Algorithms for Combinatorial Auction

20602 - Computer Science (Algorithms)

Andrea Teruzzi

May 26, 2022

Table of Contents

Table of Contents

Problem Statement

Combinatorial Auction Problem (CAP), [NRTV07]

Let $\boldsymbol{M}$, with $|M| = m$, a set of items to be sold to n bidders. Then we define

$$\boldsymbol{v_i} : S \subseteq M \to v_i(S) \in \mathbb{R}$$

as the **valuation** function for the i-th bidder, defined for each bundle of items S.
Two common assumptions on v_i:

1. $v_i(\varnothing) = 0$ (**Normalized**)
2. $S \subseteq T \subseteq M \Rightarrow v_i(S) \leq v_i(T)$ (**Monotone**)

The objective of the problem is to find an *allocation* $S_1 \cdots S_n$ with $S_i \cap S_j = \varnothing$ for every $i \neq j$ that maximize the **common welfare**, namely:

$$\max_{S_1 \cdots S_n} \sum_i v_i(S_i)$$

Main problems for CAP

Examples of CAP problems:

- *Spectrum auctions*
- *Land Auctions*
- *Logistic optimization*

The main issues of the problem that we need to take care of are:

1. The optimization problem could be **computationally hard**.
2. The input size is exponential, indeed each evaluation function v_i requires 2^m estimates to be well defined, it is a **combinatorial problem**.
3. How to **design an efficient auction**?
4. How to take into account the **strategic behaviour** of bidders?

Table of Contents

Single-Minded Case

Let us consider a **simplification of the CAP**, in particular we impose a restriction on the valuation functions:

Definition

We say a *valuation* v to be **single minded** if there exists a bundle of items S^* and a value $v^* \in \mathbb{R}$ s.t.:

$$v(S) = \begin{cases} v^* & \text{if } S \supseteq S^* \\ 0 & \text{otherwise} \end{cases}$$

Single minded valuations are very simply represented and the algorithmic allocation problem is given by:

INPUT: (S_i^*, v_i^*) for each bidders $i = 1 \cdots n$

OUTPUT: A winner subset $\boldsymbol{W} \subseteq \{1, \cdots, n\}$ such that $S_i \cap S_j = \varnothing$ for every $i \neq j$

Single-Minded Case – NP-hardness

Proposition

The allocation problem among single-minded bidders is **NP-Hard**

Proof

Consider the **Independent Set Problem** (**ISP**), namely given a graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$ find the largest possible independent set. This problem is known to be NP-hard.
Then consider the following graph $G = (E, V)$:

- The set of edges E to be the set of items.
- The set of vertexes V to be the bidders. For vertex $i \in V$, we will have the desired bundle of i to be the set of adjacent vertices, namely $S_i^* = \{e \in E : i \in e\}$ and the value will be $v_i^* = 1$.

Now notice that a set W of winner in the CAP problem satisfies $S_j^* \cap S_i^* = \varnothing$ for every $i \neq j \in W$ if and only if the set of vertices is an independent set in G. The social welfare of W is exactly the size of the independent set in G. $\square$

Single-Minded Case – Example

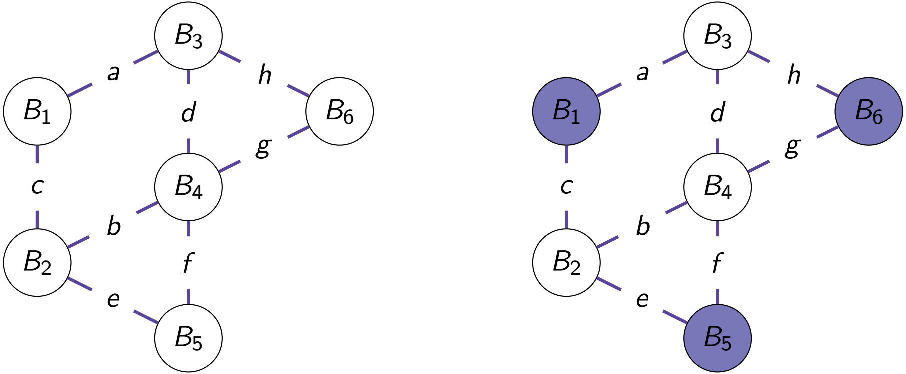

Figure: Example of CAP adapted to ISP

Table of Contents

Bidding Languages

With respect to the problem of representing bidders valuations, we have to consider the following problems:

- A naive approach asking for a valuation for each set would require a real value for each $\mathbf{2^m - 1}$ **non-empty set** for each bidder, which could be computationally unmanageable even with few items.
- We are looking for **bidding languages** that allow bidders to encode **succinctly** and **effectively** their valuations and send them to the auctioneer.
- In designing bidding languages we face an **expressiveness–simplicity tradeoff**.

