Algorithmic Mechanism Design

Paul Saikko

BSc Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, February 7, 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Paul Saikko

Työn nimi — Arbetets titel — Title
Algorithmic Mechanism Design

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
BSc Thesis	February 7, 2014	19

Tiivistelmä — Referat — Abstract

We look at algorithmic problems from a game theoretic perspective. When participants are rational, selfish agents, they cannot be relied upon to act as instructed by an algorithm if it does not benefit them. We describe a model for analyzing these problems and the general Vickrey–Clarke–Growes solution from the field of mechanism design. Some algorithmic issues in implementing the general solution are looked at and we explore potential ways to overcome them.

ACM Computing Classification System (CCS):
Theory of computation → **Algorithmic mechanism design**; *Algorithmic game theory*; Design and analysis of algorithms

Avainsanat — Nyckelord — Keywords
algorithms, mechanism design, game theory

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

With the growing prevalence of the internet as a platform for computation, problems outside the scope of traditional algorithm design have become increasingly relevant. One such problem is that for the most part, computers on the internet are owned and controlled by separate entities. In many situations, it is not reasonable to expect that these remote computers will act as instructed by an algorithm or protocol if it does not benefit the owner. This has motivated the use of methods from microeconomics and game theory, especially the field of mechanism design, in designing algorithms for the internet. At the same time, new requirements for computation time and complexity are placed on existing mechanism design models. This emerging field of research was termed algorithmic mechanism design by Noam Nisan and Amir Ronen in their 1999 article of the same name [10].

In this paper we present a model for analyzing mechanism design problems and a general solution for a common subset of these problems. We look at the computational issues inherent in implementing mechanism design solutions as algorithms. Using the task scheduling problem as an example, we explore ways to overcome these issues with randomization and by extending the model.

2 Mechanism design problems

A mechanism design problem can be though of as trying to create a set of rules for a game with some desired outcome in mind. The players of the game have some valuation for every possible outcome and we assume that they will act rationally to maximize the value they get. We use two classic mechanism design problem solutions to illustrate this further.

2.1 Single item auction

In this problem the game is a simple auction of an item. Every player participating in the auction is asked to privately place a single bid on the item. The goal is to design a mechanism so that every player i is motivated to place a bid equal to his valuation of the item t_i.

If we simply ask the winning bidder to pay an amount equal to his bid, it would be in the player's best interest to make the lowest possible bid he thinks will win the item. This could lead to a situation where the player values the item the most does not make the highest bid.

The solution, known as the Vickrey auction rule, is to have the winning bidder i pay an amount p_i equal to the second highest bid [15]. If a bidder now reports his valuation truthfully and wins the auction, he will make a profit equal to his valuation of the item t_i minus the price he paid p_i. A bidder clearly cannot increase his profit by placing a bid lower than his

valuation for the item, but does risk losing the profit from winning the auction. Conversely, if a bidder i wins an item due to making a bid higher than his valuation t_i, the second highest bid p_i must also have been higher than t_i so i makes a loss of $t_i - p_i$. This ensures that a rational bidder will make a bid equal to his valuation for the item.

2.2 Public project problem

Suppose that a town of n people is planning a public project such as a new road. The project has a cost C, and every resident i would benefit from the finished project by some amount t_i known only to them. The town wants to complete the project only if its cost is less than the total benefit to taxpayers $\sum_{i=1}^{n} t_i$, so residents are asked to report their valuations t_i before starting the project. The goal is to impose a tax on the residents based on their reported valuations of the project so that every resident i truthfully reports its value for the project t_i to the city.

If we spread the cost evenly among n residents so that each of them pays a tax of C/n, residents for whom $t_i > C/n$ are motivated report their valuation as greater than C to ensure that the project is started. If we have every resident i pay a tax equal to his reported valuation instead, i will be inclined to report a valuation less than t_i to save money. When many residents do this, the project is not started even if $C < \sum_{i=1}^{n} t_i$.

The solution, known as the Clarke tax, is to make the tax for every resident i be the smallest amount he would have to pay so that the project is completed [1]. This amount is equal to the cost of the project C minus the sum of the reported valuations of the other residents $\sum_{j \neq i} t_j$ or 0 when that difference is negative. To show that this solution gives the desired result, we analyze a generalization of it (the VCG mechanism) in section three.

Even this simple result has potential uses in computing environments. Consider a situation where multiple processors share a single cache. There is a cost for loading some data into the cache, but every processor gets some benefit from it. If we want to load the data only when the total benefit is greater than the cost, this situation is clearly analogous to the public project problem [9] and a similar solution applies.

3 The Mechanism Design Model

In order to analyze these problems and their solutions we need to formally define them. To that end, we present a formal model for mechanism design problems and for the mechanism itself [10].

3.1 The model

Definition 1 (Mechanism design problem)**.** The mechanism design problem specifies what the players (*agents*) want and the possible outcomes given the wants of the player (the *output specification*). There is a set of possible outcomes O from which some outcome o is chosen. Specifically,

1. There are n agents and every agent i
   (a) has a *type* $t_i \in T_i$ which is private information,
   (b) has a *valuation function* $v_i : T_i \times O \to \mathbb{R}$ where $v_i(t_i, o)$ is a measure of how much i values an outcome o given its type t_i,
   (c) can receive a *payment* $p_i \in \mathbb{R}$ from the mechanism (see definition 3), and
   (d) wants to maximize its *utility* $u_i = p_i + v_i(t_i, o)$.
2. An output specification $S : \prod_{i=1}^{n} T_i \to \mathcal{P}(O)$ maps every tuple of types $\bar{t} = (t_1, \ldots, t_n)$ to the set of allowed outcomes, some subset of O.

