Multiagent Systems: Spring 2006

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Combinatorial Auctions

In a *combinatorial auction*, the auctioneer puts several goods on sale and the other agents submit bids for entire bundles of goods.

Given a set of bids, the *winner determination problem* is the problem of deciding which of the bids to accept.

- The solution must be *feasible* (no good may be allocated to more than one agent).
- Ideally, it should also be *optimal* (in the sense of maximising revenue for the auctioneer).

So besides the game-theoretical problem of stopping bidders from strategising, in combinatorial auctions we also face a challenging *algorithmic* problem.

Plan for Today

- Reminder *why* combinatorial auctions are important
- *Computational* issues: complexity of winner determination in combinatorial auctions
- *Algorithmic* issues: solving a combinatorial auction
- Game-theoretical issues will be discussed next week

Complements and Substitutes

The value an agent assigns to a *bundle* of goods may relate to the value it assigns to the individual goods in a variety of ways ...

- *Complements:* The value assigned to a set is *greater* than the sum of the values assigns to its elements.

  A standard example for complements would be a pair of shoes (a left shoe and a right shoe).

- *Substitutes:* The value assigned to a set is *lower* than the sum of the values assigned to its elements.

  A standard example for substitutes would be a ticket to the theatre and another one to a football match for the same night.

In such cases an auction mechanism allocating one item at a time is problematic as the best bidding strategy in one auction may depend on whether the agent can expect to win certain future auctions.

Combinatorial Auction Protocol

- Setting: one seller (*auctioneer*) and several potential buyers (*bidders*); many *goods* to be sold
- Bidding: the bidders *bid* by submitting their valuations to the auctioneer (not necessarily truthfully)
- Clearing: the auctioneer announces a number of *winning bids*

  The winning bids determine which bidder obtains which good, and how much each bidder has to pay. No good may be allocated to more than one bidder.

Bidding Languages

- As there are $2^n - 1$ non-empty bundles for n goods, submitting complete valuations may not be feasible.
- We assume that each bidder submits a number of atomic bids (B_i, p_i) specifying the price p_i the bidder is prepared to pay for a particular bundle B_i.
- The *bidding language* determines what combinations of individual bids may be accepted. Today, we (mostly) assume that *at most one bid* of each bidder can be accepted.
- In general, we may think of the bidding language as determining a *conflict graph:* bids are vertices and edges connect bids that cannot be accepted together.
- The bidding language also determines how to compute the overall price (in most cases, including today, simply the sum).

The Winner Determination Problem

The *winner determination problem* (WDP) is the problem of finding a set of winning bids that will maximise the revenue of the auctioneer.

Example

Each bidder submit a number of bids describing their valuation. Each bid (B_i, p_i) specifies which price p_i the bidder is prepared to pay for a particular bundle B_i. The auctioneer may accept at most one atomic bid per bidder (other bidding languages are possible).

Agent 1: $(\{a, b\}, 5)$, $(\{b, c\}, 7)$, $(\{c, d\}, 6)$

Agent 2: $(\{a, d\}, 7)$, $(\{a, c, d\}, 8)$

Agent 3: $(\{b\}, 5)$, $(\{a, b, c, d\}, 12)$

What would be the optimal solution?

▶ The importance of CAs has been recognised for quite some time (in Economics), but not until very recently have algorithms that can solve realistic problem instances been developed (in Computer Science).

Recap: Complexity Theory

- Given a class of problems parametrised by their "size", how hard it is to solve a problem of size n?
- Distinguish: *time*/space *worst-case*/average-case complexity
- Problems that can be solved in *polynomial* time (P) are considered tractable, problems requiring *exponential* time (EXPTIME) not.
- Think of a problem that requires searching through a tree. If you are lucky and go down the right branch at every branching point, you may need only polynomial time, otherwise exponential time.

  A *nondeterministic* algorithm is a (hypothetical) algorithm with an "oracle" that tells us which branch to explore next.
- NP is the class of decision problems that can be solved by such *nondeterministic* algorithms in *polynomial* time.

Recap: Complexity Theory (cont.)

