

CS364B: Frontiers in Mechanism Design

Lecture #6: Gross Substitutes: Welfare Maximization in Polynomial Time*

Tim Roughgarden[†]

January 22, 2014

1 Introduction

Last lecture we introduced the gross substitutes condition.

Definition 1.1 (Gross Substitutes) A valuation v_i defined on item set U satisfies the *gross substitutes (GS)* condition if and only if the following condition holds. For every price vector \mathbf{p} , every set $S \in D_i(\mathbf{p})$, and every price vector $\mathbf{q} \geq \mathbf{p}$, there is a set $T \subseteq U$ with

$$(S \setminus A) \cup T \in D_i(\mathbf{q}),$$

where $A = \{j : q(j) > p(j)\}$ is the set of items whose prices have increased (in \mathbf{q} relative to \mathbf{p}).

We motivated this condition as the natural one under which the Kelso-Crawford auction converges to a Walrasian equilibrium, where we think of the set S as the last bundle of goods that a bidder i bid on (at prices \mathbf{p}) and $S \setminus A$ as the items bidder i still possesses (at the original prices) at some later iteration of the auction with prices \mathbf{q} . The condition asserts that i still wants the items of $S \setminus A$ and does not want to withdraw its standing bids for them. We discussed last lecture how, more generally, the gross substitutes condition is in a sense the most general one under which Walrasian equilibria are guaranteed.

This lecture gives a completely different sense in which gross substitutes valuations represent the frontier of tractability. Today we study the optimization problem of computing a welfare-maximizing allocation in polynomial time. This is essentially equivalent to studying whether or not the VCG mechanism can be implemented in polynomial time. The reason is that the VCG mechanism reduces to $n + 1$ instances of welfare maximization, where n is the number of bidders — once to compute the allocation, and once more per bidder to

*©2014, Tim Roughgarden.

[†]Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

3 Extension to General Valuations

We now return to the general valuation model described last lecture, where each bidder has an arbitrary valuation v_i that is monotone and that satisfies $v_i(\emptyset) = 0$. All of the arguments and conclusion of the previous section, for unit-demand bidders, carry over with little trouble. The main difference is in the initial integer programming formulation. Since bidders now might want any subset of items, we require one decision variable per bidder i and bundle

$S \subseteq U$:

$$x_{iS} = \begin{cases} 1 & \text{if } i \text{ gets the bundle } S \\ 0 & \text{otherwise.} \end{cases}$$

The welfare objective is now

$$\max \sum_{i=1}^n \sum_{S \subseteq U} v_i(S) x_{iS}.$$

To make sure that every item goes to at most one bidder, we modify the first set of constraints to

$$\sum_{i=1}^n \sum_{S \subseteq U: j \in S} x_{iS} \leq 1 \quad \text{for every } j.$$

Every bidder should get at most one bundle, so

$$\sum_{S \subseteq U} x_{iS} \leq 1 \quad \text{for every } i,$$

Feasible allocations are in a one-to-one objective function-preserving correspondence with the feasible solutions of this integer program ($IP - GEN$). In the linear relaxation ($P - GEN$) of this integer program, we replace each integrality constraint $x_{iS} \in \{0, 1\}$ by the linear constraint $x_{iS} \geq 0$ (again, the constraint $x_{iS} \leq 1$ would be redundant).

Our primal linear program again has one constraint per bidder and per good, so the decision variables in the dual linear program are exactly the same as before. The full dual is

$$\begin{aligned} \min \quad & \sum_{i=1}^n u_i + \sum_{j=1}^m p_j \\ \text{subject to:} \quad & \\ (D - GEN) \quad & u_i + \sum_{j \in S} p_j \geq v_i(S) \quad \text{for every } i \text{ and } S \subseteq U \\ & u_i, p_j \geq 0 \quad \text{for every } i \text{ and } j. \end{aligned}$$

As before, we can rewrite the constraints suggestively as $u_i \geq v_i(S) - \sum_{j \in S} p_j$ for every i and $S \subseteq U$.

