

Solving Optimization Problems Among Selfish Agents

Thesis submitted for the degree “Doctor of Philosophy”

Amir Ronen

Submitted to the Senate of the Hebrew University in Jerusalem (2000)

This work was carried out under the supervision of
Prof. Noam Nisan

Acknowledgment

I had the pleasure and the privilege to have professor Noam Nisan as my advisor. It is hard to exaggerate about how happy I am about this. Noam – your brilliance, "Manuel Blum's" style of guiding, creativity, energy and of course personality, made this era of my life such a great experience. Maybe above all I am grateful for the time you have spent teaching me so much about thinking. So thanks Noam, lets stay in touch!

A special thank to professor Motty Perry from the Rationality center who devoted a lot of time guiding us in the paths of mechanism design literature. Our discussions over the years have greatly shaped the course of my thesis.

Finally to my beloved family, Inbally, Tal and the ones to follow – I dedicate this to you!

Prologue – some food for thought

Before commencing, the reader is welcome to spend some time considering the following story: You are given a set of tasks which you would like to allocate, say over the Internet, to a group of people whom you are not familiar with and who may not necessarily cooperate with you. Your goal is to minimize the completion time of the last assignment (the make-span). What would you do?

Contents

Abstract	v
1 Introduction	1
1.1 Background	4
1.1.1 Game theory	4
1.1.2 Mechanism Design	5
1.1.3 Some fundamental concepts of theoretical computer science	6
1.1.4 Game theoretic considerations in computer science	8
1.2 Overview of the results	11
1.2.1 Algorithmic Mechanism Design	11
1.2.2 Computationally Feasible VCG Mechanisms	12
2 Algorithmic Mechanism Design	13
2.1 Introduction	15
2.1.1 Motivation	15
2.1.2 This Work	16
2.1.3 Extant Work	17
2.2 The Model	18
2.2.1 Mechanism Design Problem Description	18
2.2.2 The Mechanism	19
2.2.3 The Revelation Principle	20
2.3 Vickrey-Groves-Clarke Mechanisms	21
2.3.1 Utilitarian Functions	21
2.3.2 Example: Shortest Paths	22
2.4 Task Scheduling	23
2.4.1 The Problem	23
2.4.2 An Upper Bound	24

2.4.3	Lower Bounds	25
2.4.4	The Power of Randomization	29
2.5	Mechanisms with Verification	33
2.5.1	Mechanisms with Verification	34
2.5.2	A Reformulation of Task Scheduling	34
2.5.3	The Compensation-and-Bonus Mechanism	35
2.5.4	The Generalized Compensation-and-Bonus Mechanism	37
2.5.5	Budget Considerations	37
2.5.6	Poly-Time Mechanisms	39
2.6	Conclusions and Further Research	41
3	Computationally Feasible VCG Mechanisms	43
3.1	Introduction	45
3.2	Preliminaries	47
3.2.1	Mechanism design problems	48
3.2.2	VCG-based mechanisms	49
3.2.3	Computational considerations in mechanism design	50
3.2.4	Example: Combinatorial auctions	50
3.3	Limitations of truthful VCG-based mechanisms	51
3.3.1	Truthful VCG-based mechanisms for combinatorial auctions	52
3.3.2	Truthful VCG-based mechanisms for cost minimization problems	55
3.4	Computationally feasible VCG mechanisms	57
3.4.1	Feasibly dominant actions	58
3.4.2	The second-chance mechanism	60
3.4.3	Existence of computationally efficient feasibly dominant truth-telling actions	62
3.4.4	Other implementation issues	65
3.4.5	Summary of the method	66
4	Discussion	68
4.1	Conclusions and further research	68
4.2	Applications	69
A	A note about single object allocations	81

Abstract

A large part of research in computer science is concerned with protocols and algorithms for inter-connected collections of computers. The designer of such an algorithm or protocol always makes an implicit assumption that the participating computers will act as instructed – except, perhaps, for the faulty or malicious ones. While this assumption seems obvious for “traditional” applications programmed by a single entity, it cannot be taken for granted when considering, for example, applications which are intended to operate on the Internet. Many of these applications involve self-interested participants (e.g. individual users and private companies). Such participants have their *own* goals and preferences and are likely to follow their own self interest. The goal of this thesis is to develop a theory that will enable the construction of algorithms and protocols that perform well when the participants behave selfishly.

