J. Software Engineering & Applications, **2010, 3, 718-722**
doi:10.4236/jsea.2010.37082 Published Online July 2010 (http://www.SciRP.org/journal/jsea)

CRAB—CombinatoRial Auction Body Software System

Petr Fiala[1], Jana Kalčevová[1], Jan Vraný[2]

[1]Department of Econometrics, Faculty of Informatics and Statistics, University of Economics, Prague, Czech Republic; [2]Software Engineering Group, Faculty of Information Technology, Czech Technical University, Prague, Czech Republic.
Email: pfiala@vse.cz, kalcevov@vse.cz, jan.vrany@fit.cvut.cz

Received February 4th, 2010; revised June 17th, 2010; accepted June 19th, 2010.

ABSTRACT

Auctions are important market mechanisms for the allocation of goods and services. Combinatorial auctions are those auctions in which buyers can place bids on combinations of items. Combinatorial auctions have many applications. The paper presents the CRAB software system. CRAB is a non-commercial software system for generating, solving, and testing of combinatorial auction problems. The system solves problems by Balas' method or by the primal-dual algorithm. CRAB is implemented in Ruby and it is distributed as the file crab.rb. The system is freely available on web pages for all interested users.

Keywords: *Combinatorial Auction, Complexity, Software System, Generating, Solving, Testing*

1. Introduction

Auctions are important market mechanisms for the allocation of goods and services. Auction theory has caught tremendous interest from both the economic side as well as the Internet industry. Design of auctions is a multidisciplinary effort made of contributions from economics, operations research, software sciences, and other disciplines. The popularity of auctions and the requirements of e-business have led to growing interest in the development of complex trading models [1]. Combinatorial auctions are those auctions in which buyers can place bids on combinations of items, called bundles. This is particular important when items are complements.

CRAB system is suitable for:

- fast generating of standard combinatorial restrictions,
- adding specific restrictions,
- generating of bundle evaluations with complementarity properties,
- solving combinatorial auction problems by Balas' method or by the primal-dual algorithm,
- experimenting, and testing.

The user has to install Ruby interpreter in order to run CRAB. Needed information is at web pages http://users.fit.cvut.cz/~vranyj1/software/crab. There is also a possibility to download CRAB (more precisely the file crab.rb).

This design gives a possibility to study and extend the CRAB system, especially about the implemented models and methods, by the user. The system is available on web pages for all interested users.

2. Combinatorial Auctions

Combinatorial auctions are those auctions in which buyers can place bids on combinations of items, so called bundles. The advantage of combinatorial auctions is that the buyer can more precisely express his preferences. This is particular important when items are complements. The auction designer also derives value from combinatorial auctions. Allowing buyers more fully to express preferences often leads to improved economic efficiency and greater auction revenues. However, alongside their advantages, combinatorial auctions raise a host of questions and challenges [2]. Many types of combinatorial auctions can be formulated as mathematical programming problems.

The problem, called the winner determination problem, has received considerable attention in the literature. The problem is formulated as: Given a set of bids in a combinatorial auction, find an allocation of items to buyers that maximize the seller's revenue. Let us suppose that one seller offers a set M of m items, $j = 1, 2,\ldots, m$, to n potential buyers. Items are available in single units. A bid made by buyer i, $i = 1, 2, \ldots, n$, is defined as

$$B_i = \{S, v_i(S)\},$$

$S \subseteq M$, is a combination of items, so called bundle.

$v_i(S)$ is the valuation or offered price by buyer i for the bundle S.

The objective is to maximize the revenue of the seller given the bids made by buyers. Constraints establish that no single item is allocated to more than one buyer and that no buyer obtains more than one combination. This constraint is not necessary but there is a possibility to generate it in the CRAB system. Bivalent variables are introduced for model formulation:

$x_i(S)$ is a bivalent variable specifying if the combination S is assigned to buyer i ($x_i(S) = 1$).

The winner determination problem can be formulated as follows

$$\sum_{i=1}^{n} \sum_{S \subseteq M} v_i(S)\, x_i(S) \to \max$$

subject to $\sum_{S \subseteq M} x_i(S) \le 1, \forall\, i = 1,2,\dots,n,$

$$\sum_{i=1}^{n} \sum_{S \subseteq M} x_i(S) \le 1, \forall\, j \in M, \tag{1}$$

$$x_i(S) \ge 0, \forall\, S \subseteq M,\ \forall\, i = 1,2,\dots,n.$$

The objective function expresses the revenue. The first constraint ensures that no buyer receives more than one combination of items. The second constraint ensures that overlapping sets of items are never assigned. In the CRAB system a generalization is applied. The set of items M can be divided in p subsets P_k, $k = 1, 2, \dots, p$, called packages. All combinations of items are generated in each package. By this way all bundles are generated.

