t:

1. find a candidate payment vector p^t given all core constraints generated so far
2. find the *most blocking coalition* (if any), given the current candidate payment vector p^t

The algorithm is initialized using the VCG payments p^{VCG} as the first payment vector. Step (2) then requires solving the following IP:

$$z(p^t) = \max \sum_{i \in N} \sum_{S \subseteq G} v_i(S) y_i^S - \sum_{i \in W} (v_i(wd(N)) - p_i^t)\gamma_i \quad (3)$$

$$\text{subject to} \sum_{S \supseteq \{j\}} \sum_{i \in N} y_i^S \leq 1 \quad \forall j \in G \quad (4)$$

$$\sum_{S \subseteq G} y_i^S \leq 1 \quad \forall i \in N \setminus W \quad (5)$$

$$\sum_{S \subseteq G} y_i^S \leq \gamma_i \quad \forall i \in W \quad (6)$$

$$\gamma_i \in \{0,1\}, y_i^S \in \{0,1\} \quad \forall i \in N, S \subseteq G \quad (7)$$

In this IP, we have two kinds of decision variables. First, for all bidders i, γ_i is equal to 1 if bidder i is part of the most blocking coalition. Second, for all i, and all bundles of goods $S \subseteq G$, y_i^S is equal to 1 if bidder i is part of the most blocking coalition and is allocated bundle S. Note that $v_i(wd(N))$ denotes bidder i's value for the efficient allocation.

This IP essentially solves the winner determination problem, but where winning bids are reduced by their opportunity cost. The objective $z(p^t)$ represents the *coalitional value* of the most blocking coalition, i.e., the maximum total payment which the coalition would be willing to offer the seller. If this amount is equal to the current sum of the winners' payments (i.e., if $z(p^t) = \sum_i p_i^t$), then no blocking coalition exists, and the overall algorithm terminates. Otherwise we can utilize the constraint set (2) to create the constraint $\sum_{i \in W \setminus C^\tau} p_i \geq z(p^\tau) - \sum_{i \in W \cap C^\tau} p_i^\tau$, where C^τ denotes the coalition identified in round τ of the algorithm. This is then added to the following LP, which is solved to find the candidate price vector for the next iteration:

$$p^t = \arg\min \sum_{i \in W} p_i \quad (8)$$

$$\text{subject to} \sum_{i \in W \setminus C^\tau} p_i \geq z(p^\tau) - \sum_{i \in W \cap C^\tau} p_i^\tau \quad \forall \tau \leq t \quad (9)$$

$$p_i^{\text{VCG}} \leq p_i \leq v_i(wd(N)) \quad \forall i \in W. \quad (10)$$

This LP will find a new candidate price vector, where each p_i is lower bounded by i's VCG payment and upper-bounded by i's bid. Additionally, the prices will obey all core constraints (9) that have been added in any prior iteration. Here, the objective of the LP is to minimize total bidder payments, which Day and Raghavan (2007) argue reduces the total potential gains from deviating from truth-telling. However, this does not result in a unique price vector. Parkes, Kalagnanam, and Eso (2001) originally introduced the idea of finding payments that minimize some distance metric to VCG payments. Following this idea, Day and Cramton (2012) propose to minimize the L_2 norm to VCG by solving a Quadratic Program (QP), which produces a unique price vector in every iteration of the algorithm. This is also the approach taken in practice for the most recent spectrum auctions in the UK, the Netherlands, and Switzerland. For this reason, we also adopt the L_2 norm for our experiments in Section 6.

3 The Max-Traitor Heuristic

In running CCG, it is often the case that multiple blocking coalitions have the same coalitional value, and thus we have a choice over which specific core constraint to add in a particular iteration. Day and Cramton (2012) briefly mention in their appendix that it might be helpful to minimize the cardinality of $W \setminus C$ as a secondary objective when searching for the most blocking coalition. This is equivalent to maximizing $W \cap C$, i.e., the number of winners in C, which we henceforth call *traitors*. We use the term *Max-Traitor heuristic* to refer to the algorithm that generates constraints with a maximal number of traitors (given the same coalitional value).

