A Flexible Model for Tree-Structured Multi-Commodity Markets

Per Carlsson
EnerSearch and Uppsala University,
Sweden

Arne Andersson
Uppsala University,
Sweden

Abstract

In this paper we study tree-structured multi-commodity, multi-unit markets. The concept is a way to handle dependencies between commodities on the market in a tractable way. The winner determination problem of a general combinatorial market is well known to be NP-hard.

It has been shown that on single-unit single-sided combinatorial auctions with tree-structured bundles the problem can be computed in polynomial time. We show that it is possible to extend this to multi-unit double-sided markets. Further it is possible to handle the commodities of a bundle not only as complements but as perfect substitutes too. Under certain conditions the computation time is still polynomial.

Keywords: multi commodity markets, electronic markets, computational markets, equilibrium markets, resource allocation, power markets, bandwidth markets, computational complexity.

1 Introduction

Actors on markets where multiple commodities are traded simultaneously typically have dependencies between the traded commodities (complementarities and substitutability). Hence, market mechanisms that support the expression of these dependencies are highly interesting from an actors' perspective. Traditional simultaneous multi-commodity markets, single sided auctions as well as two-sided markets give no or very limited support for the expression of dependencies between commodities. Examples of situations where this is an issue are day-ahead power markets (e.g. the Nordic NordPool market) and the often discussed radio frequency auctions (as run by e.g. the FCC)[12].

The economical efficiency of a competitive market depends on the actors' possibilities to express their true valuations. The more they are able to express their valuations in terms of complementarities and substitutability between commodities, the higher is the potential efficiency of the market.

To reach optimal economical efficiency on the market the ideal would be to allow all possible combinatorial bids expressing synergies and substitutability. Given that the participants act price-takers, this would open for to compute an optimal price vector and corresponding allocations (assuming that they exist). The bad news is that this gives a computational problem that is known to be NP-hard, i.e. the worst case computational complexity is too high. Hence, there is a conflict or tradeoff between economical and computational efficiency.

Since the winner determination problem of the general combinatorial auction is computationally hard, a number of simplifications and special cases are studied in the literature [7, 12, 13]. One natural case is tree-structured markets where any two bundles either are disjoint or one is a proper subset of the other. As pointed out by Rothkopf et al [12], for a single-unit single-sided auction this type of bid structure can be handled with a polynomial time algorithm.

In this paper we study more general tree-structured market mechanisms[1] handling multi-commodity, multi-unit markets. As in the market described by Rothkopf et al, we allow for complementarities to be expressed for tree-structured bundles. However, we also allow the following more general types of bids:

- *Multi-unit bidding.* A bid on a predefined bundle is expressed as a demand curve,[2] which implies a multi-unit market.
- *Two-sided market.* A demand curve can be buy, sell, or both.
- *Bidding for substitutes.* Bidding for substitutes (XOR-type of bids) normally implies that the winner determination becomes computationally intractable even for tree-structured bids. We show that the problem is solvable in subexponential time also when a bidder is allowed to express that

[1] We use the notion of a *market mechanism* to denote the rules of the market, whereas *market protocol* includes the behaviour of the actors on the market.

[2] The bid format is attractive when markets are distributed, as it reduces communications significantly.

the commodities in a bundle are perfect substitutes, i.e. the volume in the bid's demand curve is on the total allocation over the commodities within the bundle.

1.1 Motivation for Tree-Structured Markets

There are a number of reasons for studying markets with tree-structured bidding. Two of these arguments are:

- *Expressibility vs. complexity.* Tree-structured bids are signifficantly more expressive than plain single-commodity bids. In our case, we manage to handle both complemtens and substitutes. Compared to arbitrary combinatorial bids, the possibility to compute winner determination in polynomial time makes an important difference. All in all, we achieve a tradeoff between expressibility and computational complexity.
- *A natural structure.* A hierarchical structure of commodities is natural and easy to understand. For example, in many of today's systems for e-Sourcing, such as online B2B auctions a terminology of "lots", grouped into "categories" is very common, which clearly reflects an underlying tree-structure.

1.2 Structure

In the following section we present a small result on single-unit auctions. Then we move on to multi-unit markets in Section 3. We give a sketch of a market mechanism for tree-structured markets. After this we move to the main result of this paper, presented in Section 4. Here we present a novel approach to tree structured markets that does not only handle bundling of complementarities on a multi-unit market, but it handles substitutes too. After this we draw some general conclusions in the final section.