The algorithms proposed for solving combinatorial auction problems fall into two classes: exact algorithms, and approximate algorithms. The CRAB system uses Balas' method for finding of optimal solutions.

Alternatively, the primal-dual algorithm can be taken as a decentralized and dynamic method to determine the pricing equilibrium. A primal-dual algorithm usually maintains a feasible dual solution and tries to compute a primal solution that is both feasible and satisfies the complementary slackness conditions. If such a solution is found, the algorithm terminates. Otherwise the dual solution is updated towards optimality and the algorithm continues with the next iteration. The fundamental work [3] demonstrates a strong interrelationship between the iterative auctions and the primal-dual linear programming algorithms. A primal-dual linear programming algorithm can be interpreted as an auction where the dual variables represent item prices. The algorithm maintains a feasible allocation and a price set, and it terminates as the efficient allocation and competitive equilibrium prices are found.

For the winner determination problem we will formulate the LP relaxation and its dual. Consider the LP relaxation of the winner determination problem (1):

$$\sum_{i=1}^{n} \sum_{S \subseteq M} v_i(S)\, x_i(S) \to \max$$

subject to $\sum_{S \subseteq M} x_i(S) \le 1, \forall\, i = 1,2,\dots,n,$

$$\sum_{i=1}^{n} \sum_{S \subseteq M} x_i(S) \le 1, \forall\, j \in M, \tag{2}$$

$$x_i(S) \in \{0,1\}, \forall\, S \subseteq M,\ \forall\, i = 1,2,\dots,n.$$

The corresponding dual to problem (2)

$$\sum_{i=1}^{n} p(i) + \sum_{j \in S} p(j) \to \min$$

subject to

$$\begin{aligned} &p(i) + \sum_{j \in S} p(j) \ge v_i(S), \forall\, i, S, \\ &p(i), p(j) \ge 0, \forall\, i, j. \end{aligned} \tag{3}$$

The dual variables $p(j)$ can be interpreted as anonymous linear prices of items, the term

$$\sum_{j \in S} p(j)$$

is then the price of the bundle S and

$$p(i) = \max_{S} \left[v_i(S) - \sum_{j \in S} p(j) \right]$$

is the maximal utility for the bidder i at the prices $p(j)$.

3. CRAB Tool

During the research on combinatorial auctions a need for input problem generator arose. There is a publicly available tool CATS [4], developed at Stanford University, however it has several drawbacks that make it, at least, hardly usable for our purpose. To fulfill our needs we have developed our own tool: CRAB. This tool has several advantages over the CATS, namely:

- fast problem generation,
- combinations are generated in a more predictable way,
- combinations are generated only in given subset of all items,
- CSV as the primary data format,
- fine-grained control over generated problem,
- built-in linear problem solver,
- multiple output formats.

This tool is implemented in Ruby. Although obvious programming language of choice for such kind of tools is C or C++ for performance reasons, we choose Ruby mainly for its dynamic, agile nature which enables us to quickly experiment with different approaches.

3.1 Overview

The combinatorial auction problem is given by the number of buyers and the number of all feasible combinations of goods—bundles. For each buyer also prices of bundles, bids and budget are needed. Number of goods is read in the vector form where the number of vector components (comma separated) is equal to the number of packages. Each vector component corresponds to the number of goods in the package.

In the first phase there are generated all combinations of goods in each package (except empty set). This step is done for every package. By this way all bundles are generated. The list of these bundles is saved in the file (*.csv)—one bundle on row—and there are prepared one column for each buyer. The first row contains column label and the second row is given for buyer's budget.

The user can load the file in CSV into a text editor or in a spreadsheet and fill in the bids (*i.e.* the price offered by the buyer for particular bundle) and budgets for each buyer. If the user uses CRAB only for tests he/she can use automatically generated prices and budget. In both cases the final file has to be saved also in CSV format.

In the second phase the file is transformed into the binary programming problem. The bundles correspond to variables and bids correspond to prices of objective function that is maximized. The problem consists of automatic constraints for each good (each good can be sold only once) and each buyer (buyer cannot get over his budget). The user is free to change automatically generated constraints and remove or add additional (for example not-typical) constraints. All data have to be saved in CVS format again.

Finally, the problem might be passes to the built-in binary programming solver to find out the optimal solution for given combinatorial auction. If so, the problem is transformed into form with minimizing objecttive function with non-negative prices and all constraints in form "less or equal". More detailed description of normalization process is in Sub-section 3.6. Afterwards the transformed model is passed to the Balas' algorithm [5].