One intuition as to why this heuristic may be helpful is that it will decrease the number of variables in a generated constraint, thereby strengthening it. We will build upon this intuition to strengthen the core constraints themselves in Section 4. The following example demonstrates why the Max-Traitor heuristic can be useful:

Bidder 1: {A}=10	Bidder 3: {C}=10
Bidder 2: {B}=10	Bidder 4: {A,B}=6

Bidders 1, 2 and 3 form the winning coalition with a total value of 30. Two different coalitions with coalitional value 6 are available: {4} and {3, 4}. The former coalition, which contains no traitors, induces the constraint:

$$p_1 + p_2 + p_3 \geq 6 \quad (11)$$

resulting in provisional payments of (2,2,2,0) when first minimizing the sum of all payments, and then minimizing the L_2 metric to VCG. The coalition {3, 4}, which contains a traitor (Bidder 3), induces the constraint:

$$p_1 + p_2 \geq 6 \quad (12)$$

This constraint is stronger, as it contains fewer variables, and immediately leads to the final core payments of (3,3,0,0).

Day and Cramton (2012) state in their appendix that among their test cases, maximizing the set of traitors never led to constraints that were *non-binding* at termination of the algorithm. However, we have been able to identify examples where the heuristic produces non-binding constraints. We now provide a representative example of how this occurs.

Bidder 1: {A}=10	Bidder 5: {A,B,C,D,E}=12
Bidder 2: {B}=10	Bidder 6: {A,B,E}=8
Bidder 3: {C}=10	Bidder 7: {C,D,E}=8
Bidder 4: {D}=10	

Bids {1,2,3,4} form the winning allocation with a value of 40. The VCG payments in this example are 0 for all bidders. The unique most blocking coalition consists of Bidder 5 and has a coalitional value of 12, which is higher than the current total payments of 0. Thus, the generated constraint is:

$$p_1 + p_2 + p_3 + p_4 \geq 12. \quad (13)$$

This leads to the payment vector (3,3,3,3,0,0,0). Next, the algorithm finds the blocking coalition consisting of {1,2,7} with a coalitional value of 14, which is higher than the current total payments of 12. The generated constraint is:

$$p_3 + p_4 \geq 8 \quad (14)$$

This leads to the payment vector (2,2,4,4,0,0,0).

Next, the blocking coalition {3, 4, 6} with coalitional value 16 is selected, again with value greater than the current payments of 12, and leading to the constraint:

$$p_1 + p_2 \geq 8 \quad (15)$$

This produces the final payment vector (4,4,4,4,0,0,0). There does not exist a blocking coalition at this payment vector, so the algorithm terminates. However, the constraint (13) ends up being non-binding, even though we employed the Max-Traitor heuristic. Note that this is a representative example which does not exploit any particular corner case of the bidding structure. This example not withstanding, the Max-Traitor heuristic is often effective at reducing the runtime of the CCG algorithm, yet, our algorithm dominates it (see Sections 5 and 6).

4 Theoretical Results

4.1 Core of Non-Blocking Allocations

The core of an auction is generally defined in terms of coalitions. We now extend this definition to allocations. First, remember the definition we provided in Section 2.1:

$$\text{(C1):} \sum_{j \in W \setminus C} p_j \geq V(wd(C)) - V_C(wd(N)) \quad \forall C \subseteq N \quad (16)$$

We now formulate this constraint set in terms of allocations.

$$\text{(C2):} \sum_{j \in W \setminus C(X)} p_j \geq V(X) - V_{C(X)}(wd(N)) \quad \forall X \in \mathbb{X} \quad (17)$$

We will call any allocation X for which this constraint is violated a *blocking allocation*. Proposition 1 shows that the constraint set $(C2)$ describes the same core prices as $(C1)$, even if it contains weakly more constraints.