Definition 2 (Mechanism design optimization problem)**.** We will look specifically at mechanism design optimization problems, a subset of mechanism design problems where the output specification S is given by an objective function $g(o, \bar{t})$ that we want to minimize. In the exact case,

$$S(\bar{t}) = \arg\min_{o \in O}(g(o, \bar{t})).$$

For the approximate case we require a c-approximation instead of an optimal solution, where $S(\bar{t})$ is the set of all the outcomes o that satisfy $g(o, \bar{t}) \leq c \cdot g(o', \bar{t})$ for every other possible outcome o' and some fixed $c \in \mathbb{R}_+$.

Definition 3 (The mechanism)**.** The mechanism $m = (o, \bar{p})$ is made up of two parts: the output function and an n-tuple of payments. That is to say, the mechanism

1. defines for every agent i
   (a) a set of strategies A_i from which it can choose to perform any a_i,
   (b) a payment $p_i(a_1, \ldots, a_n)$ which is determined by a function $p_i : \prod_{i=1}^{n} A_i \to \mathbb{R}$, and
2. provides an output function $o : \prod_{i=1}^{n} A_i \to O$.

To clarify these notions we define the single item auction problem and Vickrey auction rule in the context of this model. The outcome o tells which of the n bidders won the auction and $O = \{1, \ldots, n\}$ is the set of all possible winners. Each bidder is an agent and the type t_i of agent i is how much it

values the item being auctioned. The output specification $S(\bar{t}) = \arg\max_i(t_i)$ is the requirement that the item being auctioned should go to the bidder that values it the most. An agent's valuation of an outcome $v_i(t_i, o)$ is equal to t_i if i won the item and 0 otherwise. The mechanism asks the agents to report their valuations by bidding privately on the item. An agent's strategy a_i is the bid it submits given its type t_i, it can be any positive amount so $A_i = \mathbb{R}_+$. The mechanism's output function $o(a_1, \ldots, a_n) = \arg\max_i(a_i)$ simply picks the agent with the highest bid to be the winner. The payment $p_i(a_1, \ldots, a_n)$ is the negation of the second highest bid for the winning agent and 0 for everyone else. This single item auction problem is a mechanism design optimization problem and the objective function $g(o, \bar{t})$ that we want to minimize is $-\sum_{i=1}^{n} v_i(t_i, o)$.

3.2 Properties of mechanisms

Many useful properties of mechanisms and mechanism design problems can be defined using the formal model. We will use the following notation for a tuple $\bar{a}$ with the i^{th} element removed: $\bar{a}_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$. When convenient to do so, we will also say $\bar{a} = (\bar{a}_{-i}, a_i)$.

Definition 4. We say that an agent i's strategy is *dominant* if it maximizes the agent's utility regardless of the strategies chosen by the others. More formally, when

$$o = o(\bar{a}_{-i}, a_i) \qquad p_i = p_i(\bar{a}_{-i}, a_i)$$
$$o' = o(\bar{a}_{-i}, a'_i) \qquad p'_i = p_i(\bar{a}_{-i}, a'_i)$$

a_i satisfies the following condition:

$$v_i(t_i, o) + p_i \geq v_i(t_i, o') + p'_i$$

for every $a'_i \in A_i$ and $\bar{a}_{-i} \in \prod_{j \neq i} A_j = A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n$.

Definition 5 ([10])**.** A mechanism is an *implementation with dominant strategies* or in short an *implementation* if for every agent i and type t_i there exists a dominant strategy a_i and for every tuple $\bar{a}$ of dominant strategies $o(\bar{a}) \in S(\bar{t})$. For mechanism design optimization problems this means that every tuple of dominant strategies $\bar{a}$ minimizes or maximizes $g(o(\bar{a}), \bar{t})$.

We will focus on *direct revelation mechanisms*, mechanisms where an agent's strategy is to report a type to the mechanism, so for every agent i the set of possible strategies A_i is the same as the set of possible types T_i.

Definition 6 ([10])**.** A direct revelation mechanism is *truthful* (alternatively, *incentive compatible*) if for every agent i, truthfully reporting its type is a dominant strategy. This is to say that the agent's strategy a_i is the same as its type t_i. A mechanism is *strongly truthful* if for every agent i, truthfully reporting its type is the only dominant strategy.

The following result allows us to restrict ourselves to truthful mechanisms without loss of generality.

Theorem 7 (Revelation Principle [12])**.** *If a problem can be implemented by a mechanism, there exists a truthful implementation as well.*

Proof. Suppose that $a_1, \ldots, a_n$ is a tuple of dominant strategies for a direct revelation mechanism $m(o, \bar{p})$. Let $a_i(t_i)$ be the strategy chosen by i when its type is t_i. We define a new mechanism $m'(o', \bar{p}')$ which mimics the agents' strategies, where

$$o'(\bar{t}) = o(a_1(t_1), \ldots, a_n(t_n)) \text{ and}$$
$$p'_i(\bar{t}) = p_i(a_1(t_1), \ldots, a_n(t_n))$$

Now each agent will get the same utility from reporting t_i as it did from reporting a_i in the original mechanism, making truth-telling equivalent to the original dominant strategy. □

3.3 Properties of truthful mechanisms

All truthful implementations share certain basic properties. We define the two requirements for such implementations. Intuitively, the first requirement demands that if an agent falsely reports its type in a way that does not change the outcome, the agent's utility is not increased. The second requirement then simply states that the outcome chosen by the mechanism optimizes the utility of each agent. This means that if an agent falsely reports its type in a way that does change the overall outcome, the resulting outcome does not increase the agent's utility.