3.2 Practical Usage

The CRAB tool as well as the source code could be downloaded from the above mentioned web page and it can operate in two modes: 1) interactive mode and 2) command line mode.

Interactive mode

CRAB could be run in interactive mode which means that the user is prompted for the data interactively. To start CRAB in interactive mode, type[1]

```
ruby crab.rb
```

in terminal emulator (or *command line* window on Windows) from within the directory where the program is installed. Please note that not all features are available when running in interactive mode, if you need them, you have to run it in *command line mode* described below.

Command line mode

As opposite to interactive mode, the CRAB could operate in *command line mode*, which means that all input data as well as all options have to be supplied on the command line. This mode allows one to automate tasks using scripts (and run CRAB non-interactively in night batch jobs, for instance). To get list of all available options, type

```
ruby crab.rb --help
```

All examples in following sections will be given in the command line mode.

3.3 Generating Bundles

As we said before, the CRAB can generate combinatorial problem as specified in Section 2. The principle command is:

```
ruby crab.rb --output <outfile> generate --buyers <nb> --bundles <bundlespec>
```

where <nb> is number of buyers (non-negative integer value) and <bundlespec> is vector specifying number of items in each bundle. Number of bundles is determined by dimension of the vector. Vector should be entered as comma-separated sequence of positive integer values with no spaces in between. CRAB can also generate random prices for each package as well as budget for each buyer.

Following command will generate combinatorial auction with four buyers and two packages first with 5 goods, second one with 3 goods and generate random prices and budgets. Output will be saved to file bids.csv:

```
ruby crab.rb --output bids.csv generate --buyers 4 --goods 5,3 --randomize
```

Every single good in generated combinatorial auction get assigned a unique integer id, starting at one. In previous example, the first good of first package gets id 1, second good in first package gets 2. The first good of second package gets 6, second one gets 7 and so on.

3.4 Output Format

The generated files are ordinary CSV[2] files and thus editable by almost every spreadsheet application. First row contains only column labels and contains no data. Second row contains budget of each buyer. Rest of rows specifies bids of bundles given by each buyer.

First two columns contain only labels and no data. First column denotes package, second one denotes particular good combination within the bundle. Bundle is denoted by id of first and last good in the bundle, the combination is

[1]Assuming the directory with ruby program is listed in the PATH environment variable. If not, fully path to the Ruby interpreter must be given, such as C:\Ruby\bin\ruby.

[2]Comma Separated Values, RFC 4180

denoted by minus-separated list of good ids. Following columns contains the bids given by buyers.

For output from the previous command see **Figure 1**.

User is free to modify the generated file to meet his/her needs; however the format of the file as described above must be preserved.

3.5 Transforming Combinatorial Auction to Binary Programming Problem

Once the combinatorial auction is generated, it can be transformed to the form of binary programming problem and passed to binary programming solver afterwards. The principle command for combinatorial auction transformation is:

```
./ruby crab.rb --output <output file> transform --bids <input file>
```

where <input file> is CSV file specifying the combinatorial auction. The form of input file must be the same as output of generate command. Optionally, you may use the --format option to specify output format. CRAB currently supports two output formats: 1) csv (which is the default) and 2) xa. The first one is the one used by built-in solver; the latter one could be directly passed to the XA integer solver [6].

Following command will transform file bids.csv into binary programming problem, saving the output to file named problem.cvs using the csv format.

```
./ruby crab.rb --output problem.cvs transform --bids bids.csv
```

Following command will create binary programming problem specification file as used by the XA solver:

```
./ruby crab.rb --output problem.lp transform --format xa --bids bids.csv
```

```
   Group,Bid,Offer  of  b1,Offer  of
b2,Offer of b3,Offer of b4
     "",Budget,8570,5879,6500,9065
     1-5,1,9,5,6,2
     1-5,2,7,4,8,3
     1-5,3,9,6,7,3
     1-5,4,6,4,4,7
     1-5,5,9,6,4,8
     1-5,1-2,8,7,11,13
     1-5,1-3,12,9,9,7
     1-5,1-4,9,14,10,12
     ...
     ...
     ...
     6-8,8,8,2,3,5
     6-8,6-7,10,12,10,8
     6-8,6-8,13,7,12,10
     6-8,7-8,11,10,8,9
     6-8,6-7-8,17,19,19,15
```

Figure 1. Example of output

Output format

As we mention above, the transform command can generate output files in two formats. In-depth description of XA format is beyond the scope of this paper. The CRAB format is ordinary CSV file. The first row and first column contains only column resp. row labels and no meaningful data. Remaining but last rows specifies constraints, last column is constraint's right hand side, pre-last column denotes relation. The last row contains objective function.