Proposition 1 (Core of Non-Blocking Allocations). *The two sets of constraints (C1) and (C2) describe the same core.*

Proof. We will show that for each constraint in one set there exists a constraint in the other set that implies it. The two constraint sets, therefore, must describe the same polytope.

"$(C2) \implies (C1)$": Every constraint in $(C1)$ corresponds to a coalition C. For every such C. there exists an allocation X such that $X = wd(C)$. Thus every constraint in $(C1)$ also exists in $(C2)$.

"$(C1) \implies (C2)$": Every constraint in $(C2)$ corresponds to an allocation X which in turn corresponds to a coalition $C(X)$. Because $(C1)$ contains one constraint for every coalition. it also contains one for the coalition $C(X)$. Because the winner determination algorithm selects the value-maximizing allocation for a given coalition. we know that $V_{C(X)}(wd(C(X))) \geq V(X)$. Thus, the constraint corresponding to coalition $C(X)$ in $(C1)$ is weakly stronger than the corresponding constraint in $(C2)$. □

Remark 1. *The existence of a blocking allocation implies the existence of a blocking coalition.*

4.2 Core Conflicts

Let us now re-consider the first example from Section 3, and the constraint (11) that was generated through the coalition {4}. The constraint implicitly assumes that the coalition. i.e. Bidder 4 cares about the Bidder 3's payment, even though the allocations from Bidder 3 and Bidder 4 are not in conflict. Bidder 4 in fact has 0 value for good C and Bidder 3 is single minded on good C. Thus, Bidder 4 should be indifferent to Bidder 3's payment. We will now show that we can formalize this intuition and thereby generate a more powerful set of core constraints

We say that an allocation X^1 is *in conflict with* an allocation X^2 if it is not feasible to simultaneously realize the two allocations X^1 and X^2. Analogously, a set of winners $W' \subset W$ is not in conflict with an allocation X if there exists a feasible allocation $X' = X \cup wd(W')$, where we let "$\cup$" denote the natural combination of two allocations. Observe that in the example discussed above, winners 1 and 2 are in conflict with the allocation corresponding to coalition {Bidder 4} but winner 3 is not, i.e. 1 and 2 receive goods that bidder 4 wants.

As we have seen, some *blocking allocations* are not in conflict with all winners W. Let $W_X \subseteq W$ denote the set of winners who *are* in conflict with an allocation X. By dropping the non-conflicted winners from the left side of equation $(C2)$, we get the following set of weakly stronger constraints:

$$\text{(C3):} \sum_{i \in W_X \setminus C(X)} p_i \geq V(X) - V_{C(X)}(wd(N)) \quad \forall X \subseteq \mathbb{X} \quad (18)$$

The following theorem shows that $(C3)$ still describes the same core polytope as $(C1)$ and $(C2)$. In Section 5, we will show that this insight is valuable because it enables us to generate, for any blocking allocation, a weakly stronger core constraint while preserving the core polytope.

Theorem 1 (Weakly Stronger Core Constraints). *The two sets of constraints $(C2)$ and $(C3)$ describe the same core.*

Proof. We follow the same structure as the previous proof:

"$(C3) \implies (C2)$": For each constraint in $(C2)$ there is a corresponding constraint in $(C3)$ defined on the same allocation X. If all winners in $W \setminus C(X)$ are in conflict with

X, then the two constraints are the same. If there exists a winner in $W \setminus C(X)$ that is not in conflict with X, then we have strictly fewer variables on the LHS of the constraint in $(C3)$ but the same (constant) terms on the RHS. Because payments are positive and additive, we see that the constraint in $(C3)$ is strictly stronger. In both cases, the constraint in $(C3)$ implies the constraint in $(C2)$.