Theorem 2.4 extends, with exactly the same proof, to the present general setting.

Theorem 3.1 ([6]) *Let OPT_{IP} and OPT_{LP} denote the optimal objective function values of the integer program ($IP - GEN$) and its linear relaxation ($P - GEN$), respectively.*

1. *There exists a Walrasian equilibrium if and only if $OPT_{LP} = OPT_{IP}$.*
2. *In this case, \mathbf{p} is a Walrasian price vector if and only if it participates in an optimal solution (\mathbf{p}, \mathbf{u}) to (D).*

In the present general setting, the first statement of Theorem 3.1 is very interesting. We know that there are valuation profiles for which there are no Walrasian equilibria. The first part of Theorem 3.1 says the existence vs. non-existence of WE is governed entirely by whether or not the linear relaxation of ($IP - GEN$) is exact. This is remarkable in that the

latter condition has nothing to do with prices per se. The second part of the theorem says that Walrasian price vectors can always be interpreted as dual variables in a natural way.

Last lecture, we use the Kelso-Crawford auction to prove that when all bidders' valuations satisfy the gross substitutes condition, there exists a Walrasian equilibrium. Combining this result with Theorem 3.1 immediately yields the following non-trivial fact.

Corollary 3.2 *If v_1, \dots, v_n satisfy the gross substitutes condition, then the linear relaxation $(P - GEN)$ of $(IP - GEN)$ is exact.*

Corollary 3.2 can also be proved directly, although it is not easy to do so [7, §8.7].² Recapping the sequence of ideas that got us to this point, we catch a glimpse of the rich and interconnected theory around the gross substitutes condition, which holds simultaneously the key to strong incentive and computational tractability guarantees. We used a non-incentive-compatible ascending auction (the Kelso-Crawford auction) to establish the existence of Walrasian equilibria, which implies (through Theorem 3.1) the seemingly unrelated property that we can maximize welfare via linear programming, which opens the door for a polynomial-time implementation of the VCG mechanism in the next section (which has different payments than the Kelso-Crawford auction and is DSIC).

4 Implementing the VCG Mechanism in Polynomial Time

Implementing the VCG mechanism in polynomial time reduces to computing an allocation (S_1, \dots, S_n) that maximizes the welfare $\sum_{i=1}^n v_i(S_i)$ in polynomial time — $n + 1$ invocations of the latter suffice to compute the VCG mechanism's allocation and payment rules. This section focuses on the latter, purely algorithmic, problem.

4.1 Representations of Valuations

We're going to pursue algorithms that run in time polynomial in the input size. But what is "input size"? There are at least three possible answers to this question.

The first approach is to assume that the valuations are represented explicitly as a list of $v_i(S)$'s, for all bidders i and bundles S . This means that the input size is roughly $n2^m$. In this case, welfare-maximization can be solved in polynomial time by dynamic programming, even for the most general valuation model from the last lecture (see the exercises). Two issues with this result are: 2^m is an unreasonably large number once m is of moderate size; and the complexity of the problem is independent of the bidders' preferences (e.g., whether items are complements or not), which contradicts all real-world experience with the problem.

The second approach, which is interesting but not pursued in this section, is to focus on special classes of valuations that can be represented with a number of bits that is polynomial in n and m . All of the concrete examples of gross substitutes values that we've mentioned,

²In fairness, the direct proof does not assume the strong duality of linear programs, as we have done here.

such as k -unit valuations, have this property. In such cases, we look for algorithms that run in time polynomial in n and m .

The third approach is to model a valuation as a “black box” that is capable of answering “queries.” If this sounds too abstract for you, notice that this is exactly how we’ve been treating bidders in our ascending auctions along: we have no idea how a bidder represents internally their valuation, nor do we think of it as part of the input, but we expect a bidder to be able to quickly answer questions such as demand queries.