Theorem 8 ([12])**.** *An implementation is truthful if and only if for all agents i and for every $\bar{t}_{-i}$,*

1. *for all $o \in O$ there exists a value $p \in \mathbb{R}$ such that for every $t_i \in T_i$ for which $o(\bar{t}_{-i}, t_i) = o$, the payment $p_i(\bar{t}_{-i}, t_i)$ equals p, and*
2. *$o(\bar{t}_{-i}, t_i) \in \arg\max_o(v_i(t_i, o) + p_i)$ for every $t_i \in T_i$ when $o \in \{o(\bar{t}_{-1}, t'_i) : t'_i \in T_i\}$. That is, the mechanism optimizes the outcome for i.*

Proof.

("If") Let $o = o(\bar{t}_{-i}, t_i)$, $o' = o(\bar{t}_{-i}, t'_i)$, $p = p_i(\bar{t}_{-i}, t_i)$ and $p' = p_i(\bar{t}_{-i}, t'_i)$. If $o = o'$, agent i cannot increase its utility by reporting t'_i instead of t_i because $p = p'$. If $o \neq o'$ the same applies because the mechanism optimizes the result for i. In both cases this means $v_i(t_i, o) + p \geq v_i(t_i, o') + p'$ for all t'_i so the mechanism is truthful.

("Only if")

1. Suppose there exists a type declaration $t_i' \in T_i$ such that $p_i(t_i) < p_i(t_i')$ and $o(\bar{t}_{-i}, t_i') = o(\bar{t}_{-i}, t_i)$. Because the outcome (and therefore the agent's valuation) does not change, we see that agent i can increase its utility by reporting t_i' instead of its actual type t_i. This directly contradicts the definition of a truthful mechanism. The same reasoning applies to the case where $p_i(t_i) > p_i(t_i')$ as any agent with a type of t_i' would now benefit from reporting t_i.

2. Suppose $o(\bar{t}_{-i}, t_i) \notin \arg\max_o \big(v_i(t_i, o) + p_i\big)$ for some $t_i \in T_i$ when $o \in \{o(\bar{t}_{-1}, t_i') : t_i' \in T_i\}$. Now there exists a $o' \in \arg\max_o \big(v_i(t_i, o) + p_i\big)$ such that $o' = o(\bar{t}_{-i}, t_i')$ and an agent with type t_i can increase its utility by reporting t_i' instead of t_i.

□

4 The Vickrey–Clarke–Growes mechanism

In this section we present and analyze a generalized truthful solution for mechanism design optimization problems where the goal is to find the outcome that maximizes the total value given to the agents. We assume that the set of possible outcomes is finite.

Definition 9. A mechanism design optimization problem is *utilitarian* if the objective function we want to maximize $g(o, \bar{t})$ is the sum of the agents' valuations,

$$g(o, \bar{t}) = \sum_{i=1}^{n} v_i(t_i, o).$$

4.1 The mechanism

The Vickrey–Clarke–Groves (VCG) mechanism gives a truthful solution to many utilitarian problems. It is a generalization of the Vickrey auction rule by Edward Clarke and Theodore Groves [1, 5].

Definition 10 (Vickrey–Clarke–Groves Mechanism [10])**.** A VCG mechanism is a direct revelation mechanism defined by two conditions:

1. The mechanism maximizes the sum of agent valuations,

$$o(\bar{t}) \in \arg\max_{o \in O} \left(\sum_{i=1}^{n} v_i(t_i, o) \right).$$

2. The payment $p_i(\bar{t})$ given to agent i is

$$p_i(\bar{t}) = \sum_{j \neq i} v_j(t_j, o) + h_i(\bar{t}_{-i}),$$

where h_i is an arbitrary function of $\bar{t}_{-i}$.

A notable feature of the VCG mechanism is that the payment p_i to agent i does not directly depend on its reported type t_i. In addition to this, $h_i(\bar{t}_{-i})$ is essentially a constant from agent i's perspective. It has no strategic value to i because its value cannot be influenced the agent's actions in any way. The function $h_i(\bar{t}_{-i})$ can be used to control how much money is paid out by the mechanism in total. Setting $h_i = 0$ for all i produces a truthful mechanism, but it many cases the payments produced by such a mechanism are unreasonable.

Definition 11 (Clarke pivot rule [12])**.** A common way to choose the function $h_i(\bar{t}_{-i})$ when agent valuations are positive is to let h_i be the greatest possible sum of agent valuations when i is ignored,

$$h_i(\bar{t}_{-i}) = -\max_{o \in O} \Big(\sum_{j \neq i} v_j(t_j, o) \Big).$$

When used with the Clarke pivot rule, the VCG mechanism first maximizes the sum of agent valuations and then fines each agent i the difference between the optimal outcome with i and the optimal outcome without i. Intuitively, every agent i is charged a price equal to the amount of harm it caused to other agents, or the benefit received by the other agents if i was not present [3].

Lemma 12. *A VCG mechanism that makes payments using the Clarke pivot rule makes no positive transfers. That is, $p_i \leq 0$ for every agent i.*

Proof. We note first that a VCG mechanism using the Clarke pivot rule makes payments

$$p_i(\bar{t}) = \sum_{j \neq i} v_j(t_j, o) - \max_{b \in O} \Big(\sum_{j \neq i} v_j(t_j, b) \Big). \tag{1}$$

From there we can see that

$$\sum_{j \neq i} v_j(t_j, o) - \max_{b \in O} \Big(\sum_{j \neq i} v_j(t_j, b) \Big) \leq 0 \tag{2}$$

because b is chosen specifically to maximize $\sum_{j \neq i} v_j$ while o was chosen to maximize $\sum_{i=1}^{n} v_i$. □

The Vickrey auction rule fills both criteria for a VCG mechanism. Since there is only one item to auction off, the sum of the agents' valuations is clearly maximized by giving the item to the agent with the highest valuation for it. It is less clear that the payments specified by the Vickrey auction rule are VCG payments. We know that VCG mechanisms make payments of the form $p_i(\bar{t}) = \sum_{j \neq i} v_j(t_j, o) + h_i(\bar{t}_{-i})$. If we let $h_i(\bar{t}_{-i}) = 0$ for all i, the payments made are simply $p_i(\bar{t}) = \sum_{j \neq i} v_j(t_j, o)$ and the result is an auction where the winning bidder is charged nothing, and the losing agents are given payments equal to the winner's bid. This mechanism is still truthful but clearly impractical.