"$(C2) \implies (C3)$": Each constraint in $(C3)$ corresponds to some allocation $\bar{X}$, and we let W_X denote the corresponding set of winners that are in conflict with X. Let us call the subset of $wd(N)$ that allocates goods to $W \setminus W_X$, $\bar{X}$. We consider the constraint in $(C2)$ that corresponds to the allocation $X \cup \bar{X}$. This allocation needs to be feasible because X and $\bar{X}$ are not in conflict. We can, therefore, infer the following:

$$V(X \cup \bar{X}) = V(X) + V(\bar{X}) = V(X) + V_{W \setminus W_X} wd(N) \quad (19)$$

Since $C(X \cup \bar{X}) = C(X) \cup (W \setminus W_X)$ we can infer the following on the opportunity value of $C(X \cup \bar{X})$

$$V_{C(X \cup \bar{X})}(wd(N)) \leq V_{C(X)}(wd(N)) + V_{W \setminus W_X} wd(N) \quad (20)$$

We can also infer that

$$W \setminus C(X \cup \bar{X}) = W \setminus (W \setminus W_X) \setminus C(X) = W_X \setminus C \quad (21)$$

If we plug this into the constraint set $(C2)$. we get:

$$\sum_{i \in W_X \setminus C(X)} p_i = \sum_{i \in W \setminus C(X \cup \bar{X})} p_i \quad \text{by (21)} \quad (22)$$

$$\geq V(X \cup \bar{X}) - V_{C(X \cup \bar{X})}(wd(N)) \quad \text{by (C2)} \quad (23)$$

$$\geq V(X) + V_{W \setminus W_X} wd(N) - \quad \text{by (19)} \quad (24)$$

$$(V_{C(X)}(wd(N)) + V_{W \setminus W_X}(wd(N))) \quad \text{by (20)} \quad (25)$$

$$= V(X) - V_{C(X)}(wd(N)) \quad (26)$$

The constraint corresponding to $X \cup \bar{X}$ in $(C2)$, therefore, implies the constraint corresponding to X in $(C3)$. □

The theorem as well as the constructive nature of the proof indicates that for any constraint from constraint set $(C2)$ there exists a weakly stronger constraint in set $(C3)$ that can directly be constructed. "Small" allocations that are in conflict with only few winners are especially strengthened through $(C3)$.

4.3 Splitting up allocations

Our next results shows that we can split up allocations in the constraint generation process under certain conditions.

Proposition 2. *Consider an allocation X that consists of two separable sub-allocations X_1 and X_2 such that $X = X_1 \cup X_2$ and $C(X_1) \cap C(X_2) = \emptyset$. If $W_{X_1} \cap W_{X_2} = \emptyset$, then the constraints corresponding to X_1 and X_2 imply the constraint corresponding to X.*

Proof. The constraint generated by X_1 is:

$$\sum_{i \in W_{X_1} \setminus C(X_1)} p_i \geq V(X_1) - V_{X_1}(wd(N)) \quad (27)$$

The constraint generated by X_2 is:

$$\sum_{i \in W_{X_2} \setminus C(X_2)} p_i \geq V(X_2) - V_{X_2}(wd(N)) \quad (28)$$

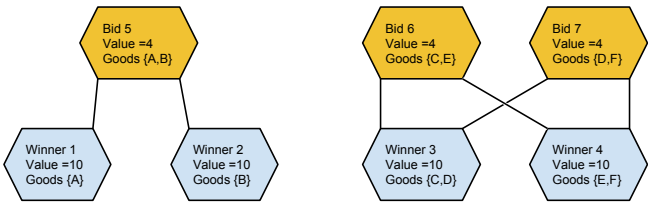

Figure 1: Conflict Graph for "Separability Example."

Because $W_{X_1} \cap W_{X_2} = \emptyset$ and $C(X_1) \cap C(X_2) = \emptyset$. we can add (27) and (28). leading to the constraint for X:

$$\sum_{i \in (W_{X_1} \cup W_{X_2}) \setminus C(X)} p_i \geq V(X) - V_X(wd(N)). \quad (29)$$

□

We can use this proposition to speed our constraint generation algorithm, as we describe next.