With black-box valuations, what kinds of queries are permitted? The answer is somewhat negotiable, but in auction design, the simpler and more natural the better. Two types of queries have been extensively studied.

Demand queries. We provide a black-box valuation v_i with a price vector \mathbf{p} on items, and expect back one or all of its favorite sets $D_i(\mathbf{p})$. All of our ascending auctions thus far have used demand queries to interact with bidders.

Value queries. We provide a black-box valuation v_i with a subset S of items, and expect back its value $v_i(S)$ for that set.

Arguing the reasonableness of a query type can be done in both mathematical and non-mathematical ways. A good mathematical argument is to show that the query can be implemented in polynomial time for many or all concrete classes of valuations that one cares about (cf., the second approach above). Then, counting the query as merely a constant or polynomial amount of work is not cheating by pushing a ridiculous amount of work onto the bidder. A good case can be made for value queries along such lines. For demand queries, the mathematical argument is more mixed — they can be implemented in polynomial time for many but not all concrete valuation classes of interest. The informal “naturalness” argument for demand queries is strong, however — it’s hard to think of any iterative auctions in the real world that don’t use them.

4.2 A Polynomial-Time Algorithm for Welfare Maximization

The goal of this section is to prove that welfare-maximization with gross substitutes valuations is, at least in principle, tractable.

Theorem 4.1 ([4]) *If valuations v_1, \dots, v_n satisfy the gross substitutes condition and are represented as “black boxes,” then a welfare-maximizing allocation can be computed in a polynomial (in n and m) amount of time, value queries, and demand queries.*³

Theorem 4.1 implies that, under the same assumptions, the VCG mechanism can be implemented in polynomial time.

The algorithm in Theorem 4.1 requires heavy linear programming machinery and is not practical.⁴ It is theoretically and conceptually interesting, however. We’ll see next lec-

³In fact, only value queries are required. Time permitting, we’ll explain why in a future bonus lecture.

⁴There are also combinatorial algorithms, but these are also pretty slow and complex (see [5]).

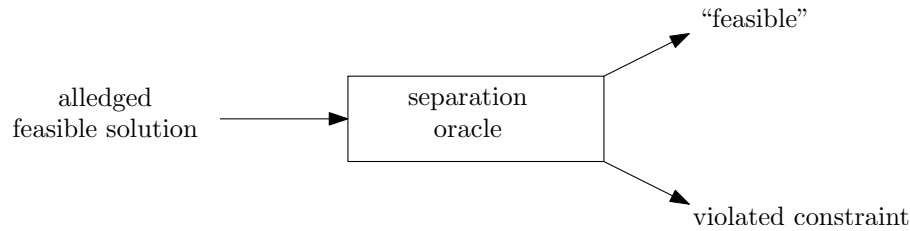


Figure 2: A separation oracle.

ture that, for valuations slightly more general than gross substitutes valuations, welfare-maximization cannot be done in polynomial time (assuming $P \neq NP$).

The algorithm Theorem 4.1 uses the *ellipsoid algorithm* [3] as a subroutine. The ellipsoid algorithm is an old but still magical algorithm that can solve many implicitly described linear programs in polynomial time. Other polynomial-time linear programming algorithms, like interior-point methods, only work for explicitly described linear programs. As such, the ellipsoid algorithm is an indispensable tool for proving tractability (in principle) results.⁵

More precisely, consider a linear program such that:

1. There are N decision variables.
2. There are any number of constraints, for example exponential in N . These constraints are not provided explicitly as input.
3. There is a polynomial-time *separation oracle* for the set of constraints. By “polynomial-time,” we mean running time polynomial in N and the maximum number of bits of precision required. A separation oracle (Figure 2) is a subroutine that takes as input an alleged feasible solution to the linear program, and either (i) correctly declares the solution to be feasible; or (ii) correctly declares the solution to be infeasible, and more strongly provides a proof of infeasibility in the form of a constraint that the proposed solution violates.