2 Improving the Result of Rothkopf et al

As a first observation we show how to improve the complexity of a single-unit single-sided multi-commodity auction with tree-structured bids. In such a market each commodity corresponds to a leaf, and each multi-commodity bundle corresponds to an internal node. There are no unary nodes and therefore the total number of nodes is at most $2n-1$, n being the number of commodities. Under the assumption that we know the best bid on each bundle we only need to consider at most $2n-1$ bids, and we can express the complexity in terms of n regardless of the number of bids.

Under these conditions Rothkopf et al presented an $\mathcal{O}(n^2)$ time algorithm for winner determination [12]. However, with a simple algorithm the computations can be performed in linear time. We assume this to be folklore, but since we have not found it in the literature we here provide a proof sketch.

Given a tree representing the bid structure we store one number in each node, representing the best bid value on the bundle. The winner determination is computed during two tree traversals:

1. Perform a postorder traversal of the tree, do the following at each non-leaf node: If the sum of the children values is larger than the value stored in the node, mark the node's bid as replaced and replace the value with the sum of the children.
2. Make a partial preorder traversal and do the following at each visited node: If the value in the node is not marked as replaced include it in the set of winning bids, else proceed to its child nodes.
3. Present the set of winning bids.

Both traversals require $\mathcal{O}(n)$ time.

An example is given in Figure 1. The values inside the nodes represent the highest bid on the corresponding bundle. A value to the left of the node corresponds to a replacing value found during the first traversal. The nodes of the winning combination, i.e. the topmost nodes with non-replaces values, are highlighted with a double ring (37, 12, 10, 7, and 50 summing to 116).

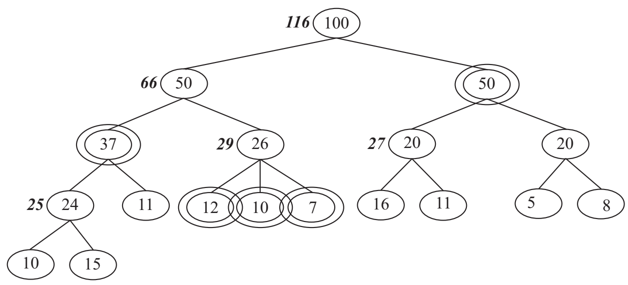

Figure 1. Illustration of the linear winner determination algorithm for single unit single sided markets.

We summarise this with the following observation:

Observation 2.1 *The winner determination problem of tree-structured single-unit single-sided markets can be solved in time linear in the number of commodities.*

3 Multi-Unit with Demand Curves

On a two-sided multi-commodity, multi-unit market with demand curves the opportunities of the bidder are richer than in the above auction. Furthermore, computationally it presents us with a more complex problem. Here, the bidder expresses his demand for individual commodities as well as for predefined bundles as a function of price, and supply is expressed as negative demand. (A bundle bid expresses the same demand for all commodities of the bundle.) We assume that each demand function is a piecewise linear function over a predefined, evenly distributed set of prices. Furthermore, we assume that the function is continuous and decreasing in price.

A set of equilibrium prices can be established using a resource oriented algorithm. As with the single unit auction we build the bundle tree, with each node representing a predefined bundle. All nodes hold an aggregate excess demand function based on the demand of all bidders. The root of the tree corresponds to a bundle including all commodities. We determine the total demand for this bundle in a binary search. At each step of the search, we recursively solve the complete problem for each sub-tree. The goal is to establish a set of prices for the individual commodities, that renders an equilibrium with respect to the demand expressed in all nodes of the tree.

Example. Consider a two commodity market where it is possible to express a (positive or negative) demand for each commodity separately, and for the combination, Figure 2. This creates three submarkets. Treated separately, the equilibrium prices of the three submarkets are $\{4,\ 4.5,\ 3\}$ (dashed line *'a'* in the figures). This set of prices is clearly not efficient as the average price of the commodities is higher than the bundle price. Instead, asserting a negative excess demand of 1 on the bundle, balanced by a corresponding positive demand for each single commodity (*'b'* in the figures), we end up with a new price vector $\{3,\ 4,\ 3.5\}$, where the average of the single commodities equals the bundle price, i.e. an optimal set of prices. □

If the bundle tree e.g. is a balanced binary tree the height of the tree is $\log n$. If the size of the search space over the above excess demand is s, the binary search of a node requires $\lceil \log(s+1) \rceil$ search steps. The total complexity satisfies:

$$T(1) = \lceil \log(s+1) \rceil$$
$$T(n) = \lceil \log(s+1) \rceil \left(2T\left(\frac{n}{2}\right)\right) = n \lceil \log(s+1) \rceil^{1+\log n} = \mathcal{O}\left(n^{1+\log\lceil \log(s+1) \rceil} \log s\right),$$

where $T(n)$ is the total number of comparisons and $\lceil \log(s+1) \rceil$ is the number of comparisons in one binary search. In the above we assume n to be an even power of two. From this we derive that the time complexity is polynomial given that s is constant. This gives the following theorem:

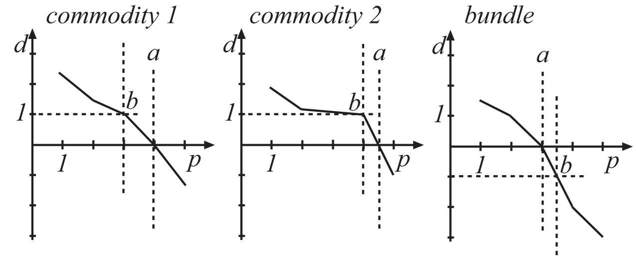

Figure 2. A small market example, determination of equilibrium prices. For each submarket *'a'* is the local equilibrium price, and *'b'* indicates the equilibrium price after adjusting so that the average price of the single commodities equals the bundle price.

Theorem 3.1 *The worst case time complexity of solving an n commodity multi-commodity, multi-unit market with demand curves structured as a balanced tree is*

$$\mathcal{O}\left(n^{1+\log\lceil \log(s+1) \rceil} \log s\right)$$

where s is the size of the excess demand space.

Proof. The complexity is given by the recursion above. □

Details are left to Section 4 where we discuss a mechanism that also handles substitutes.

4 Main Result

4.1 Allowing Substitutes

Finally we add the possibility to bid for perfect substitutes. That is, the volumes in a bid are on the total allocation over the commodities in the bundle.

The intuition behind is that when the commodities within the bundle are perfect substitutes for the bidder, a buyer prefers to pick among the lowest price commodities, and a seller prefers to sell at highest possible price. In a practical situation a bidder typically submits this type of bids as a complement to his other bids. The practical usefulness of this type of bids is clear but it introduces a number of computational problems.

The rest of this paper is devoted to solving the problems of a tree-structured market allowing bids on substitutes, a market mechanism — the CONSEC[3] mechanism — Figure 3, and an algorithm description.

[3] CONSEC, a tree-structured market mechanism for e.g. a set of CONSECutive time periods.

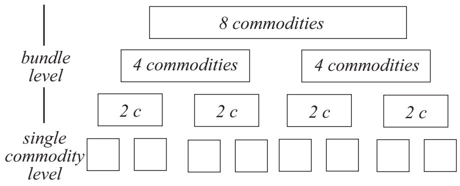

Figure 3. The tree-structured market mechanism that is presented in this paper. In this example a market with eight commodities organised in a binary structure.

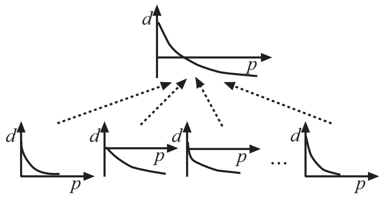

Figure 4. The demand functions of each separate bidding track are aggregated into one, giving full information on supply and demand of the system (the aggregation is a simple summation of all curves). Supply is expressed as negative demand.

4.2 Main Idea

A bidder on a CONSEC market could submit multiple bids according to the following:

1. *single bids*: demand curve for each commodity,
2. *bundle bids*: demand curve for a bundle, the same volume for each included commodity,
3. *substitute buy bids*: a curve describing a consumer demand that could be arbitrarily distributed over a bundle (the consumer preference is to buy at lowest price),
4. *substitute sell bids*: symmetric to the substitute buy bids.

Demand is assumed to be continuous and decreasing in price.

The tree can be organised in different ways, with arbitrary degree in each node (though unary nodes do not make sense, and hence the maximum number of nodes is limited by $2n-1$). As an example consider a market with n commodities, where n is an even power of two. If this market is organised as a binary tree structure of bundles, each bidder can give $4n-3$ different bids (demand functions), hence we say that there are $4n-3$ *bidding tracks* (one single track for each commodity/leaf node, and three tracks for each bundle/internal node, type one and type two – four in the list above, respectively).

Please note that all demand functions of a track may be aggregated into a single function giving the excess demand of the track as a function of price, see Figure 4. This aggregation is outside the scope of this paper, standard techniques are presented in e.g. [5, 6, 1].

The four bid types allow for a fairly flexible market. Compared to a fully implemented combinatorial market, the single bids are similar, bundle bids correspond to traditional combinatorial bids expressing synergies, and the substitute bids corresponds to XOR bids.

As shown below, although we combine single bids, bundle bids, and XOR-type of bids, the market can be handled optimally in a computationally efficient way.