If we use the Clarke pivot rule to determine $h_i(\bar{t}_{-i})$ instead, the payments become $p_i(\bar{t}) = \sum_{j \neq i} v_j(t_j, o) - \max_{b \in O}\left(\sum_{j \neq i} v_j(t_j, b)\right)$. This is equivalent to the Vickrey auction rule. If an agent i wins the auction, $\sum_{j \neq i} v_i\big(t_j, o(\bar{t})\big) = 0$, and $\max_{b \in O}\left(\sum_{j \neq i} v_j(t_j, b)\right) = \max_{j \neq i}(t_j)$ is the best sum of agent valuations that could be achieved if i was ignored, the second highest bid. If agent i does not win the auction, $\sum_{j \neq i} v_i\big(t_j, o(\bar{t})\big) = \max_{b \in O}\left(\sum_{j \neq i} v_j(t_j, b)\right)$, so $p_i = 0$.

Lemma 13. *VCG mechanisms are truthful.*

Proof. This follows from Theorem 15 when $\beta_i = 0$ for every agent i. □

4.2 Weighted VCG

The VCG mechanism can be applied to weighted utilitarian maximization problems, a more general type of utilitarian maximization problem where for weights $\beta_1, \ldots, \beta_n > 0$, the objective function is

$$g(o, \bar{t}) = \sum_{i=1}^{n} \beta_i v_i(t_i, o).$$

Definition 14 (Weighted Vickrey–Clarke–Growes mechanism)**.** The VCG mechanism for weighed utilitarian problems differs from the basic VCG mechanism only by including weights in the payment function. It is a direct revelation mechanism where

$$o(\bar{t}) \in \arg\max_{o \in O}\big(g(o, \bar{t})\big) \text{ and}$$
$$p_i(\bar{t}) = \frac{1}{\beta_i} \sum_{j \neq j} \beta_j v_j\big(t_j, o(\bar{t})\big) + h_i(\bar{t}_{-i}).$$

Theorem 15 ([12])**.** *Weighted VCG mechanisms are truthful.*

Proof. We show that if truth-telling is not a dominant strategy for some mechanism, then it can not be a weighted VCG mechanism. To that end,

suppose there exists a weighted VCG mechanism which is not truthful. From Definition 6 we know that for some agent i, type t_i, false declaration t'_i, and declarations $\bar{t}_{-i}$,

$$v_i(t_i, o(\bar{t}_{-i}, t_i)) + p_i(\bar{t}_{-i}, t_i) < v_i(t_i, o(\bar{t}_{-i}, t'_i)) + p_i(\bar{t}_{-i}, t')$$

Assume for simplicity that $h_i = 0$ and let $o(\bar{t}) = o(\bar{t}_{-i}, t_i)$ and $o' = o(\bar{t}_{-i}, t'_i)$. Once we substitute the VCG mechanism's payment function into the above inequality we see that it implies

$$\begin{aligned}
v_i(t_i, o(\bar{t})) + \frac{1}{\beta_i} \sum_{j \neq i} \beta_j v_j(t_j, o(\bar{t})) < v_i(t_i, o') + \frac{1}{\beta_i} \sum_{j \neq i} \beta_j v_j(t_j, o') \quad &\Leftrightarrow \\
\beta_i v_i(t_i, o(\bar{t})) + \sum_{j \neq i} \beta_j v_j(t_j, o(\bar{t})) < \beta_i v_i(t_i, o') + \sum_{j \neq i} \beta_j v_j(t_j, o') \quad &\Leftrightarrow \\
\sum_{i=1}^{n} \beta_i v_j(t_j, o(\bar{t})) < \sum_{i=1}^{n} \beta_i v_j(t_j, o').
\end{aligned}$$

This directly contradicts the definition of a weighted VCG mechanism which requires that

$$o(\bar{t}) \in \arg\max_o \left(\sum_{i=1}^{n} \beta_i v_i(t_i, o) \right).$$

□

4.3 Computational complexity of VCG

The VCG mechanism guarantees a truthful solution to utilitarian problems, but this property is a direct result of being able to compute an optimal outcome for the problem. This can be seen from the proof of the weighted VCG mechanism's truthfulness, which relies explicitly on the outcome being optimal. If we try to make a VCG-like mechanism with an approximation instead, we can no longer be sure that the mechanism is a truthful implementation of the problem. In fact, using an approximation to determine the outcome is known to result in a mechanism that is not truthful for a wide range of problems, including combinatorial auctions and many cost-minimization problems [11].

The requirement for an optimal output can be a severe limitation on the usefulness of VCG mechanisms. For many interesting problems, this means that computing the agents' payments and the output of the mechanism takes exponential time. In those cases, using a VCG mechanism is clearly impractical if we have a large number of agents. Even in cases where the payments and outcome can be calculated in polynomial time, such as the shortest paths mechanism design problem we discuss in the next section, the total computation time for the payments of a truthful mechanism far exceeds that of the basic problem.