Commonly used bidding languages are:

1. **OR bid**
2. **XOR bid**
3. **OR/XOR bid**

Bidding Languages – OR bids

Common bidding languages are combinations of **atomic bids**. This simple evaluations are in the form $(\boldsymbol{S}, \boldsymbol{p})$, meaning an offer of p monetary units for any bundle T, with $T \supseteq S$.

OR bids

OR language considers different bids as totally independent. Given an OR valuation for the $j-th$ bidder $v_j = (S_1, p_1) OR \ldots OR (S_k, p_k)$, the valuation for the bundle S is:

$$v(S) = \max_W \sum_{j \in W} p_i$$

where W is valid collection of pairs, meaning for all $i \neq j \in W, S_i \cap S_j = \varnothing$.

OR can represent only **superadditive** valuations, namely:

$$v(S \cup T) \geq v(S) + v(T) \quad \forall S \cap T = \varnothing$$

Bidding Languages

XOR bids

XOR language considers different bids as totally mutually exclusive. Given a XOR valuation for the $j-th$ bidder $v_j = (S_1, p_1) XOR \dots XOR(S_k, p_k)$, the valuation for the bundle S is:

$$v_j(S) = \max_{i | S_i \subseteq S_n} p_i$$

XOR can directly represent **unit demand** valuations of this kind:

$$v(S) = \max_{j \in S} v(\{p_i\})$$

and thus it can represent **every valuations**, but with bids of **exponential size**!

It is possible to form general combinations of **OR/XOR**:

$$\boldsymbol{e.g.}\ \ v(S) = (u) OR(\{d\}, 5) \\ = ((\{a, b\}, 3) XOR(\{c\}, 2)) OR(\{d\}, 5)$$

Bidding Languages – Example

e.g. $v(S) = (\{a, b\}, 3) OR (\{c, d\}, 5)$

$v(\{a, c\}) = 0$

$v(\{a, b\}) = 3$

$v(\{a, b, c, d\}) = 8$

$u(S) = (\{a, b\}, 3) XOR (\{c, d\}, 5)$

$u(\{a, c\}) = 0$

$u(\{a, b\}) = 3$

$u(\{a, b, c, d\}) = 5$

$w(S) = (\{a, b, \boldsymbol{D_1}\}, 3) OR (\{c, d, \boldsymbol{D_1}\}, 5)$

$w(\{a, c\}) = 0$

$w(\{a, b\}) = 3$

$w(\{a, b, c, d\}) = \mathbf{5}$

We call this formulation $\mathbf{OR}^*$, defined on $M \cup D$, with D adequate set of dummy variables.

Bidding Languages – Closing words

OR bids can represent **only superadditive valuations**, while **XOR** can represents **every valuation**, however for additive evaluations they need a **bid of exponential size**.

Proposition

Any valuation OR/XOR of size s can be represented by OR^* bids of size s using at most s^2 dummy items.

OR^* is a very appealing bidding language:

- OR^* bids look like a regular OR on a larger set of items.
- *OR* looks at an allocation algorithm just like a collection of **atomic bids from different players**. We can use the **same algorithms for single-minded bids** (it does not matter the number of bidders).

Table of Contents

Integer programming formulation of CAP

CAP ILP, [VV03]

Let N be the set of n bidders and M the set of m items. For every subset $S \subseteq M$ let $b_j(S)$ be the bid of the agent $j \in N$ for S. Let $b(S) = \max_{j \in N} b_j(S)$ and $x_S = 1$ when the set S is accepted, while $x_S = 0$ when the set is refused.
Then the CAP problem can formulated as the following integer program:

$$\max \sum_{S \subset M} b(S) x_S$$

$$s.t. \quad \sum_{S \ni i} x_S \leq 1 \quad \forall i \in M$$

$$x_S \in \{0, 1\} \quad \forall S \subset M$$

Integer programming formulation of CAP

OR* bids give the possibility to **express general problems in term of atomic bids** as in the single minded case.
Consider this **example**:

$$\mathbf{x} = \begin{bmatrix} x_{\{a\}} & x_{\{b\}} & x_{\{c\}} & x_{\{a,b\}} & x_{\{a,b,c\}} \end{bmatrix}^\mathsf{T}$$
$$\mathbf{b} = \begin{bmatrix} \max_{j\in N} b_j\{a\} & \max_{j\in N} b_j\{b\} & \max_{j\in N} b_j\{c\} & \max_{j\in N} b_j\{a,b\} & \max_{j\in N} b_j\{a,b,c\} \end{bmatrix}^\mathsf{T}$$