4.3 Problem Formulation

Given a market with the bid types presented above, compute

- a price p_i^* for each commodity i, and
- an allocation for the substitute bids,

that renders an equilibrium, i.e. the excess demand for each commodity on the market as a whole is zero, but it might well be non-zero on individual bidding tracks.

With demand expressed as above, Section 3, the bids of the separate tracks are aggregated giving a set of demand functions, one for each track. These functions give full information on supply and demand and an equilibrium can be calculated without any further communication.[4]

4.3.1 Definitions

The market is organised in a tree structure, hence it is convenient to refer to single commodities as leaf nodes and bundles as internal nodes (or bundle nodes). We define the following (aggregate) demand functions describing the demand of all bidding tracks:

Definition. 4.1 *Let*

- $d_j(p)$ *or* d_j *be the main demand function of node* j *as a function of a price* p,
- $b_j(p)$ *or* b_j *be the buying demand with the commodities of bundle node* j *viewed as perfect substitutes, and*
- $s_j(p)$ *or* s_j *be the corresponding selling demand.*

Note that b_j and s_j are only defined for bundle nodes.

We define the following demand that is expressed in the interaction between nodes in the market tree:

[4]This holds under the assumption that the optimal solution is within the price span of the given bids.

Definition. 4.2 *Let*

- d_j^{ch} *be the (positive or negative) demand of node j that it expresses towards its children, and that they all have to meet,*
- d_j^{min} *be the (negative) excess demand of node j induced by the minimum price set by the parent, that it expresses towards the parent, and*
- d_j^{max} *the corresponding (positive) excess demand induced by the maximum price set by the parent.*

Finally we define the following price notations:

Definition. 4.3 *Let*

- p_j^{min} *be the minimum price imposed by node j on its child nodes,*
- p_j^{max} *the corresponding maximum price,*
- p_j *be the price of node j, for a bundle node it is defined as the average price of the children.*

We now move on to a formal specification of the problem, the existence of an equilibrium, and in Section 4.4 an algorithm that computes an equilibrium.

4.3.2 Formal Specification of the Problem

The problem is to determine a price vector p^*

$$p^* = \{p_1^*, p_2^*, \ldots, p_n^*\} \tag{1}$$

s.t.
$\forall$ leaf nodes j with parent i:

$$d_j\left(p_j^*\right) + d_i^{ch} - d_j^{min} - d_j^{max} = 0, \tag{2}$$
$$p_i^{min} \le p_j^* \le p_i^{max} \tag{3}$$

$\forall$ bundle nodes j with parent i and children v, w:

$$d_j\left(p_j^*\right) + d_i^{ch} - d_j^{ch} = 0, \tag{4}$$
$$b_j + d_v^{min} + d_w^{min} - d_j^{min} = 0, \tag{5}$$
$$s_j + d_v^{max} + d_w^{max} - d_j^{max} = 0, \tag{6}$$

if a bundle node has more than two children the equations are adjusted to take all of them into account. Figure 5 and 6 illustrate the equations (without the notational assumption that a bundle node has no more than two children).

It should be noted that for bundle nodes the constraint that $p_i^{min} \le p_j^* \le p_i^{max}$ is secured by the definition of p_j as the average price of its children, with no child price outside the boundaries.

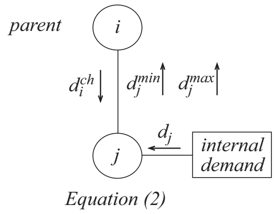

Figure 5. The demand for resources related to a leaf node j of a market tree. The arrows indicate the origin of each demand. With parent node i, d_i^{ch} is a demand that node j has to meet; if either the minimum or the maximum price influences the node, the induced excess demand is expressed towards the parent. With this the node is in equilibrium, c.f. Eq. (2).

4.3.3 The Existence of an Equilibrium

The nodes of the market tree interact with both higher and lower levels of the hierarchy to enhance the market outcome. The root of the system and leaf nodes are special cases with limited interaction.

In order to prove the existence of an equilibrium for the system, we prove the existence of an equilibrium for an arbitrary node.

Lemma 4.1 *For an arbitrary node j with parent i, given*

- *a (positive or negative) demand expressed by the parent d_i^{ch},*
- *a minimum price p_i^{min}, and*
- *a maximum price p_i^{max}*

there exists

- *a price p_j,*
- *a (negative) excess demand d_j^{min} generated by the minimum price, left for the parent to meet, and*
- *a corresponding (positive) demand d_j^{max} generated by the maximum price,*

setting the node in balance with respect to the input variables. As a side effect the system rooted in the node is set in balance too.

The proof is given in Appendix A. From this lemma we get the following theorem on the existence of an equilibrium.