5 Applications of the VCG Mechanism

The VCG mechanism can be applied as-is to solve a number of interesting problems. We take a look at two relatively simple applications of the mechanism to well-known computer science problems.

5.1 Shortest paths

Many graph problems where the graph's edges can be thought of as agents and the edge weights determine the agents' valuations can be solved using the VCG mechanism. We look at the shortest paths problem which has clear real-world applications when the graph in question is a computer network.

Definition 16. The mechanism design version of the shortest paths problem is defined as follows:

1. Each edge e of a directed graph G is an agent.
2. The type t_e of agent e is the cost of sending a message along that edge.
3. The possible outcomes are paths from some node x to another node y, and the goal is to find the cheapest of these paths.
4. An agent's valuation v_e for a path is $-t_e$ if e is on that path and 0 otherwise.

The problem is nontrivial because every agent is free to lie about the cost of sending data along its edge if the lie will increase its profit. In order to avoid cases where some path from x to y cannot be completed without some single agent, we will assume that the graph is 2-connected. We construct a truthful implementation with the VCG mechanism and Clarke pivot rule [10].

The solution first calculates the shortest path o based on the agents' reported costs $\bar{t}$. Every agent is given the VCG payment of $p_e(\bar{t}) = \sum_{j \neq e} v_j(t_j, o) + h_e(\bar{t}_{-e})$. The first part of this payment, $\sum_{j \neq e} v_j(t_j, o)$ is the cost of the shortest path from x to y if agent i's edge had no cost. The second part, $h_e(\bar{t}_{-e})$, is $-\max_o(\sum_{j \neq e} v_j(t_j, o))$ when the Clarke pivot rule is used. This amount is the cost of the shortest path from x to y that does not include edge e. Because this is a VCG mechanism, we know that an incorrectly reported cost can never increase an agent's utility so truth-telling is a dominant strategy for every agent. One drawback of this solution is that the payments are costly to calculate. If we simply use Dijkstra's algorithm to find the payment for every agent, the mechanism would require $O(nm \log n)$ computation time overall, but this may not be the best possible approach.

5.2 Task scheduling

A more revealing mechanism design problem is task scheduling. The agents' types are more complex than in a single item auction or a shortest paths problem.

Definition 17. The mechanism design version of the task scheduling problem consists of the following parts:

1. There are n agents and k tasks to be split between them.
2. The type of an agent i is a vector of execution times $t_i = (t_{i_1}, \ldots, t_{i_k})$ where t_{i_j} is the amount of time the agent needs to execute task j.
3. The possible outcomes $x = x(\bar{t})$ are tuples of task allocations $(x_1(\bar{t}), \ldots, x_n(\bar{t}))$, where $x_i = x_i(\bar{t})$ is the subset of all tasks allocated to agent i.
4. An agent's valuation v_i is $-\sum_{j \in x_i} t_{i_j}$, the negation of the total time it will take the agent to complete all the tasks allocated to it. We assume that agents cannot execute tasks in parallel.
5. The objective function $g(x, \bar{t})$ we want to minimize is the *make-span* $\max_i(\sum_{j \in x_i} t_{i_j})$, or the amount of time it takes to finish the final task.

We will see in Section 6 that within the limits of the basic model we defined, no mechanism can give an optimal solution to the problem. On the other hand, the VCG mechanism does not guarantee a truthful mechanism if we use an approximation algorithm for the output function instead. We get around this limitation by optimally solving an approximation of the problem instead [10].

A simple approximation of the task scheduling problem is to minimize the total amount of work done by all the agents $\sum_{i=1}^{n}(\sum_{j \in x_i} t_{i_j})$. We can use this approximation of the problem as a basis for an approximation mechanism. Consider a mechanism that

1. allocates each task j to $\arg\min_i(t_{i_j})$, the agent that can perform it in the least amount of time and
2. for every task allocated to i, gives i a payment equal to the second best time for that task so the total payment p_i is $\sum_{j \in x_i}(\min_{i' \neq i}(t_{i'_j}))$.

We will call this the *min work* mechanism. It can be viewed as auctioning each task individually using the Vickrey auction rule. Because the final utility of each agent is the sum of its utilities from each step, it is clear that the mechanism is truthful. Nisan and Ronen [10] show that this mechanism gives an n-approximation for the task scheduling problem.

6 Computational Limits of Mechanisms

In the previous section we gave only an approximation mechanism for the task scheduling problem. This is because within the model presented in Section 3, a truthful mechanism that gives an optimal solution does not exist. In this section we will show that no mechanism within the limits of this model do better than a 2-approximation [10].

Recall that Theorem 8 (part 2) states that if a mechanism is truthful, then for each agent i and every $\bar{t}_{-i}$, when $o \in \{o(\bar{t}_{-1}, t'_i) : t'_i \in T_i\}$,

$$o(\bar{t}_{-i}, t_i) \in \arg\max_o \big(v_i(t_i, o) + p_i\big) \text{ for all } t_i.$$

Lemma 18. *Let D be an arbitrary allocation of tasks to i and $t_i(D) = \sum_{j \in D} t_{i_j}$, now $v_i(t_i, o) = -t_i(D)$. In the context of the task scheduling problem, the above condition states that for all t_i when $D \in \{x_i(\bar{t}_{-i}, t'_i) : t'_i \in T_i\}$,*

$$x_i(\bar{t}_{-i}, t_i) \in \arg\max_D \big(p_i(D, \bar{t}_{-i}) - t_i(D)\big).$$

It follows that for every i and $\bar{t}_{-i}$, if a truthful mechanism allocates a set of tasks X to i, then the following inequalities must hold for every t_i and $D \in \{x_i(\bar{t}_{-i}, t'_i) : t'_i \in T_i\}$ as well.