Applications that involve self-interested parties emerge from several domains: Participants in an electronic trade pursue their personal profit, users of a wide area network want their own traffic to be optimized and software agents are self-interested by definition. Problems that stem from such applications are essentially different from traditional algorithmic problems. To make this difference concrete, consider an imaginary system aimed to utilize resources on the Internet:

Ideally, such a system would function like a regular distributed system. It would efficiently allocate CPU-intensive jobs to CPU-servers, store data in computers with free disk space, extract necessary information from various available databases and combine the services of the many software packages found on the Internet. Beyond many difficult coordination and optimization problems which need to be overcome, such a system faces a major novel challenge: All these resources *belong* to self-interested parties (such as private companies or institutions) and there is no a-priori reason that these parties will allow the system to freely use them, in particular when they are costly. The system may, thus, need to provide some motivation for these owners to “play along”. Supplying such motivation is likely to involve some sort of payment. Payment, real or virtual, carries however problems of its own. A payment model which is not carefully designed can easily lead into a situation where manipulating the system’s algorithms and protocols can significantly increase the profits of the participants (for example, participants may pretend that they are not capable to perform tasks which are not profitable enough). Such manipulations might severely damage the efficiency of such a system, preventing it to accomplish the purposes which it was designed for. As we shall see, the algorithmic and the monetary aspects of protocols for such systems are tightly coupled. Any solution for an algorithmic problem that involves selfish participants should simultaneously deal with these two aspects. Solving such problems is therefore a difficult and exciting challenge!

Mechanism design is a sub field of game theory and micro economics which deals with the design of protocols for rational agents. Generally, a mechanism design problem can be described as the task of selecting from a collection of feasible games, a game which will yield desirable results for the designer. Specifically, the theory has focused on problems where the goal is to satisfactorily aggregate privately known preferences of several agents towards a “social choice”.

An application of this theory within the framework of computer science is by no means straightforward. There are two main reasons for that:

Different goals Traditionally, mechanism design theory was developed for applications that arise from economics. Naturally, problems that stem from computer science applications have different goals and assumptions.

Computational complexity Mechanism design theory ignores the computational aspect of the protocol. This turns essentially all solutions proposed by this theory to be impractical for computationally complex problems.

The approach taken in this work is based on mechanism design as well as on basic concepts from theoretical computer science. In particular I have chosen to focus on solutions in dominant strategies, i.e. strategies in which it is always for the benefit of the participants to choose them.

This thesis is based on the following papers: “Algorithmic Mechanism Design”, that appeared in STOC99 and in Games and Economic Behavior and “Computationally Feasible VCG Mechanisms” that will appear in the second ACM Conference on Electronic Commerce (EC-00). Both papers are co-authored with my advisor, Professor Noam Nisan. In the discussion and the appendix I have also included some material from an invited paper for the 27th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM2000) which is currently in preparation.

Overview of the results

The results in this thesis naturally divide into two main parts. The first, concerns the possibility (or impossibility) to develop strategy-proof mechanisms (i.e. mechanisms with dominant strategies for the agents) for problems that involve rational agents under various assumptions. The second, studies the computational aspect of such mechanisms.

This thesis shows, among the rest, that many problems that naturally stem from computer science applications can be satisfactorily handled. When the compensation and bonus mechanism developed in the first part is unified with the second chance mechanism developed in the second, one obtains a generic method for handling a wide variety of problems where verification assumptions

are available. In this method the designer develops an *algorithm* for the corresponding optimization problem and automatically obtains a protocol with the following properties:

- The *only* rational strategy for the agents is to cooperate with the algorithm. (A precise definition of the solution concept appears in the second part of this work.)
- When the agents act rationally, the mechanism performs *at least as good* as the algorithm.
- If the algorithm is polynomial then so is the mechanism.
- Participation constraints are satisfied.
- The designer never pays more than her budget.
- The protocol is simple from the agents' perspective.