Theorem 4.1 *There exists a price vector $p^* = \{p_1^*, p_2^*, \ldots, p_n^*\}$, rendering an equilibrium on a* CONSEC *market.*

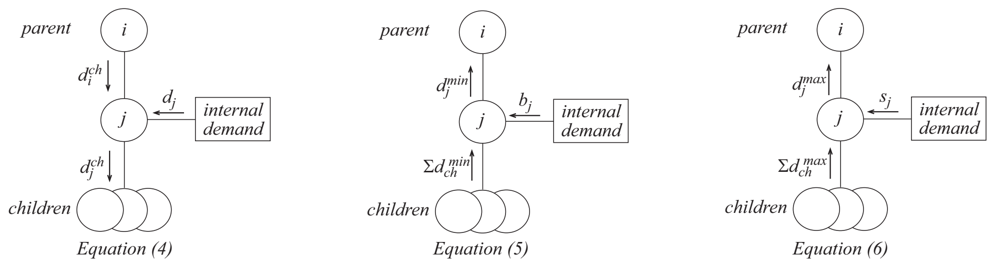

Figure 6. The demand for resources related to a bundle node as given by Eq. (4) – Eq. (6). The arrows indicate the origin of each demand. To maintain an equilibrium the internal bundle demand, d_j together with the demand of this node on its children d_j^{ch} has to meet the demand of the parent node, d_i^{ch}. Further, the children's demand, $\sum d_{ch}^{min/max}$, induced by $p_j^{min/max}$ is balanced by $(b_j - d_j^{min})$ and $(s_j - d_j^{max})$, respectively. If $p_j^{min} > p_i^{min}, d_j^{min} = 0$, and if $p_j^{max} < p_i^{max}, d_j^{max} = 0$.

The proof is given in Appendix A. From this we move on to the algorithm that we suggest for establishment of a market equilibrium.

4.4 Algorithm

We describe the algorithm from the viewpoint of an arbitrary bundle node j with parent i. Given a triplet consisting of (i) the demand from the parent node, (ii) a minimum price, and (iii) a maximum price it returns a corresponding triplet consisting of (a) an equilibrium price, (b) a negative excess demand induced by the minimum price, and (c) a positive excess demand induced by the maximum price. The algorithm mainly consists of a nested exponential-and-binary search.[5] Each exponential-and-binary search starts at the value of the last iteration.

Algorithm 4.1 (Equilibrium Search) :
double[] findEquilibrium$(d_i^{ch}, p_i^{min}, p_i^{max})\{$
do an exponential-and-binary search in d_j^{ch}
starting from the value of the previous call,
at each step in the search:$\{$
do an exponential-and-binary search in p_j^{min}
starting from p_i^{min}*, and*
do an exponential-and-binary search in p_j^{max}
starting from p_i^{max}*,*
at each step of the searches:$\{$
*/*recursive call*/*
$\forall children$:
findEquilibrium$(d_j^{ch}, p_j^{min}, p_j^{max})$;
use return values for these evaluations:
$\}$ *until Eq. (5 & 6) are fullfilled*
$\}$ *until Eq. (4) is fullfilled*
return $\{p_j, d_j^{min}, d_j^{max}\}$;
$\}$

[5] In an exponential-and-binary search we start by performing an expanding inverted binary search, followed by an ordinary binary search within the boundaries defined by the first search.

In practice the breaking conditions are that the solutions are sufficiently close to optimal.

The root node and a leaf node are special cases. By definition *a leaf node* does not have any children and it does only have a single demand function. Hence it establishes an equilibrium price corresponding to the given demand from the parent node. If this price is lower than the imposed minimum price or higher than the maximum price it is adjusted and any excess demand generated by this adjustment is expressed towards its parent. To set a full market in balance *the root node* is set in balance, by definition with no demand from above and no restricting minimum and maximum prices.

4.4.1 Algorithm 4.1 Computes an Optimal Set of Prices

What we need to prove is that Algorithm 4.1 computes a set of prices as in Theorem 4.1.

Theorem 4.2 *Algorithm 4.1 correctly computes a set of prices* $p^* = p_1^*, p_2^*, .., p_n^*$ *as in Theorem 4.1, and hence it establishes an equilibrium on a* CONSEC *market.*

The proof is given in Appendix A.

4.4.2 The Complexity of Algorithm 4.1

The complexity of this market mechanism is highly dependent on the depth of the market tree. A straightforward binary search implementation that does not utilise prior knowledge soon becomes impractical as the tree grows. Even a market such as the one of Figure 7 has shown to be compuationally impractical. On the other hand algorithms that utilise expontential-and-binary search, as the one we suggest, has shown to be fast in practice when it comes to the same problem instances.