$$\begin{aligned} p_i(X, \bar{t}_{-i}) - t_i(X) &\geq p_i(D, \bar{t}_{-i}) - t_i(D) && \Leftrightarrow \\ p_i(X, \bar{t}_{-i}) - t_i(X) + t_i(X \cap D) &\geq p_i(D, \bar{t}_{-i}) - t_i(D) + t_i(X \cap D) && \Leftrightarrow \\ p_i(X, \bar{t}_{-i}) - t_i(X \setminus D) &\geq p_i(D, \bar{t}_{-i}) - t_i(D \setminus X) \end{aligned}$$

We look specifically at task-scheduling problems with two agents ($n = 2$) and three or more tasks ($k \geq 3$). Problems where $n > 2$ reduce to this case when the additional agents execute tasks much slower than 1 and 2. Let both agents have the same execution time for every task, so the type vector $\bar{t}$ is defined by

$$t_{i_j} = 1 \text{ for } i \in \{1, 2\} \text{ and } j \in \{1, \ldots, k\}.$$

The optimal solution in this case is clearly to divide the tasks equally between the two agents. If k is odd, one agent will have an extra task assigned to it so we assume arbitrarily that $|x_1(\bar{t})| \leq |x_2(\bar{t})|$. Let $0 < \epsilon < 1$, we now define a new type vector $\hat{t}$ by assigning

$$\hat{t}_{i_j} = \begin{cases} \epsilon & \text{if } i = 1 \text{ and } j \in x_1(\bar{t}) \\ 1 + \epsilon & \text{if } i = 1 \text{ and } j \in x_2(\bar{t}) \\ 1 & \text{if } i = 2. \end{cases}$$

Lemma 19. *When $\bar{t}$ and $\hat{t}$ are as defined above, $x(\bar{t}) = x(\hat{t})$. That is, both type vectors produce the same allocation of tasks.*

Proof. Because $n = 2$, the allocation $x(\bar{t})$ is uniquely defined by $x_1(\bar{t})$ so it is sufficient to show that $x_1(\bar{t}) = x_1(\hat{t})$. Lemma 18 gave us an inequality that holds for all agents i, we now apply it specifically to agent 1 and the type vector $\bar{t}$:

$$p_1(x_1(\bar{t}), \bar{t}_{-1}) - t_1(x_1(\bar{t}) \setminus D) \geq p_1(D, \bar{t}_{-1}) - t_1(D \setminus x_1(\bar{t})).$$

We can see that $x_1(\bar{t}) \setminus D$ contains only tasks in $x_1(\bar{t})$ so from the definition of $\hat{t}$ it follows that $t_i(x_1(\bar{t}) \setminus D) < \hat{t}_i(x_1(\bar{t}) \setminus D)$. Similarly, $D \setminus x_1(\bar{t})$ contains only tasks in $x_2(\bar{t})$ so it follows that $t_i(D \setminus x_1(\bar{t})) > \hat{t}_i(D \setminus x_1(\bar{t}))$ and the inequality becomes strict. Additionally, $\bar{t}_{-1} = \hat{t}_{-1}$ because the definition of $\hat{t}$ only changes the type of agent 1 so we can see that $x_1(\bar{t})$ satisfies

$$p_1(x_1(\bar{t}), \hat{t}_{-1}) - \hat{t}_1(x_1(\bar{t}) \setminus D) > p_1(D, \hat{t}_{-1}) - \hat{t}_1(D \setminus x_1(\bar{t})).$$

Applying lemma 18 to agent 1 and the type vector $\hat{t}$ we see that $x_1(\hat{t})$ must satisfy

$$p_1(x_1(\hat{t}), \hat{t}_{-1}) - \hat{t}_1(x_1(\hat{t}) \setminus D) \geq p_1(D, \hat{t}_{-1}) - \hat{t}_1(D \setminus x_1(\hat{t})).$$

We just showed that $x_1(\bar{t})$ strictly satisfies this same inequality so it must be that $x_1(\hat{t}) = x_1(\bar{t})$. □

Theorem 20. *There does not exist a mechanism that implements a better result than a 2-approximation for the task scheduling problem.*

Proof. The proof follows from lemma 19 but examines $x_2(\bar{t})$ instead of $x_1(\bar{t})$ because $|x_1(\bar{t})| \leq |x_2(\bar{t})|$. We prove the claim for when an even number of tasks are allocated to agent 2 and show that the same applies for an odd amount as well.

1. Even $|x_2(\bar{t})|$: The make-span for the allocation $x(\hat{t})$ given by a truthful mechanism is
$$g(x(\hat{t}), \hat{t}) = |x_2(\hat{t})| = |x_2(\bar{t})|.$$
However, the allocation that gives agent half of the tasks in $x_2(\bar{t})$ in addition to the tasks in $x_1(\bar{t})$ results in a make-span of $\frac{1}{2}|x_2(\bar{t})| + k \cdot \epsilon$. The make-span of the optimal allocation $opt(\hat{t})$ can be no greater than this, so
$$g(opt(\hat{t}), \hat{t}) \leq \frac{1}{2}|x_2(\bar{t})| + k \cdot \epsilon.$$
Thus, $\frac{g(x(\hat{t}),\hat{t})}{g(opt(\hat{t}),\hat{t})} \to c$ as $\epsilon \to 0$ for some $c \leq 2$.

2. Odd $|x_2(\bar{t})|$: We choose an arbitrary $j \in |x_2(\bar{t})|$ and consider a type vector where $\hat{t}_{2_j} = \epsilon$. This yields the same allocation as above.

