Vickrey-Clarke-Groves (VCG) Auctions

M. B. Caminati* M. Kerber* C. Lange† C. Rowat‡

May 8, 2015

Abstract

A VCG auction (named after their inventors Vickrey, Clarke, and Groves) is a generalization of the single-good, second price Vickrey auction to the case of a combinatorial auction (multiple goods, from which any participant can bid on each possible combination). We formalize in this entry VCG auctions, including tie-breaking and prove that the functions for the allocation and the price determination are well-defined. Furthermore we show that the allocation function allocates goods only to participants, only goods in the auction are allocated, and no good is allocated twice. Furthermore we show that the price function is non-negative. These properties also hold for the automatically extracted Scala code.

Contents

1 Introduction **1**
- 1.1 Rationale for developing set theory as replacing one bidder in a second price auction 2
- 1.2 Bridging 3
- 1.3 Main theorems 3
- 1.4 Scala code extraction 3

1 Introduction

An auction mechanism is mathematically represented through a pair of functions (a, p): the first describes how some given goods at stake are allocated among the bidders (also called participants or agents), while the second specifies how much each bidder pays following this allocation. Each possible output of this pair of functions is referred to as an outcome of the

*School of Computer Science, University of Birmingham, UK
†Fraunhofer IAIS and University of Bonn, Germany, and School of Computer Science, University of Birmingham, UK
‡Department of Economics, University of Birmingham, UK

auction. Both functions take the same argument, which is another function, commonly called a bid vector b; it describes how much each bidder values the possible outcomes of the auction. This valuation is usually expressed through money. In this setting, some common questions are the study of the quantitative and qualitative properties of a given auction mechanism (e.g., whether it maximizes some relevant quantity, such as revenue, or whether it is efficient, that is, whether it allocates the item to the bidder who values it most), and the study of the algorithms running it (in particular, their correctness).

A VCG auction (named after their inventors Vickrey, Clarke, and Groves) is a generalization of the single-good, second price Vickrey auction to the case of a combinatorial auction (multiple goods, from which any participant can bid on each possible combination). We formalize in this entry VCG auctions, including tie-breaking and prove that the functions a and p are well-defined. Furthermore we show that the allocation function a allocates goods only to participants, only goods in the auction are allocated, and no good is allocated twice. Furthermore we show that the price function p is non-negative. These properties also hold for the automatically extracted Scala code. For further details on the formalization, see [4]. For background information on VCG auctions, see [5].

The following files are part of the Auction Theory Toolbox (ATT) [1] developed in the ForMaRE project [2]. The theories `CombinatorialAuction.thy`, `StrictCombinatorialAuction.thy` and `UniformTieBreaking.thy` contain the relevant definitions and theorems; `CombinatorialAuctionExamples.thy` and `CombinatorialAuctionCodeExtraction.thy` present simple helper definitions to run the definitions on given examples and to export them to the Scala language, respectively; `FirstPrice.thy` shows how easy it is to adapt the definitions to the first price combinatorial auction. The remaining theories contain more general mathematical definitions and theorems.

1.1 Rationale for developing set theory as replacing one bidder in a second price auction

Throughout the whole ATT, there is a duality in the way mathematical notions are modeled: either through objects typical of lambda calculus and HOL (lambda-abstracted functions and lists, for example) or through objects typical of set theory (for example, relations, intersection, union, set difference, cartesian product).

This is possible because inside HOL, it is possible to model a simply-typed set theory which, although quite restrained if compared to, e.g., ZFC, is powerful enough for many standard mathematical purposes.

ATT freely adopts one approach, the other, or a mixture thereof, depending on technical and expressive convenience. A technical discussion of this topic can be found in [3].

1.2 Bridging

One of the differences between the approaches of functional definitions on the one hand and classical (often set-theoretical) definitions on the other hand is that, commonly (although not always), the first approach is better suited to produce Isabelle/HOL definitions which are computable (typically, inductive definitions); while the definitions from the second approach are often more general (e.g., encompassing infinite sets), closer to pen-and-paper mathematics, but also not computable. This means that many theorems are proved with respect to definitions of the second type, while in the end we want them to apply to definitions of the first type, because we want our theorems to hold for the code we will be actually running. Hence, bridging theorems are needed, showing that, for the limited portions of objects for which we state both kinds of definitions, they are the same.

1.3 Main theorems

The main theorems about VCG auctions are

the definiteness theorem: our definitions grant that there is exactly one solution; this is ensured by vcgaDefiniteness.

PairwiseDisjointAllocations: no good is allocated to more than one participant.

onlyGoodsAreAllocated: only the actually available goods are allocated.

the adequacy theorem: the solution provided by our algorithm is indeed the one prescribed by standard pen-and-paper definition.

NonnegPrices: no participant ends up paying a negative price (e.g., no participant receives money at the end of the auction).

Bridging theorems: as discussed above, such theorems permit to apply the theorems in this list to the executable code Isabelle generates.

1.4 Scala code extraction

Isabelle permits to generate, from our definition of VCG, Scala code to run any VCG auction. Use `CombinatorialAuctionCodeExtraction.thy` for this. This code is in the form of Scala functions which can be evaluated on any input (e.g., a bidvector) to return the resulting allocation and prices.

One easy way to deploy such functions is to use the provided Scala wrapper (taking care of the output and including sample inputs). In order to do so, you can evaluate inside Isabelle/JEdit the file CombinatorialAuctionCodeExtraction.thy (position the cursor on its last line and wait for Isabelle/JEdit to end all its processing). This will result in the file

`/dev/shm/VCG-withoutWrapper.scala`, which can be automatically appended the wrapper by running the shell script commented out after the end of. For details of how to run the Scala code see http://www.cs.bham.ac.uk/research/projects/formare/vcg.php.

References

[1] Formare github webpage. https://github.com/formare/auctions/tree/master/isabelle/Auction/Vcg, 2015. Accessed: 2015-05-08.

[2] Formare project webpage. http://www.cs.bham.ac.uk/research/projects/formare/, 2015. Accessed: 2015-05-08.

[3] M. B. Caminati, M. Kerber, C. Lange, and C. Rowat. Set theory or higher order logic to represent auction concepts in isabelle? In *Intelligent Computer Mathematics*, pages 236–251. Springer, 2014. http://arxiv.org/abs/1406.0774.

[4] M. B. Caminati, M. Kerber, C. Lange, and C. Rowat. Sound auction specification and implementation. 16th ACM Conference on Economics and Computation, 2015. http://dx.doi.org/10.1145/2764468.2764511, http://www.cs.bham.ac.uk/~mmk/publications/ec2015.pdf.

[5] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial auctions*. MIT Press, 2006.