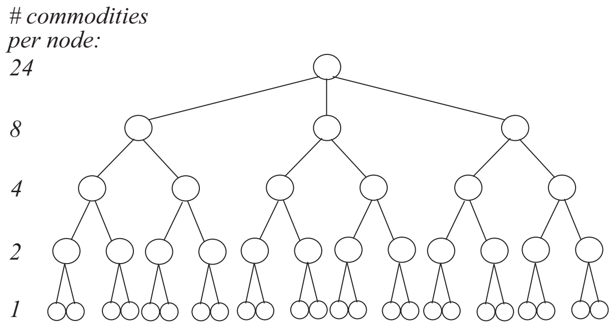

Figure 7. A market where 24 commodities are traded simultaneously, could for example be organised with three eight commodity market structures that have a common root node. A day-ahead power market is a good example of a real-world market where this fits well.

The analysis of the computational complexity presented here does not take the advantages of the exponential-and-binary search of Algorithm 4.1 into account, but is based on the simple binary search approach. When it comes to worst case behaviour, this is correct, if we would try to give a theoretical evaluation of the average behaviour it would not.

In the worst case, the establishment of a local equilibrium of a bundle node renders a full nested binary search in the three search variables, with recursive calls of the child nodes at each search step.

Let s_1 and s_2 be the size of the search spaces in volume and prices, respectively (given in a resolution such that the deviations from the true optimum are sufficiently small). Then the worst case local search cost of a bundle node is $2\lceil\log(s_1+1)\rceil\lceil\log(s_2+1)\rceil$ comparisons (one search in bundle demand, with a nested parallel search in minimum and maximum prices). The corresponding cost of a leaf node is $\lceil\log(s_1+1)\rceil$, as there is a sole search for a price corresponding to the given demand expressed by the parent node (here the work related to minimum and maximum prices is constant).

The total worst case cost depends on the depth of the market tree. The deepest tree structure that makes sense is a binary tree, where we get the recursion:

$$
\begin{aligned}
T(1) &= \lceil\log(s_1+1)\rceil \\
T(n) &= \left(2\lceil\log(s_1+1)\rceil\lceil\log(s_2+1)\rceil\right)\left(2T\left(\frac{n}{2}\right)\right) = \\
&= \left(4\lceil\log(s_1+1)\rceil\lceil\log(s_2+1)\rceil\right)\left(T\frac{n}{2}\right) = \\
&= \left(4\lceil\log(s_1+1)\rceil\lceil\log(s_2+1)\rceil\right)^{\log n}\log(s_1+1) = \\
&= \mathcal{O}\left(n^{(4+\log\lceil\log(s_1+1)\rceil+\log\lceil\log(s_2+1)\rceil)}\log s_1\right),
\end{aligned}
$$

on a binary structured market where n commodities are traded. As in Section 3, $T(n)$ is the total number of comparisons and n is assumed to be an even power of two.

This gives the following theorem on the complexity of Algorithm 4.1:

Theorem 4.3 *The worst case complexity of Algorithm 4.1 on an n commodity market structured as a balanced tree is*

$$
\mathcal{O}\left(n^{(4+\log\lceil\log(s_1+1)\rceil+\log\lceil\log(s_2+1)\rceil)}\log s_1\right)
$$

where s_1 and s_2 are the sizes of the search spaces in volumes and prices, respectively.

Proof. The complexity is given by the recursion above. □

In practice the algorithm runs significantly faster than this due to (i) the choice to utilise prior knowledge in an iteration, and (ii) the search in minimum and maximum prices only taking place when a local price has to be calculated.

4.5 Test Implementation and Design Experiences

We have written a test implementation of the algorithm in $C^{\#}$. Our goal with this test implementation was to achieve some practical experience of the algorithm [4]. The experiments show that the algorithm is sufficiently fast and suitable for handling of e.g. day-ahead electricity markets.

The implementation will be part of a market algorithm library and used for simulations of supply – demand matching in power grids within the CRISP project. Results are part of a comming project deliverable.

5 Conclusions

In this paper we present a number of results ranging from tree-structured single-unit auctions to tree-structured multi-unit multi-commodity markets. The main result is a market mechanism suitable for e.g. markets handling time dependent commodities, the CONSEC mechanism. By a number of predefined bid types, it offers useful flexibility to the bidders.

The paper presents useful abstractions, holding the combinatorial capabilities on a low level. A reason for this is to keep it easy to understand and to make it easy to convince oneself that the pricing mechanism is correct. Furthermore, there are complexity reasons with respect to communications as well as to computations to do it.