3.6 Solving

CRAB contains built-in binary programming solver based on Balas' method. The principle command for solving combinatorial auction is:

```
ruby carb.rb solve --problem <input file>
```

where <input file> is CSV file specifying the binary programming problem. The form of input file must be the same as output of transform command using the csv format.

The CRAB tool also provides few options to control the Balas' algorithm. The first option controls overall strategy to walk through the state space. Two strategies are available: depth-first (specified by --depth-first option) and breadth-first (which is the default, specified by --breadth-first option).

Second option deals with branching logic. If --one-first is specified then the one-filled branch is tried first, if --zero-first is specified, zero-filled branch is taken first. *One-first* strategy is the default one. Based on few experiments breadth-first strategy combined with one-first strategy gives the best results (by means of number of iterations required to solve particular problem).

Problem Normalization

As was written above, a binary programming problem with zero-one variables can be solved by Balas' method. Before using Balas' algorithm the problem has to have the correct form. In the CRAB system the problem has to satisfy following conditions:

- all constraints have to be in the form "less or equal",
- objective function has to be minimized and
- prices in the objective function have to be non-negative.

In the case that constraint is in the equation form CRAB automatically transforms it into two constraints—one in the form "less or equal" and the other "greater or equal". In the case that constraint is in the form "grater or equal" CRAB multiplies it by minus one. After such transformation all constraints are in the asked form.

If objective function is maximized it is multiplied by minus one and extreme is changed from max to min.

The last problem can be negative prices. As the problem is binomial (all variables are only one or zero) it is possible to use binomial substitution where the new variable equals 1—old variable. It is clear that the new variable is also binomial. After such substitution in whole model the price for new variable is positive. So it is necessary to use such substitution for all variables with negative prices in objective function.

Obviously the problem normalization is implemented in CRAB. After solving the results is automatically transformed by backward-substitution and user obtains the solution of original problem.

Output format

For output of the built-in solver see **Figure 2.**

As the solver is solving the problem, it prints some statistical information: total number partial solution in the queue, delta from last output and values of few other internal variables. After the solver finishes the computation, it prints out number of iterations done, the solution and the value of objective function.

```
   Iteration  1000:  47  solutions  in
queue (delta 46)
    z_f = 25930.0,  z = 32702,  z_max =
26433.0]
   Iteration  2000:  123  solutions  in
queue (delta 76)
    z_f = 25930.0,  z = 32702,  z_max =
26807.0]
   Iteration  3000:  269  solutions  in
queue (delta 146)
    z_f = 25991.0,  z = 32702,  z_max =
26812.0]
   Iteration  4000:  321  solutions  in
queue (delta 52)
    z_f = 27285.0,  z = 32702,  z_max =
28251.0]
   Iterations done: 4214
   Solution: Vector[1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0], Z = 5203
```

Figure 2. Output of the solver

4. Conclusions

Combinatorial auctions are those auctions in which buyers can place bids on combinations of items, called bundles. This is particular important when items are complements. Combinatorial auctions have many real applications. The proposed CRAB system is a non-commercial software system for generating, solving, and testing of combinatorial auction problems. The tool has several advantages; as fast problem generations, combinations are generated only in given subset of all items, CSV as the primary data format, built-in problem solver, to name some of them.

5. Acknowledgements

The research project was supported by Grants No. 402/07/0166 and No. P402/10/0197 from the Grant Agency of Czech Republic.

REFERENCES

[1] M. Bellosta, I. Brigui, S. Kornman and D. Vanderpooten, “A Multi-Criteria Model for Electronic Auctions,” *ACM Symposium on Applied Computing*, 2004, pp. 759-765.

[2] P. Cramton, Y. Shoham and R. Steinberg, (Eds.) “Combinatorial Auctions,” MIT Press, Cambridge, 2006.

[3] S. Bikhchandani and J. M. Ostroy, “The Package Assignment Model,” *Journal of Economic Theory*, Vol. 107, No. 2, 2002, pp. 377-406.

[4] K. Leyton-Brown, M. Pearson and Y. Shoham, “Towards a Universal Test Suite for Combinatorial Auction Algorithms,” *Proceedings of ACM Conference on Electronic Commerce*, Minneapolis, 2000, pp. 448-457.

[5] E. Balas, “An Additive Algorithm for Solving Linear Programs with Zero-one Variables,” *Operations Research*, Vol. 13, No. 4, 1965, pp. 517-546.

[6] “XA Linear Optimizer System,” 2003. http://www.Sunsetsoft.com/