Algorithmic Mechanism Design

This part studies the possibility or impossibility to solve problems that involve self-interested participants in dominant strategies. It investigates a representative task scheduling problem where the standard tools from mechanism design do not suffice. The results obtained in this part can be divided into three main parts:

Limitations of the classical model This part shows lower bounds on the approximation ratio that can be obtained by any deterministic strategy-proof mechanism for the representative problem.

Probabilistic mechanisms This part shows how to break the deterministic lower bounds using randomization without weakening the game theoretic strength of the solution concept.

The power of verification assumptions This part studies strategy-proof mechanisms for the representative problem under additional verification assumptions which are natural to many problems. It introduces the *compensation and bonus* mechanism that can solve such problems *optimally*. Moreover the mechanism supports budget constraints. A method for developing a polynomial-time mechanisms is also suggested. This method basically works in cases where dynamic programming works.

Computationally Feasible VCG Mechanisms

This part studies the *computational* aspect of strategy-proof mechanisms. It is divided into two main parts:

Limitations of the existing methods This part shows that all known general methods for the construction of strategy-proof mechanisms fail to construct reasonable, computationally feasible mechanisms. In particular it shows gap phenomena for the important general problem of combinatorial auctions as well as for many cost minimization problems. Any truthful mechanism constructed in the existing methods can either be optimal (and henceforth computationally infeasible) or produce unreasonable results.

The second-chance mechanism This part proposes a method to overcome this non-truthfulness. It introduces a *feasible* notion of truthfulness and shows a general way to construct mechanisms that are feasibly truthful under reasonable assumption on the agents. The mechanism has several desirable properties.

The solution concept suggested in this part is considerably different from previous work on bounded rationality and constitutes an important part of this thesis.

Chapter 1

Introduction

A large part of research in computer science is concerned with protocols and algorithms for interconnected collections of computers. The designer of such an algorithm or protocol always makes an implicit assumption that the participating computers will act as instructed – except, perhaps, for the faulty or malicious ones. While this assumption seems obvious for “traditional” applications programmed by a single entity, it cannot be taken for granted when considering, for example, applications which are intended to operate on the Internet. Many of these applications involve self-interested participants (e.g. individual users and private companies). Such participants have their *own* goals and preferences and are likely to follow their own self interest. The goal of this thesis is to develop a theory that will enable the construction of algorithms and protocols that perform well when the participants behave selfishly.

Applications that involve self-interested parties emerge from several domains: Participants in an electronic trade pursue their personal profit, users of a wide area network want their own traffic to be optimized and software agents are self-interested by definition. Problems that stem from such applications are essentially different from traditional algorithmic problems. To make this difference concrete, consider an imaginary system aimed to utilize resources on the Internet:

Ideally, such a system would function like a regular distributed system. It would efficiently allocate CPU-intensive jobs to CPU-servers, store data in computers with free disk space, extract necessary information from various available databases and combine the services of the many software packages found on the Internet. Beyond many difficult coordination and optimization problems which need to be overcome, such a system faces a major novel challenge: All these resources *belong* to self-interested parties (such as private companies or institutions) and there is no a-priory reason that these parties will allow the system to freely use them, in particular when they are costly. The system may, thus, need to provide some motivation for these owners to “play along”. Supplying such motivation is likely to involve some sort of payment. Payment, real or virtual, carries however

problems of its own. A payment model which is not carefully designed can easily lead into a situation where manipulating the system's algorithms and protocols can significantly increase the profits of the participants (for example, participants may pretend that they are not capable to perform tasks which are not profitable enough). Such manipulations might severely damage the efficiency of such a system, preventing it to accomplish the purposes which it was designed for. As we shall see, the algorithmic and the monetary aspects of protocols for such systems are tightly coupled. Any solution for an algorithmic problem that involves selfish participants should simultaneously deal with these two aspects. Solving such problems is therefore a difficult and exciting challenge!