The main computational (and communicational) task of the mechanism is the aggregation of demand. With the combinatorial capabilities of the mechanism expressed as independent tracks (bids on single commodities, bundle bids and substitute bids) the computational complexity of this part does not grow more than linear in the number of bidding tracks.

A real world market setup of a large CONSEC market would likely be a highly distributed market, i.e. most of the information needed for market computation is spread over the network. Since the input to Algorithm 4.1 is aggregated demand, it is natural to distribute a part of the computation — the aggregation — over the network. By this, the communicational load is diminished radically.

The market mechanism has properties that are highly relevant in e.g. day-ahead power markets [3, 9] and bandwidth markets [10, 11]. In a power setting, the big advantage of the mechanism (compared to the power markets of today, such as the Nordic NordPool [8] and the Dutch APX [2]) comes with the possibility to set up electronic markets with a huge number of participants. When a direct market participation of a large number of presumably small size actors (formerly represented by distributors) is introduced the market outcome can become considerably more efficient. To reach this one has to enhance the possibilities for actors on the market to express dependencies and constraints between the traded time slots, we believe our mechanism to be an interesting alternative when it comes to this.

The combinatorial possibilities given by the market mechanism enriches the possibilities of the actors. While being easy to understand and computationally feasible, it scales well to markets with a huge number of participants.

Acknowledgement: This paper is based on work performed within the CRISP project (http://www.ecn.nl/crisp) financially supported by the European Commission, contract nr ENK5-CT-2002-00673:CRISP, which is gratefully acknowledged.

References

[1] A. Andersson and F. Ygge. Managing large scale computational markets. In H. El-Rewini, editor, *Proceedings of the Software Technology track of the 31th Hawaiian International Conference on System Sciences (HICSS31)*, volume VII, pages 4–14. IEEE Computer Society, Los Alamos, January 1998. ISBN 0-8186-8251-5, ISSN 1060-3425, IEEE Catalog Number 98TB100216. (Available from http://www.enersearch.se/~ygge).

[2] http://www.apx.nl.

[3] S. Borenstein, M. Jaske, and A. Rosenfeld. Dynamic pricing, advanced metering and demand response in electricity markets. (Available from http://www.ef.org), 2002.

[4] P. Carlsson. Market simulations. Technical Report 2004-046, EnerSearch and Department of Information Technology, Uppsala University, 2004. (Available from http://www.it.uu.se).

[5] T. Ibaraki and N. Katoh. *Resource Allocation Problems — Algorithmic Approaches*. The MIT Press, Cambridge, Massachusetts, 1988.

[6] N. Katoh and T. Ibaraki. Resource allocation problems. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 2, chapter Resource Allocation Problems, pages 159–260. Kluwer Academic Publisher, 1998.

[7] N. Nisan. Bidding and allocation in combinatorial auctions. In *EC'00, Proceedings of the 2nd ACM conference on electronic commerce*, pages 1–12. The Association for Computing Machinery, New York, USA, 2000.

[8] http://www.nordpool.no.

[9] R. H. Patrick and F. A. Wolak. Using customer demands under spot market prices for service design and analysis. Technical Report WO 2801-11, EPRI, 2001. (Available from http://www.rci.rutgers.edu/~rpatrick/hp.html).

[10] L. Rasmusson. Evaluating resource bundle derivatives for multi-agent negotiation of resource allocation. In J. Liu and Y. Ye, editors, *E-Commerce Agents*, number 2033 in LNAI, pages 154–165, Berlin Heidelberg, 2002. Springer Verlag.

[11] L. Rasmusson and G. Paues. Network components for market-based network admission and routing. Technical report, Swedish Institute of Computer Science, SICS, Kista, Sweden, 2002.

[12] M. H Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, August 1998.

[13] M. Tennenholtz. Some tractable combinatorial auctions. In *AAAI/IAAI 2000 Proceedings*, 2000.

A Proofs

A.1 Proof of Lemma 4.1

Leaf nodes, Eq. (2): An equilibrium in the node is obtained with a price p_j such that $d_j(p_j) + d_i^{ch} = 0$. The existence of such a p_j follows from continuous and decreasing demand.

If $p_j < p_i^{min}$ an imbalance is created when setting $p_j = p_i^{min}$. The local balance can be restored by setting $d_j^{min} = d_i^{ch} + d_j\left(p_i^{min}\right)$. Everything is symmetric in p_i^{max} and d_j^{max}.

This fulfils Eq. (2) and the node is in equilibrium, c.f. Figure 5.

Bundle nodes: The notion of an equilibrium in a bundle node is different from a leaf node. Where a leaf node has a single demand function, affected by both d_i^{ch}, p_i^{min}, and p_i^{max}, a bundle node has three different demand functions d_j, b_j, and s_j, interacting with d_i^{ch}, p_i^{min}, and p_i^{max} respectively, c.f. Figure 6.