5 Algorithmic Ideas

Based on the theoretical results derived in the previous section, we now present two algorithmic ideas that improve upon the standard CCG algorithm. The goal of both ideas is to reduce the number of iterations of the CCG algorithm, whilst not increasing the complexity of each iteration.

5.1 Exploiting "Separability"

Our first idea exploits certain structure in the conflicts between the winners and losers to split up core constraints, which we can formally capture via a *conflict graph* (see Figure 1). The conflict graph for a blocking allocation X, given a set of winners W, is a bipartite graph $G = (C(X) \cup W, E)$ with $(i, j) \in E$ if and only if j is in *conflict with* i's allocation. If the conflict graph is separable into disconnected components (as in Figure 1, described next), then the corresponding allocation can be split up, and multiple constraints can be generated in one step (as described below).

Consider the example in Figure 1, where the bids of the blocking coalition are modeled as light (yellow) nodes, the winning allocations are modeled as dark (blue) nodes, and conflicts as edges. Assume that the VCG payments lie at the origin. A straightforward application of CCG would find bids {5, 6, 7} as the most blocking allocation with a value of $4+4+4 = 12$, which produces the following constraint:

$$p_1 + p_2 + p_3 + p_4 \geq 12. \quad (30)$$

This constraint by itself leads to the payment vector of (3,3,3,3,0,0,0). We now see that bids {6, 7} are still blocking, leading to the following constraint:

$$p_3 + p_4 \geq 8. \quad (31)$$

The resulting payment vector is (2,2,4,4,0,0,0), which is also the final payment vector.

Our first algorithmic idea called *Separability* exploits the separability between the bidders. In particular, in our example, Bid 5 is only in conflict with the allocations of winners 1 and 2, and bids 6 and 7 are only in conflict with the allocations of winners 3 and 4. If we generate one constraint for each separated allocation (strengthened via Theorem 1 and

Proposition 1), we get the following two constraints in one step:

$$p_1 + p_2 \geq 4 \quad (32)$$

$$p_3 + p_4 \geq 8 \quad (33)$$

Based on Proposition 1 we know that the new constraints are not over-constraining the price vector, and based on Proposition 2 we know that this split-up of the allocation implies the original constraint $p_1 + p_2 + p_3 + p_4 \geq 12$ (i.e., we are not under-constraining the price vector). Thus, our algorithm is still guaranteed to terminate with the correct core price vector. Indeed, the resulting payment vector after adding these two constraints is (2,2,4,4,0,0,0). Thus, exploiting the separability of the conflict graph, we reach the final payment vector in just one iteration of the CCG algorithm instead of two. Whenever the conflict graph is separable in the way described, we can add multiple constraints in one step. Note that this does not guarantee that the overall algorithm will necessarily terminate in fewer steps. However, our expectation is that, *on average*, this will reduce the number of CCG iterations, each of which requires the solution of an NP-hard problem (finding a blocking coalition). We will show in Section 6 that the *Separability* idea indeed leads to a significant speed-up of the CCG algorithm.

5.2 Exploiting "Incumbent Solutions"

Our second algorithmic technique employs previously discarded intermediate solutions to generate additional constraints that would otherwise not be included. Remember that the CCG algorithm solves an IP in every iteration to find the most blocking coalition, and the optimal solution to this IP then corresponds to one new core constraint. We note that every *feasible*, even *non-optimal* solution to this IP still corresponds to a feasible allocation, which by Proposition 1 corresponds to a core constraint. The main idea of "Incumbent Solutions" is to collect all (non-optimal) incumbents that are found while solving one instance of the IP, and to add all corresponding core constraints at once. The motivation for adding core constraints based on sub-optimal solutions is that these constraints may have been added in later iterations anyway (with some likelihood), and by adding them earlier we can save the corresponding CCG iterations.