- Equivalent definition: NP is the class of problems for which a candidate solution can be *verified* in (determ.) polynomial time.
- A decision problem is *NP-hard* iff it is at least as hard as any of the problems in NP.
- A decision problem is *NP-complete* iff it is NP-hard and in NP.
- We do not know whether $\mathrm{P} = \mathrm{NP}$, but strongly suspect $\mathrm{P} \neq \mathrm{NP}$.
- NP-complete problems are generally considered intractable. Unless $\mathrm{P} = \mathrm{NP}$, there can be no general algorithm solving NP-complete problems efficiently.
- As a rule of thumb, NP-completeness means that a naïve approach won't work, but a sophisticated algorithm may well give good results in practice.

Complexity of Winner Determination

The decision problem underlying the WDP is NP-complete:

Theorem 1 *Let $K \in \mathbb{Z}$. The problem of checking whether there exists a solution to a given combinatorial auction instance generating a revenue exceeding K is NP-complete.*

This has first been stated by Rothkopf *et al.* (1998).

Recall that proving NP-completeness requires us to show that the problem is both (1) *NP-hard* and (2) *in NP* ...

M.H. Rothkopf, A. Pekeč, and R.M. Harstad. *Computationally Manageable Combinational Auctions*. Management Science, 44(8):1131–1147, 1998.

Proof of NP-hardness

We are going to reduce our problem to Set Packing, one of the standard problems known to be NP-complete:

> Set Packing
> **Instance:** Collection $\mathcal{C}$ of finite sets and $K \in \mathbb{Z}$
> **Solution:** Collection of disjoint sets $\mathcal{C}' \subseteq \mathcal{C}$
> **Question:** Is there a solution $\mathcal{C}'$ such that $|\mathcal{C}'| > K$?

Given an instance $\mathcal{C}$ of Set Packing, consider the following CA:

- Goods: each item in one of the sets in $\mathcal{C}$ is a good
- Bidders: one for each set in $\mathcal{C}$ + one other bidder (called 0)
- Valuations: $v_C(R) = 1$ if $R = C$ and $v_C(R) = 0$ otherwise; $v_0(R) = 0$ for all bundles R

That is, every bidder values "its" bundle at 1 and every other bundle at 0. Bidder 0 values all bundles at 0.

Proof of NP-hardness (cont.)

Observe that not every solution to the CA immediately corresponds to a valid solution of SET PACKING: the bundles owned by individual agents may not all be sets in $\mathcal{C}$.

But: for every solution to the CA there exists a(nother) solution generating the same revenue that *does* directly correspond to a valid solution for SET PACKING — just assign any goods owned by a bidder with utility 0 to bidder 0 (this reallocation does not affect the revenue). And the revenue is equal to $|\mathcal{C}'|$.

Hence, any algorithm for the decision problem underlying the WDP can be used for SET PACKING as well. So the former must be at least as hard as the latter, *i.e.* it must be at least NP-hard.

Proof of Membership in NP

This part is in fact very easy ...

Recall that a problem belongs to NP if it is possible to verify the correctness of a potential solution in polynomial time.

This is clearly the case here: Given an allocation of goods to bidders, we can compute the revenue generated in polynomial time. The allocation then constitute a good solution iff the revenue exceeds K.

Remarks

To be precise, we have proved NP-hardness wrt. *the number of pairs of bidders and bundles with non-zero utility*. This corresponds to the number of sets involved in SET PACKING. Observe that this number itself may already be very high (exponential in the number of goods).

In practice, a bidder might not communicate a full representation of its valuation. What information does get communicated to the auctioneer depends on the choice of bidding language.

Strictly speaking, for every bidding language we need to prove a new complexity result (although we always seem to get NP-completeness).

NP-membership also holds with respect to both the number of bidders and the number of goods (a much smaller unit).

Intractable Special Cases

There are various results that show that seemingly severe restrictions of the WDP *remain NP-hard*. Our proof already shows one such result:

> *Winner determination remains NP-hard if each bidder only submits a single bid and assigns it a price of 1.*

Further results of this kind can be derived by exploiting the special characteristics of the NP-complete reference problem used for the reduction.

D. Lehmann, R. Müller, and T. Sandholm. *The Winner Determination Problem*. In P. Cramton *et al.* (eds.), Combinatorial Auctions, MIT Press, 2006.

Tractable Special Cases

Another line of research has tried to identify special cases for which the WDP becomes *tractable*. Such cases are characterised by specific structural properties of the valuations that bidders report.

Here is an example:

Theorem 2 (Rothkopf et al., 1998) *If the conflict graph is a tree, then the WDP can be solved in polynomial time.*

Proof sketch: Start from the leaves of the tree, going up. Accept a bid iff it has a higher price than the best combination you can get from its offspring.