□

7 Randomized Mechanisms

Randomization has been used to produce computationally feasible truthful mechanisms that give reasonable approximations mechanism design problems such as combinatorial auctions [2, 6]. In this section we show that the lower bound of a 2-approximation for the task scheduling problem can be improved upon using randomization. A randomized mechanism is a distribution over deterministic mechanisms. The deterministic mechanisms produced have the same n agents and the same set of possible outcomes but vary in their payment functions and outcomes. There are two common definitions of truthfulness for randomized mechanisms [12].

Definition 21. A randomized mechanism is *truthful in expectation* if truth-telling is a dominant strategy for the expected mechanism and problem. This means that for every agent i, all $\bar{t}_{-i}$, t_i, and false declaration t'_i,

$$E[v_i(o) + p_i] \geq E[v_i(o') + p'_i]$$

where (o, p_i) and (o', p'_i) are random variables for the outcome and payment when i reports t_i and t'_i.

Definition 22. A randomized mechanism is *universally truthful* if every deterministic mechanism that could be produced by the randomized mechanism is truthful. This is the stronger condition and more analogous with truthfulness in deterministic mechanisms.

Before we introduce the randomized mechanism, consider a biased version of the min work mechanism introduced in Section 5.2 for two agents. For each task j, the mechanism is biased towards one of the agents such that either

1. j is assigned to 1 when $t_{1_j} \leq \beta \cdot t_{2_j}$ and 1 is paid $\beta \cdot t_{2_j}$ and when $t_{1_j} > \beta \cdot t_{2_j}$, the task is assigned to 2 and 2 is paid $\beta^{-1} \cdot t_{1_j}$ or
2. j is assigned to 1 when $\beta \cdot t_{1_j} \leq t_{2_j}$ and 1 is paid $\beta^{-1} \cdot t_{2_j}$ and when $\beta \cdot t_{1_j} > t_{2_j}$, the task is assigned to 2 and 2 is paid $\beta \cdot t_{1_j}$.

This is equivalent to a weighted VCG mechanism with weights $\{1, \beta\}$ or $(\beta, 1)$ which we know to be truthful. The biased min work mechanism is truthful regardless of which agent a task is biased towards so it follows that a randomly biased min work mechanism is universally truthful. Nisan and Ronen [10] show that a randomly biased min work mechanism where $\beta = \frac{4}{3}$ gives a $\frac{7}{4}$-approximation for the task scheduling problem with two agents, which is below the lower bound for deterministic mechanisms.

8 Extending the Model

The basic model for mechanism design we presented is very limited in what real-world circumstances it can represent. In particular, the model was not designed with computing environments in mind so it seems natural to try to extend the model to better reflect the situations in which we use it. In this section we give a brief overview of two such extensions to the model.

8.1 Mechanisms with verification

In order to create a mechanism capable of producing an optimal outcome for the task scheduling problem, [10] introduces mechanisms with verification. A *mechanism with verification* adds an additional execution phase to the basic model for mechanisms. This execution phase takes place after the agents have made their declarations and the mechanism has allocated the tasks, but before payments are made. The purpose of the execution phase is to give the mechanism a way to learn the actual times in which the agents completed the tasks assigned to them. We use $\tilde{t} = (\tilde{t}_1, \ldots, \tilde{t}_k)$ to denote the execution times of the tasks. This additional information allows us to create a truthful mechanism with an optimal outcome.

Definition 23 (Corrected time vector)**.** When i is an agent, x is an allocation of tasks, $\tilde{t}$ is the execution times of the tasks, and $\bar{t}$ is the declarations of the agents, the corrected time vector for i is a vector of execution times for each task $corr_i(x, \bar{t}, \tilde{t})$ defined by

$$corr_i(x, \bar{t}, \tilde{t})_j = \begin{cases} \tilde{t}_j \text{ if } j \in x_i(\bar{t}) \\ t_{l_j} \text{ if } j \in x_l(\bar{t}) \text{ and } l \neq i. \end{cases}$$

The vector is based on the actual execution times of i and the declarations of every other agent.

Definition 24 (Compensation and bonus mechanism [10])**.** The compensation and bonus mechanism for the task scheduling problem is defined by an optimal allocation algorithm and payment functions $p_i(\bar{t}, \tilde{t}) = c_i(\bar{t}, \tilde{t}) + b_i(\bar{t}, \tilde{t})$ composed of

1. the compensation
$$c_i(\bar{t}, \tilde{t}) = \sum_{j \in x_i(\bar{t})} \tilde{t}_j$$
which simply pays the agent for the amount of work it did and

2. the bonus
$$b_i(\bar{t}, \tilde{t}) = -g(x(\bar{t}), corr_i(x(\bar{t}), \bar{t}, \tilde{t}))$$
which is the negation of the make-span when calculated based on a corrected time vector for i.

Nisan and Ronen show in [10] that the compensation and bonus mechanism is a truthful implementation of the task scheduling problem. They also give a generalization of the mechanism and use it to make a mechanism that can give an arbitrarily good approximation that can be computed in polynomial time for a bounded version of the task scheduling problem.

8.2 Feasible truthfulness

A different approach to extending the mechanism design model is taken in [11]. We know that *VCG-based* mechanisms which use the VCG payment function but a non-optimal algorithm to determine the outcome are not necessarily truthful. However, if the mechanism cannot compute the optimal outcome for a problem in a reasonable amount of time, it may well be the case that the agents cannot compute their optimal strategies in a reasonable amount of time either.

Definition 25. We can represent an agent's *strategic knowledge* of the problem as a function

$$b_i : A_{-i} \to A_i$$

which maps a tuple of the other agents' chosen strategies $\bar{a}_{-i}$ to the strategy a_i that the agent thinks will give him the best result in that situation.