Fortunately, the majority of optimization algorithms for solving IPs are tree-search algorithms (e.g., using branch-and-bound). These algorithms automatically encounter intermediate solutions while searching for the optimal solution. In our experiments, we use IBM's CPLEX to solve the IPs, and the default branch-and-bound IP-solver automatically collects all incumbent solutions without incurring extra work. We will show in Section 6 that the *Incumbents* idea indeed leads to a significant speed-up of the CCG algorithm.

6 Experiments

To test the ideas proposed in Section 5, we ran experiments using CATS (Combinatorial Auction Test Suite) (Leyton-Brown and Shoham 2006). We used each of the standard CATS distributions, as well as a simple legacy distribution from the literature (Sandholm 2002) called *Decay* (aka $L4$). We varied the number of goods between 16 and 256, and the number of bids between 500 and 4,000. For each combination of distribution, number of goods and number of bids, we created 50 random CATS instances. In total we created 1,550 instances and ran 9,300 experiments. The experiments were run on a high-performance computer with 24 $\times$ 2.2GHZ AMD Opteron cores and 66GB of RAM. All mathematical programs were solved using CPLEX 12.6. The total run-time of all experiments was more than 40 days.

6.1 Experimental Results

We present our primary results in Table 1. Each row represents a separate distribution, for which we report the number of goods and bids that were used to generate the particular instance. As Day and Raghavan (2007) have shown, the run-time of CCG scales exponentially in the number of bids and goods. Accordingly, the run-times for smaller CATS instances are often negligible. Therefore, Table 1 only includes results for the largest instances we could solve. Note that all problem instances reported in this table are larger than even the largest instances reported by Day and Cramton (2012). Our baseline is the standard CCG algorithm, for which we report the run-time in minutes, averaged over the 50 runs.[1] The standard deviation of the run-times per distribution is provided in parentheses, to give an idea, for each of the distributions, how homogeneous or heterogeneous the problem instances are. Next we present the relative run-time (as a percentage of the baseline run-time) for *Max-Traitor*, *Separability*, *Incumbents*, and *Separability + Incumbents*. Note that all algorithms share the same code-base, and only vary regarding which particular constraint(s) they add in each iteration, making this a fair run-time comparison.

The results in Table 1 show that Max-Traitor already does a bit better than standard CCG for most distributions. Next, we see that Separability is roughly 50% faster on L4, Matching and Paths, while it leads to only modest improvements on the other three distributions. While Incumbents leads to similar speed-ups on L4, Matching and Paths as Separability, it is also able to significantly reduce the run-time on Arbitrary, Regions, and Scheduling. But most importantly, we can see (in the last column of Table 1) that "Separability + Incumbents" leads to the largest speed-up on four out of the six distributions and is essentially tied with Incumbents for the other two. For L4, Matching, and Paths, the combined effect of both ideas is most visible: individually, each idea leads to a run-time reduction between roughly 50% and 60%, but combining the two ideas we are able to bring this percentage down to 28%-33%.

[1] Note that this is the run-time for the core constraint generation part of the algorithm only, and does not include the initial run-time for computing VCG prices. We excluded the run-time for computing VCG for two reasons: first, our ideas only affect the core constraint generation part, and thus we are only interested in measuring the resulting speed-up of that part of the algorithm. Second, VCG-prices are just one possible reference point that can be used in core-selecting CAs. Researchers have argued that other reference points could be used, e.g., the origin or well-chosen reserve prices (e.g., Erdil and Klemperer 2010), neither of which would require computing VCG prices.