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccccc} \{a\} & \{b\} & \{c\} & \{a,b\} & \{a,b,c\} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

$$\max \mathbf{b}^\mathsf{T}\mathbf{x}$$
$$\text{s.t} \quad \mathbf{A}\mathbf{x} \leq \mathbf{1}$$
$$x_i \in \{0,1\} \ \forall i$$

Integer programming

The **integer programming** is known to be **NP-hard**. However, in literature there are known several ways to tackle this problem :

1. **Solvable instances**
   - *Totally unimodular matrix*
   - *Balanced matrix*
2. **Approximations**
   - *Worst case analysis*
   - *Probabilistic analysis*
3. **Exact methods**
   - *Branch and bound*
   - *Cutting planes*
   - *Branch and cut*

The **combinatorial nature** of the problem, combined with the fact that general instances of the problem are **not polynomial** leads to a **very hard framework**(e.g. $2^{20} \approx 10^6$ columns).

Table of Contents

Approximation Single-Minded Bidders

Algorithm Greedy Mechanism for Single-Minded Bidders

Input: Ordered set of single minded bids such that $\frac{v_1^*}{\sqrt{|S_1^*|}} \geq \frac{v_2^*}{\sqrt{|S_2^*|}} \geq \cdots \geq \frac{v_n^*}{\sqrt{|S_n^*|}}$

1: $W \leftarrow \varnothing$
2: **for** $i = 1$ to N **do**
3:     **if** $S_i^* \cap \left(\bigcup_{j \in W} S_j^*\right) = \varnothing$ **then**
4:         $W \leftarrow W \cup i$
5:     **end if**
6: **end for**

Output: The set of winners W.

- Using OR^* as bidding language, we can apply this algorithm to the CAP in ILP form and not only to Single-Minded Bidders ($i.e. \, |S_j^*| = \sum_{i \in m} aij$).
- It is **efficiently computable** in polynomial time.

Greedy Mechanism for Single-Minded Bidders

Proposition

The Greedy mechanism for Single-Minded Bidders **achieves a $\sqrt{m}$ approximation**. Namely, for the allocation OPT with the maximum value of $\sum_{j\in OPT} v_j^*$:

$$\sum_{j\in OPT} v_j^* \leq \sqrt{m} \sum_{j\in W} v_i^*$$

with W the output of the greedy algorithm.

Proof

For each $i \in W$ let $OPT_i = \{j \in OPT, j \geq i | S_i^* \cap S_j^* \neq \varnothing\}$. Then $OPT \subseteq \bigcup_{i\in W} OPT_i$ and thus it is enough to prove:

$$\forall i \in W, \sum_{j\in OPT_i} v_j^* \leq \sqrt{m} v_i^*$$

Greedy Mechanism for Single-Minded Bidders II

Note for every $j \in OPT_i$ $v_j^* \leq \frac{v_i^* \sqrt{|S_j^*|}}{\sqrt{|S_i^*|}}$. Then, we can sum over all $j \in OPT_i$:

$$\sum_{j \in OPT_i} v_j^* \leq \frac{v_i^*}{\sqrt{|S_i^*|}} \sum_{j \in OPT_i} \sqrt{|S_j^*|} \tag{1}$$

Using the Cauchy-Schwarz inequality we can bound the second member of the RHS of (1):

$$\begin{aligned} \sum_{j \in OPT_i} \sqrt{|S_j^*|} \cdot 1 &\leq \sqrt{\sum_{j \in OPT_i} 1} \sqrt{\sum_{j \in OPT_i} |S_j^*|} \\ &= \sqrt{|OPT_i|} \sqrt{\sum_{j \in OPT_i} |S_j^*|} \end{aligned}$$