M.H. Rothkopf, A. Pekeč, and R.M. Harstad. *Computationally Manageable Combinational Auctions*. Management Science, 44(8):1131–1147, 1998.

Solving the Winner Determination Problem

We have seen that the WDP is intractable (NP-complete) in its general form. Nevertheless, sophisticated search algorithms often manage to solve even large CA instances in practice.

There are two types of approaches to *optimal* winner determination in the *general* case:

- Use powerful general-purpose *mathematical programming* software (next slide)
- Develop search algorithms specifically for winner determination, combining general *AI search techniques* and domain-specific heuristics (rest of this lecture)

Other options include developing special-purpose algorithms for *tractable subclasses* (as discussed) and *approximation algorithms* for the general case (which we won't discuss here).

Integer Programming Approach

Suppose bidders submit n bids as bundle/price pairs (B_i, p_i) with the implicit understanding that the auctioneer may accept any combination of non-conflicting bids and charge the sum of the associated prices (this corresponds to the so-called OR bidding language).

Introduce a decision variable $x_i \in \{0, 1\}$ for each bid (B_i, p_i).

Then the WDP becomes the following integer programming problem:

- Maximise $\sum_{i=1}^{n} p_i \cdot x_i$ subject to $\sum_{i \in Bids(g)} x_i \leq 1$ for all goods g,
  where $Bids(g) = \{i \in [1, n] \mid g \in B_i\}$

Highly optimised software packages for mathematical programming (such as CPLEX/ILOG) can often solve such problems efficiently.

Search for an Optimal Solution

Next we are going to see how to customise well-known search techniques developed in AI so as to solve the WDP.

This part of the lecture will largely follow the survey article by Sandholm (2006).

T. Sandholm. *Optimal Winner Determination Algorithms*. In P. Cramton *et al.* (eds.), Combinatorial Auctions, MIT Press, 2006.

Search Techniques in AI

A generic approach to search uses the *state-space representation:*

- Represent the problem as a set of *states* and define *moves* between states. Given an initial state, this defines a *search tree*.
- The *goal states* are states that correspond to valid solutions.
- Each move is associated with a *cost* (or a *payoff*).
- A *solution* is a sequence of moves from the initial state to a goal state with *minimum cost* (*maximum payoff*).
- Example: route finding (states are cities and moves are directly connecting roads), but it also applies to CAs ...

A *search algorithm* defines the order in which to traverse the tree:

- Uninformed search: breadth-first, depth-first, iterative deepening
- Heuristic-guided search: branch-and-bound, A*

State Space and Moves

There are (at least) two natural ways of representing the state space and moves between states:

- Either: Define a state as a set of *goods* for which an allocation decision has already been made. Then making a move in the state space amounts to making a decision for a further good.
- Or: Define a state as a set of *bids* for which an acceptance decision has already been made. In this case, a move amounts to making a decision for a further bid.

What is the initial state? What are the goal states?

According to Sandholm (2006), the bid-oriented approach tends to give better performance in practice.

Moves in Bid-oriented Search

We represent *bids* as triples (a_i, B_i, p_i): agent a_i is offering to buy the bundle B_i for a price of p_i.

The *initial state* is when no decisions regarding bids have been made.

A *move* amounts to making a decision (accept/reject) for a new bid.