Distribution	# Goods	# Bids	Standard CCG	Max-Traitor	Separability	Incumbents	Separability + Incumbents
Arbitrary	128	1000	22:37 min (2:34)	101%	95%	67%	**61%**
Decay (L4)	256	2000	8:18 min (0:21)	89%	55%	60%	**33%**
Matching	256	2000	16:57 min (0:59)	83%	56%	54%	**32%**
Paths	256	4000	14:00 min (0:30)	80%	49%	55%	**28%**
Regions	256	2000	45:48 min (11:40)	93%	93%	**51%**	**51%**
Scheduling	128	4000	178:25 min (38:37)	35%	96%	**23%**	24%

Table 1: Results for all six distributions. Absolute run-times (in minutes) are provided for standard CCG (=baseline), with standard error in parentheses. For all other algorithms, the relative run-time to the baseline is provided. All results are averages over 50 instances. In each row, the algorithm with the lowest average run-time is marked in bold.

This speed-up is largely due to our algorithm reducing the number of subordinate NP-hard IP problems that are solved, as illustrated in Table 2. For each of the six distributions, we report the correlation between the run-time of "Separability + Incumbents" and the number of constraint generation iterations of the algorithm. As we see, this correlation is extremely high, except for Scheduling.[2] The last column of Table 2 provides the worst run-time of "Separability + Incumbents" relative to standard CCG across all 50 instances. Thus, we observe that our algorithm performs extremely well, not only *on average* (as shown in Table 1), but also *in the worst case* (somewhat less true for Scheduling).

To get a sense of how these results scale in the size of the auction, consider Figure 2, where we plot the run-time of "Separability + Incumbents" relative to standard CCG for a fixed number of goods (the same as reported in Table 1), increasing the number of bids on the x-axis. Note that we exclude those data points were the average run-time for standard CCG is less than 250ms. Furthermore, each line ends at the maximum number of bids that we were able to run, as reported in Table 1. Figure 2 illustrates that the relative performance advantage of our algorithm generally gets larger as the number of bids is increased for all of the distributions but Scheduling. In total, our results demonstrate that "Separability + Incumbents" leads to a significant speed-up which is often 50% or even larger, and the performance advantage can be expected to grow as the number of bids is increased.

[2]Note that Scheduling has a significantly higher run-time variance than the other distributions; this occurs not only across the 50 instances (as reported in Table 1), but even across rounds for individual instances. Thus, all of our results for Scheduling have a significantly higher margin of error.

Distribution	Run-time & Iteration Correlation	Worst Relative Run-time
Arbitrary	0.83	104%
Decay (L4)	0.98	54%
Matching	0.96	61%
Paths	0.99	47%
Regions	0.91	101%
Scheduling	0.36	133%

Table 2: Correlation between run-time and # of CCG iterations, and the worst run-time of "Separability + Incumbents" relative to standard CCG (=baseline) across all 50 instances.

7 Conclusion

In this paper, we have made several contributions regarding the design of a faster algorithm for core constraint generation in combinatorial auctions. First, we have characterized the limits of the *Max-Traitor* heuristic. Next, we have proposed a new formulation of the *core* in terms of *allocations*, which enabled us to generate weakly stronger core constraints. Based on this theory, we have introduced two algorithmic ideas. The first one takes advantage of structural separability in allocative conflicts between bidders. The second one includes previously unused non-optimal solutions that may become binding in later iterations. Both of these ideas work by generating multiple (strong) constraints in each iteration of the CCG algorithm to reduce the total number of NP-hard problems that need to be solved to compute core prices. Our experimental results using CATS have shown that our main algorithm is significantly faster than the current state-of-the-art algorithm across all distributions.

In real-world CAs, there is often a time constraint on how long solving the pricing problem may maximally take. For this reason, the auctioneer generally restricts the maximal number of bids per bidder. Having an algorithm that is between two and four times faster will enable the auctioneer to allow for additional bids, which will increase efficiency (and thereby revenue/profits) of the auction. Given that CAs are used to allocate billions of dollars of resources every year, we expect our ideas to have a large practical impact.