Definition 26 (Feasibly dominant strategy)**.** Taking the agent's knowledge into account, we can define a strategy a_i to be *feasibly dominant* if

$$u_i(\bar{a}_{-i}, b_i(\bar{a}_{-i})) \leq u_i(\bar{a}_{-i}, a_i)$$

for every $\bar{a}_{-i}$. If agent i's knowledge b_i of the problem is optimal, then a feasibly dominant strategy is also dominant in the standard sense.

Definition 27 (Feasible truthfulness)**.** A mechanism is *feasibly truthful* if truth-telling is a feasibly dominant strategy for every agent.

Even though a VCG-based mechanism may not be truthful, the VCG payment function is structured so that the only reason for a rational agent to lie is to improve the overall result of the mechanism. The drawback here is that the agent could be wrong and make the result worse instead. This motivates us to give the agents a way to tell the mechanism about an alternative to a truthful declaration and let the mechanism choose the better of the two.

Definition 28 (Appeal function)**.** An appeal is a function that an agent i submits to the mechanism in addition to a type declaration,

$$l : \prod_{i=1}^{n} T_i \to \prod_{i=1}^{n} T_i$$

the purpose of which is to tell the mechanism that if type declarations $\bar{t}$ are submitted to the mechanism, agent i thinks that $l(\bar{t})$ would produce a better overall result.

We now define a VCG-based mechanism which makes use of these appeal functions and does not require an optimal output function. For an output function o, this *second-chance mechanism* is composed of the following interactions:

1. Every agent declares a type t_i and uses its knowledge b_i to create an appeal function l_i to send to the mechanism.

2. The mechanism computes $o(\bar{t}), o(l_1(\bar{t})), \ldots, o(l_n(\bar{t}))$ and chooses the outcome from these o that maximizes $\sum_{i=1}^{n} v_i(t_i, o)$.

3. Agents are given normal VCG payments, $p_i(\bar{t}) = \sum_{j \neq i} v_j(t_j, o(\bar{t})) + h_i(\bar{t}_{-i})$.

It is clear that when the agents are truthful the output is no worse than $o(\bar{t})$ so the mechanism gives a result at least as good as the output function it is based on. Nisan and Ronen show in [11] that the second-chance mechanism is feasibly truthful even when we require that all computations done by the mechanism and the agents can be done in polynomial time.

9 Conclusions

In this text we described a model for mechanism design problems and defined the general VCG mechanism in that context. We introduced some potential applications of mechanism design in the field of computer science and explored the disadvantages inherent to that approach. To alleviate these disadvantages, we introduced randomized mechanisms and other possible extensions to the basic mechanism design model.

Overall, we have provided only a cursory glance at the possible uses of mechanism design in computer science. Other important considerations include implementing these mechanisms in distributed environments [13, 4] and creating the protocols that make such implementations possible [7]. Mechanism design solutions that do not involve money [12, 14] or have a limited budget [8] have also been studied.

References

[1] Clarke, Edward H.: *Multipart pricing of public goods.* Public Choice, 11:17–33, 1971.

[2] Dobzinski, Shahar, Nisan, Noam, and Schapira, Michael: *Truthful randomized mechanisms for combinatorial auctions.* In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 644–652. ACM, 2006.

[3] Easley, David and Kleinberg, Jon: *Networks, Crowds, and Markets.* Cambridge University Press, 2010.

[4] Grosu, Daniel and Chronopoulos, Anthony T.: *Algorithmic mechanism design for load balancing in distributed systems.* IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 34(1):77–84, 2004.

[5] Groves, Theodore: *Incentives in Teams.* Econometrica, 41:617–631, 1973.

[6] Lavi, Ron and Swamy, Chaitanya: *Truthful and Near-Optimal Mechanism Design via Linear Programming.* Journal of the ACM, 58(6):25:1–25:24, 2011.

[7] Monderer, Dov and Tennenholtz, Moshe: *Distributed Games: From Mechanisms to Protocols.* In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, AAAI '99, pages 32–37. AAAI, 1999.

[8] Monderer, Dov and Tennenholtz, Moshe: *k-Implementation.* In *Proceedings of the Fourth ACM Conference on Electronic Commerce*, EC '03, pages 19–28. ACM, 2003.

[9] Nisan, Noam: *Algorithms for selfish agents: Mechanism design for distributed computation.* In *Proceedings of the Sixteenth Annual Symposium on Theoretical Aspects of Computer Science*, STACS '99, pages 1–15. Springer Berlin Heidelberg, 1999.

[10] Nisan, Noam and Ronen, Amir: *Algorithmic mechanism design.* In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 129–140. ACM, 1999.

[11] Nisan, Noam and Ronen, Amir: *Computationally feasible VCG mechanisms.* In *Proceedings of the Second ACM Conference on Electronic Commerce*, EC '00, pages 242–252. ACM, 2000.

[12] Nisan, Noam, Roughgarden, Tim, Tardos, Eva, and Vazirani, Vijay V.: *Algorithmic Game Theory.* Cambridge University Press, 2007.

[13] Parkes, David C. and Shneidman, Jeffrey: *Distributed Implementations of Vickrey-Clarke-Groves Mechanisms.* In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '04, pages 261–268. IEEE Computer Society, 2004.

[14] Procaccia, Ariel D. and Tennenholtz, Moshe: *Approximate mechanism design without money.* In *Proceedings of the Tenth ACM conference on Electronic Commerce*, EC '09, pages 177–186. ACM, 2009.

[15] Vickrey, William: *Counterspeculation, auctions, and competitive sealed tenders.* The Journal of Finance, 16:8–37, 1961.