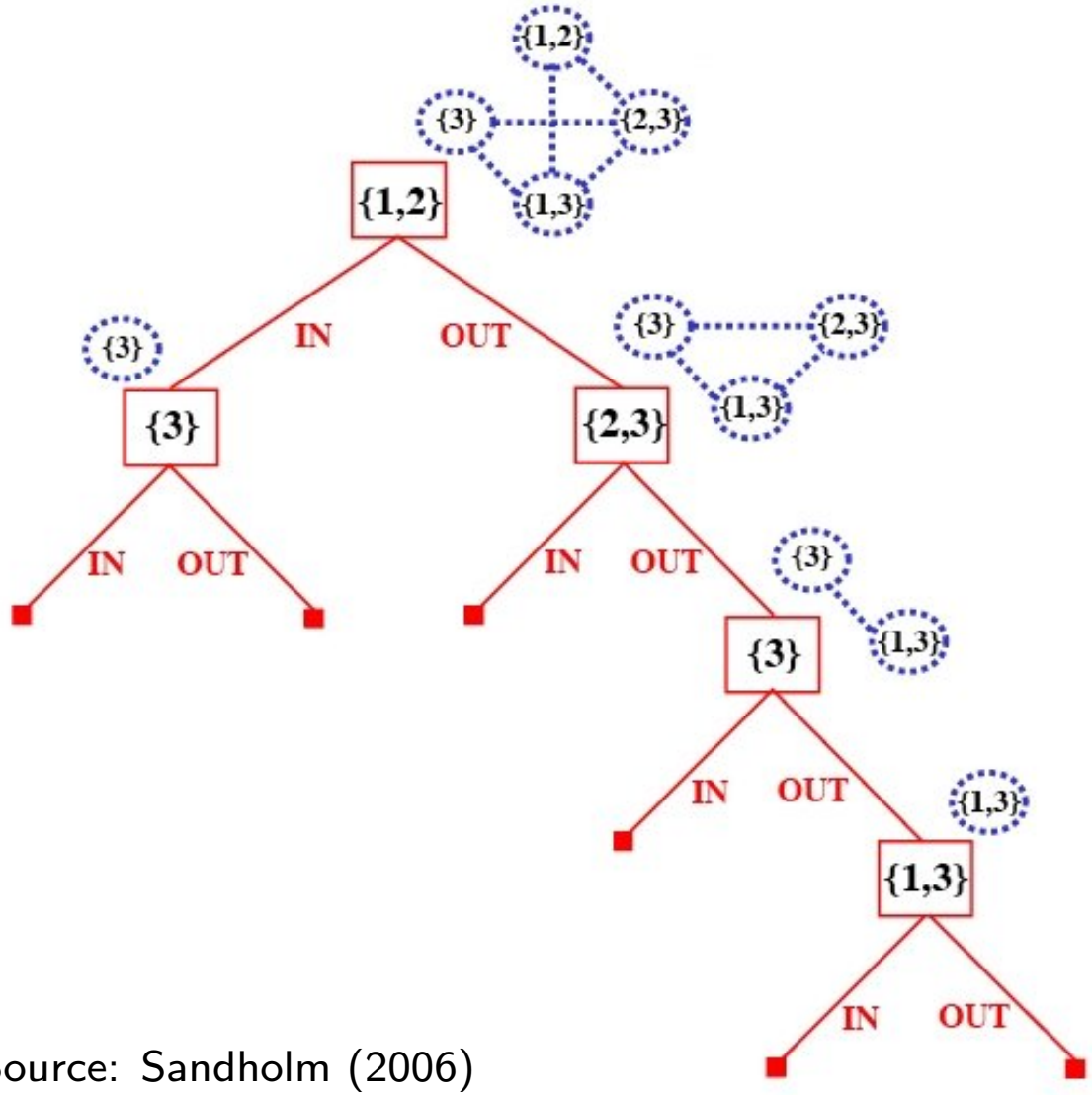
The *bidding language* specifies which bids (if any) *must* be accepted/rejected given earlier decisions. Example:

- If each agent submits a bid for every bundle with non-zero valuation, then we can accept *at most one bid per agent* (corresponding to the so-called XOR language); and only bids with *empty intersection of bundles* may be accepted.

All states are *goal states* (can stop any time, but may not be optimal).

Observe that that the search tree will be *binary* (accept or reject?).

Example



Source: Sandholm (2006)

Uninformed Search

Uninformed search algorithms (in particular depth-first search) can be used to find a solution with a given minimum revenue: traverse the tree and keep the best solution encountered so far in memory.

Optimality can only be guaranteed if we traverse the entire search tree (not feasible for interesting problem instances).

Heuristic-guided Search

In the worst case, any algorithm may have to search the entire search tree. But good *heuristics*, that tell us which part of the tree to explore next, often allow us to avoid this in practice.

For any node N in the search tree, let $g(N)$ be the revenue generated by accepting the bids accepted according to N, and only those.

We are going to need a heuristic that allows us to estimate for every node N how much revenue over and above $g(N)$ can be expected if we pursue the path through N. This will be denoted as $h(N)$. The more accurate the estimate, the better — but the only strict requirement is that h *never underestimates* the true revenue.

We are going to describe two algorithms using such heuristics:

- *depth-first branch-and-bound*
- *the A* algorithm*

Heuristic Upper Bounds on Revenue

Sandholm (2006) discusses several ways of defining a heuristic function h such that $g(N) + h(N)$ is guaranteed to be an upper bound on revenue for any path through node N.