Greedy Mechanism for Single-Minded Bidders III

From the definition of OPT_i it follows $|OPT_i| \leq |S_i^*|$ and since OPT is an allocation $\sqrt{\sum_{j \in OPT_i} |S_j^*|} \leq \sqrt{m}$. We can substitute these two results in the Cauchy-Schwarz inequality obtaining:

$$\sum_{j \in OPT_i} \sqrt{|S_j^*|} \leq \sqrt{|OPT_i|} \sqrt{\sum_{j \in OPT_i} |S_j^*|}$$
$$\leq \sqrt{|S_i^*|} \sqrt{m}$$

Plugging this result in (1) we obtain:

$$\sum_{j \in OPT_i} v_j^* \leq \frac{v_i^*}{\sqrt{|S_i^*|}} \sum_{j \in OPT_i} \sqrt{|S_j^*|}$$
$$\leq \sqrt{m} v_i^* \quad \square$$

Greedy Mechanism for Single-Minded Bidders

```julia
function greedy_solver(M::Vector{String}, S::Dict{Vector, Int64})
    # M is the ground set of items
    # S is a dictionary with some valuations
    l = sort(collect(keys(S)), by = x -> S[x]/sqrt(size(x)[1]), rev=true)
    W=[]
    z=0
    for set in l
        if intersect(M, set) == set
            append!(W, [set])
            setdiff!(M,set)
            z+=S[set]
        end
    end
    return W, z
end
```

Figure: Greedy algorithm in Julia

Table of Contents

Exact Solutions for ILP

For obtaining exact solutions (or at least better approximations) we need more involved algorithms, mainly based on the simplex algorithm.

- **JuMP** is package in **Julia** used for solving optimization program. It uses algebraic modeling languages, such as **HiGHS** for designing and solving efficiently LP and ILP.
- For solving LP it uses a parallelized version of the **revised dual simplex** algorithm.
- It uses branch and bound algorithms to solve ILP when there are few columns.
- Some applications can be found at `https://github.com/andreateruzzi/combinatorial_auction_ILP`.

Exact Solutions for ILP

```julia
function cap_solver(M::Vector{String}, S::Dict{Vector, Int64}, optmizer, display::Bool=false)

    # M is the ground set of items
    # S is a dictionary with some valuations

    #model construction
    l = collect(keys(S))
    model = Model(optmizer)
    if display==false
        set_silent(model)
    end
    @variable(model, x[l] >= 0, Bin)
    @objective(
        model,
        Max,
        sum(S[s] * x[s] for s in l),
    );

    for e in M
        V = [s for s in l if e in s]
        intake = @expression(
            model,
            sum(x[subset] for subset in V),
        )
        @constraint(model, intake <= 1)
    end

    #solving  model
    optimize!(model)
end
```

Figure: Integer program solver in Julia

Exact Solutions for ILP – Example

$$\textbf{INPUT}: \mathbf{M} = \{a, b, c\} \rightarrow v_1 = (\{a\}, 3) OR (\{b\}, 3) OR (\{c\}, 3)$$
$$\rightarrow v_2 = (\{a, b\}, 5) OR (\{b\}, 4) OR (\{a, b, c\}, 6)$$
$$\rightarrow v_3 = (\{c\}, 5)$$

$$\mathbf{x} = \left[\begin{array}{ccccc} x_{\{a\}} & x_{\{b\}} & x_{\{c\}} & x_{\{a,b\}} & x_{\{a,b,c\}} \end{array}\right]^{\mathsf{T}}$$
$$\mathbf{b} = \left[\begin{array}{ccccc} 3 & 4 & 5 & 5 & 6 \end{array}\right]^{\mathsf{T}}$$

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccccc} \{a\} & \{b\} & \{c\} & \{a,b\} & \{a,b,c\} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

$$\max z = \mathbf{b}^{\mathsf{T}}\mathbf{x}$$
$$\text{s.t} \quad \mathbf{Ax} \leq \mathbf{1}$$
$$x_i \in \{0, 1\} \ \forall i$$

$$\textbf{OUTPUT}: \mathbf{x}^* = \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \end{array}\right]^{\mathsf{T}}$$
$$z^* = 12$$

Bibbliography I

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
Introduction to Algorithms, Third Edition.
The MIT Press, 3rd edition, 2009.

Iain Dunning, Joey Huchette, and Miles Lubin.
Jump: A modeling language for mathematical optimization.
SIAM Review, 59(2):295–320, 2017.

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani.
Algorithmic Game Theory, Chapter 11.
Cambridge University Press, USA, 2007.

Sven Vries and Rakesh Vohra.
Combinatorial auctions: A survey.
INFORMS Journal on Computing, 15:284–309, 08 2003.