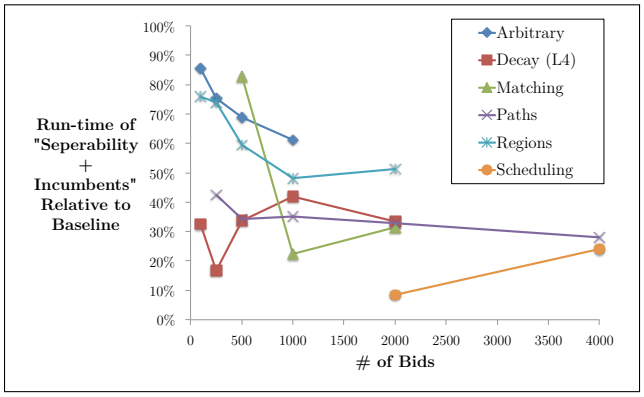

Figure 2: Run-time of "Separability + Incumbents" relative to standard CCG (=baseline), for a fixed number of goods.

References

Ausubel, L. M., and Baranov, O. V. 2014. A practical guide to the combinatorial clock auction. Working paper.

Ausubel, L. M., and Milgrom, P. R. 2002. Ascending auctions with package bidding. *Frontier of Theoretical Economics* 1(1):1–42.

Ausubel, L. M., and Milgrom, P. 2006. The lovely but lonely Vickrey auction. In Cramton, P.; Shoham, Y.; and Steinberg, R., eds., *Combinatorial Auctions*. MIT Press. 17–40.

Balinski, M. L. 1965. Integer programming: methods, uses, computations. *Management Science* 12(3):253–313.

Clarke, E. H. 1971. Multipart pricing of public goods. *Public Choice* 11(1):17–33.

Cramton, P. 2013. Spectrum auction design. *Review of Industrial Organization* 42(2):161–190.

Dantzig, G.; Fulkerson, R.; and Johnson, S. 1954. Solution of a large-scale traveling-salesman problem. *Operations Research* 2:393–410.

Day, R., and Cramton, P. 2012. Quadratic core-selecting payment rules for combinatorial auctions. *Operations Research* 60(3):588–603.

Day, R., and Milgrom, P. 2008. Core-selecting package auctions. *International Journal of Game Theory* 36(3):393–407.

Day, R., and Raghavan, S. 2007. Fair payments for efficient allocations in public sector combinatorial auctions. *Management Science* 53(9):1389–1406.

Erdil, A., and Klemperer, P. 2010. A new payment rule for core-selecting package auctions. *Journal of the European Economics Association* 8(2–3):537–547.

Goeree, J., and Lien, Y. 2014. On the impossibility of core-selecting auctions. *Theoretical Economics*. Forthcoming.

Goetzendorff, A.; Bichler, M.; Day, R.; and Shabalin, P. 2014. Compact bid languages and core-pricing in large multi-item auctions. *Management Science*. Forthcoming.

Groves, T. 1973. Incentives in teams. *Econometrica* 41(4):617–631.

Hoffman, K. L., and Padberg, M. 1993. Solving airline crew scheduling problems by branch-and-cut. *Management Science* 39(6):657–682.

Künzi-Bay, A., and Mayer, J. 2006. Computational aspects of minimizing conditional value-at-risk. *Computational Management Science* 3(1):3–27.

Leyton-Brown, K., and Shoham, Y. 2006. A test suite for combinatorial auctions. In *Combinatorial Auctions*. MIT Press.

Milgrom, P. 2007. Package auctions and exchanges. *Econometrica* 75(4):935–965.

Nemhauser, G. L., and Wolsey, L. A. 1988. *Integer and Combinatorial Optimization*, volume 18. Wiley-Interscience.

Nisan, N. 2006. Bidding languages for combinatorial auctions. In Cramton, P.; Shoham, Y.; and Steinberg, R., eds., *Combinatorial Auctions*. MIT Press.

Parkes, D. C.; Kalagnanam, J.; and Eso, M. 2001. Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 1161–1168.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* 135(1):1–54.

Sandholm, T. 2007. Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine* 28(3):45–58.

Vickrey, W. 1961. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance* 16(1):8–37.