Here is one such heuristic function:

- For each good g, compute its *maximum contribution* as:

$$c(g) \quad = \quad \max\{\frac{p}{|B|} \mid (B, p) \in Bids \text{ and } g \in B\}$$

- Then define $h(N)$ as the sum of all $c(g)$ for those goods g that have not yet been allocated in N.

This is indeed an upper bound (why?).

Can you think of an obvious refinement of this heuristic?

Depth-first Branch-and-Bound

This algorithm works like basic (uninformed) depth-first search, except that branches that have no chance of developing into an optimal solution get pruned on the fly:

- Traverse the search tree in *depth-first* order.
- Keep track of which of the nodes encountered so far would generate maximum revenue. Call that node N^*.
- If a node N with $g(N) + h(N) \leq g(N^*)$ is encountered, *remove* that node and all its offspring from the search tree.

This is *correct* (guarantees that the optimal solution does not get removed) whenever the heuristic function h is guaranteed never to underestimate expected marginal revenue (why?).

The A* Algorithm

The *A* algorithm* (Hart *et al.*, 1968) is probably the most famous search algorithm in AI. It works as follows:

- The *fringe* is the set of leaf nodes of the subtree visited so far (initially just the root node).
- Compute $f(N) = g(N) + h(N)$ for every node N in the fringe.
- Expand the node N *maximising* $f(N)$; that is, remove it from the fringe and add its (two) immediate children instead.

By a standard result in AI, A* with an *admissible* heuristic function (here: h never underestimates marginal revenue) is *optimal:* the first solution found (when no bids are left) will generate maximum revenue.

P. Hart, N. Nilsson, and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

Branching Heuristics

So far, we have not specified *which bid* to select for branching in each round (for any of our algorithms). This choice does not affect correctness, but it may affect speed.

There are two basic heuristics for bid selection:

- Select a bid with a *high price* and a *low number of items*.
- Select a bid that would *decompose* the conflict graph of the remaining bids (if available).

Tractable Subproblems

As a final example for possible fine-tuning of the algorithm, we can try to *identify tractable subproblems* at nodes and solve them using *special-purpose algorithms*.

Here are two very simple examples:

- If the bid conflict graph is *complete*, *i.e.* any pair of remaining bids is in conflict, then only one of them can be accepted.
  $\leadsto$ Simply pick the one with the *highest price*.
- If the bid conflict graph has *no edges*, then there is no conflict between any of the remaining bids.
  $\leadsto$ Accept *all* remaining bids (assuming positive valuations/prices).

Summary

- Combinatorial auctions are mechanisms to allocate a number of indivisible goods to a number of agents.
- Simple (non-combinatorial) auctions *do not suffice* if agents have non-additive valuations.
- Winner determination in CAs is *NP-complete* in the general case.
- We have seen both *special cases* that are still NP-complete, and other that are tractable.
- The WDP can be tackled using both off-the-shelf mathematical programming software and specialised *AI search techniques*.
- Our criterion for optimality has been *maximum revenue*. Alternatively, we could try to optimise wrt. a social welfare ordering (observe that revenue and utilitarian social welfare coincide in case bidders submit true and complete valuations).

References

The main reference on combinatorial auctions is the recent book edited by Cramton, Shoham, and Steinberg:

- P. Cramton, Y. Shoham, and R. Steinberg (eds.). *Combinatorial Auctions*. MIT Press, 2006.

Of particular relevance to this lecture are the following two chapters:

- D. Lehmann, R. Müller, and T. Sandholm. *The Winner Determination Problem* (Chapter 12).
- T. Sandholm. *Optimal Winner Determination Algorithms* (Chapter 14).

The paper by Rothkopf *et al.* discussing tractable instances of the WDP is one of the earliest examples of work on the computational aspects of CAs:

- M.H. Rothkopf, A. Pekeč, and R.M. Harstad. Computationally Manageable Combinational Auctions. *Management Science*, 44(8):1131–1147, 1998.

What next?

Today we have looked into the *computational* and the *algorithmic* aspects of combinatorial auctions. Next week we are going to deal with the *game-theoretical* side of combinatorial auctions:

- *Mechanism Design*

Following this, we will look into the representation of preferences in combinatorial domains in general, and into *bidding languages* for combinatorial auction in particular.