Eq. (4): The equation expresses an equilibrium regarding bundle demand (i.e. a demand for the same resource over all commodities of the bundle). In the equation d_i^{ch} is fixed. Let p_j be the average price of the children as in Definition 4.3, given d_j^{ch}. By continuous and decreasing demand we have that an increase in d_j^{ch} gives a higher p_j, and hence a lower $d_j\left(p_j\right)$. Hence there exists a price p_j, such that $d_j(p_j) + d_i^{ch} - d_j^{ch} = 0$. As p_j is given by the average price of the system rooted in the node, this system is in balance too when this price is established.

Eq. (5): The equation expresses an equilibrium regarding the effects of minimum prices and buying demand with substitutability. This demand is not affected by the bundle price p_j, but the minimum price of the child nodes.

With children v, w, there exists some lowest price p_j^{lowest} of the system rooted in node j, such that the buyer demand with substitutability of this node is in balance with the demand, $d_v^{min} + d_w^{min}$, expressed by its child nodes (independent of p_i^{min}, the minimum price given by node j's parent). Depending on whether $p_j^{lowest} > p_i^{min}$ or $p_j^{lowest} \leq p_i^{min}$, we get two cases:

1. $p_j^{lowest} > p_i^{min}$: this implies that $b_j(p_i^{min}) + d_v^{min} + d_w^{min} > 0$. Let $p_j^{min} = p_j^{lowest}$ and $d_j^{min} = 0$, and
2. $p_j^{lowest} \leq p_i^{min}$: this implies that $b_j(p_i^{min}) + d_v^{min} + d_w^{min} \leq 0$. Let $p_j^{min} = p_i^{min}$ and $d_j^{min} = b_j(p_i^{min}) + d_v^{min} + d_w^{min}$,

in both cases there exists p_j^{min} and d_j^{min} such that $b_j(p_j^{min}) + d_v^{min} + d_w^{min} = 0$.

Eq. (6): Everything is symmetric when it comes to maximum prices, Eq. (6).

The node is in equilibrium when all three of Eq. (4) – Eq. (6) hold, i.e. then the node and the system rooted in it is in balance given d_i^{ch}, p_i^{min}, and p_i^{max}.

All together this gives the lemma.

A.2 Proof of Theorem 4.1

We have from Lemma 4.1 that for a given triplet $(d_i^{ch}, p_i^{min}, p_i^{max})$, given by its parent node i, there exists a price such that Eq. (2) holds for an arbitrary leaf node, and such that Eq. (4) – Eq. (6) hold for an arbitrary bundle node. In both cases this constitutes a local equilibrium of the node.

In particular, with a zero demand from the outside on the root node (which covers all commodities), and without restricting minimum and maximum prices at the same node, we have an equilibrium of the whole market when this node is in equilibrium, as an equilibrium in one node is based on its child nodes being in equilibrium recursively.

A.3 Proof of Theorem 4.2

Leaf nodes: For an arbitrary single commodity node j with parent node i, the actions of the algorithm consists of the parent expressing a demand, d_i^{ch}, a minimum and a maximum price, p_i^{min} and p_i^{max} respectively. Based on this input the node computes a price p such that $d_j(p) + d_i^{ch} = 0$ or in practice $\left|d_j\left(p\right), -d_i^{ch}\right| < \epsilon$ for some sufficiently small $\epsilon > 0$. If $p < p_i^{min}$, $p \leftarrow p_i^{min}$ and $d_j^{min} \leftarrow d_j(p_i^{min}) + d_i^{ch}$ to compensate for this. If the maximum price is exceeded, a corresponding action is taken. By this Eq. (2) is fullfilled for the given input, and the node is in balance.

Bundle nodes: For an arbitrary bundle node j with parent node i, given a triplet $(d_i^{ch}, p_i^{min}, p_i^{max})$ the algorithm performs a three dimensional search in a nested loop. The goal is to set a new triplet $(d_j^{ch}, p_j^{min}, p_j^{max})$ for its children (with $p_j^{min} \geq p_i^{min}$ and $p_j^{max} \leq p_i^{max}$) such that Eq. (4) – Eq. (6) hold at node j for the given input triplet. As stated in Theorem 4.1, continuous and decreasing demand gives that such a triplet exists. As above, in practice the search will be ended when sufficiently close to the solution. The recursive approach gives that when the equilibrium is reached in the node, the same holds for all nodes rooted in j.

When the root node of the system is in balance, the whole system is in balance, and this concludes the proof.