Fact 4.2 (Ellipsoid Algorithm [3]) *Every linear program that admits a polynomial-time separation oracle can be solved in polynomial time.*

Again, “polynomial” means polynomial in the number of variables and in the number of bits of precision. Many researchers use the ellipsoid algorithm as a “black box” without understanding how it works, and we will continue that tradition in the lecture. For more on how and why the algorithm works, see e.g. [2, 1].

Proof of Theorem 4.1: The first step is to solve the primal linear program ($P - GEN$) in polynomial time. The number of variables in this linear program is exponential in m , so the ellipsoid algorithm cannot help us. It has only $n + m$ constraints, however, and a result the

⁵While the ellipsoid algorithm is not practical, other algorithms that rely on a separation oracle, such as cutting plane methods, are often practically useful (if not polynomial time in the worst case).

dual program ($D - GEN$) has only $n + m$ decision variables. This makes the dual program a candidate for application of the ellipsoid algorithm, provided we can implement a separation oracle in polynomial time.

Given an alleged solution (\mathbf{p}, \mathbf{u}) to the dual program ($D - GEN$), how can we quickly verify whether or not it is feasible? We can check the $n + m$ nonnegative constraints directly — the hard part is verify the exponentially many constraints of the form

$$u_i \geq v_i(S) - \sum_{j \in S} p_j \tag{4}$$

for a bidder i a bundle S . We do this for each of the n bidders in turn. For a bidder i , we use a demand query (using the dual variables \mathbf{p} as prices) to identify a bundle S_i^* that maximizes the right-hand side of (4) — if any constraint for bidder i is violated, it is this one. Then, using a value query (for S_i^*) we evaluate the right-hand side of (4) and compare it to u_i . If these n tests (one per bidder) are passed, together with the nonnegativity constraints, then we (correctly) declare (\mathbf{p}, \mathbf{u}) to be feasible; otherwise, this procedure identifies a violated constraint that we can return for use in the ellipsoid algorithm. The procedure uses a polynomial (in n , m , and precision required) number of steps, value queries, and demand queries.⁶

Fact 4.2 and the above separation oracle imply that we can solve the dual program ($D - GEN$) in polynomial time. By strong duality, the optimal values of ($P - GEN$) and ($D - GEN$) coincide, so we can compute the optimal value of ($P - GEN$) in polynomial time. Because the valuations satisfy the gross substitutes condition, Corollary 3.2, this is also the optimal value for ($IP - GEN$) — the welfare of an optimal (integral) allocation.

The final step is to use repeatedly our polynomial-time algorithm for computing the *value* of a welfare-maximizing allocation to reconstruct the allocation itself. This can be done using a standard “self-reducibility” trick. The idea is to start with the first item and try assigning it to each bidder $i = 1, 2, \dots, n$ in turn. One of these n tries is the correct “guess,” and correctness can be verified by applying our subroutine (on the reduced problem) for computing the value of a welfare-maximizing allocation. We leave the details as an exercise. ■

References

- [1] A. Ben-Tal and A. Nemirovski. Optimization iii. Lecture notes, 2012.
- [2] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. Second Edition, 1993.
- [3] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

⁶To this point, we have not used the assumption that the valuations satisfy the gross substitutes condition, only that they support value and demand queries.

- [4] N. Nisan and I. Segal. The communication requirements of efficient allocations and supporting prices. *Journal of Economic Theory*, 129(1):192–224, 2006.
- [5] R. Paes Leme. Gross substitutability: an algorithmic survey. Working paper, 2013.
- [6] L. S. Shapley and M. Shubik. The assignment game I: The core. *International Journal of Game Theory*, 1(1):111–130, 1972.
- [7] R. V. Vohra. *Advanced Mathematical Economics*. Routledge, 2005.