Mechanism design is a sub field of game theory and micro economics which deals with the design of protocols for rational agents. Generally, a mechanism design problem can be described as the task of selecting from a collection of feasible games, a game which will yield desirable results for the designer. Specifically, the theory has focused on problems where the goal is to satisfactorily aggregate privately known preferences of several agents towards a "social choice".

An application of this theory within the framework of computer science is by no means straightforward. There are two main reasons for that:

Different goals Traditionally, mechanism design theory was developed for applications that arise from economics. Naturally, problems that stem from computer science applications have different goals and assumptions.

Computational complexity Mechanism design theory ignores the computational aspect of the protocol. This turns essentially all solutions proposed by this theory to be impractical for computationally complex problems.

The approach taken in this work is based on mechanism design as well as on basic concepts from theoretical computer science. In particular I have chosen to focus on solutions in dominant strategies, i.e. strategies in which it is always for the benefit of the participants to choose them.

This work is divided into two main parts. The first one studies, among others, the limitations of the "classical" model of mechanism design and significantly extends the class of problems that can be *optimally* solved in dominant strategies. The second part studies the computational aspect of such problems. It shows that it is impossible, using all general methods known to date, to construct computationally feasible mechanisms that produce reasonable results and proposes a novel method of doing so.

The thesis is based on the following papers: "Algorithmic Mechanism Design", that appeared in STOC99 [77] and in Games and Economic Behavior [78] and "Computationally Feasible VCG

Mechanisms” [79] that will appear in the second ACM Conference on Electronic Commerce (EC-00). Both papers are co-authored with my advisor, Professor Noam Nisan. In the discussion and the appendix I have also included some material from an invited paper for the 27th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM2000) [90] which is currently in preparation.

1.1 Background

1.1.1 Game theory

Game theory may be described as a mathematical framework for the analysis of the concept of rational decisions and of the interaction among rational decision makers. The originators of the theory are considered to be Von Neumann and Morgenstern [113], although some of its roots stem from earlier work, in particular of Zermello [13]. Game theory has influenced many scientific disciplines, ranging from pure mathematics (e.g. [106], [71, chapter 2]) to biology. Arguably the most influenced discipline is Economic theory (probably because humans are particularly rational about their money...). An introduction to this field can be found in several textbooks (e.g. [22, 81]). More advanced material may be found in a recent handbook [4].

This work analyses protocols from a non-cooperative game theory perspective. In particular, it focuses on solutions in dominant strategies. I.e. strategies in which it is *always* for the benefit of the agents to choose them. Although weaker solution concepts are commonly studied in the literature (e.g Nash [74] and Bayesian Nash equilibria [32]), solutions in dominant strategies, when available, are much preferable in particular for Internet applications [101].

An important ingredient of problems that are analyzed in this work is the fact that agents have private input. This ingredient is modeled in this work by the standard framework of games with incomplete information (see e.g. [60, pp. 253-260]).

Another fundamental aspect of problems which are studied in this thesis is computational. It is actually twofold. First, it is necessary to develop protocols that can be executed in a reasonable amount of time. The second aspect is related to the decision making process of the agents: Making good decisions in a game is usually a very complex task (in many cases PSPACE complete [98]). Therefore, it is not reasonable to expect that the agents will be able to make the optimal decisions as implicitly assumed by traditional game theory. A sub-field of game theory called bounded rationality investigates games which are played by agents with limited reasoning capabilities. While currently not much is known, in particular about games which are played by polynomial time Turing machines, chapter 3 of this thesis was influenced by the bounded rationality literature. A recent survey of this topic may be found in [92].

Finally I would like to mention two sub-fields of game theory which are not studied in this thesis but seem fundamental to future research on this topic: In a repeated game setting the game is not played once but repeated finitely or infinitely many times. This has a strong effect on the nature of the game and should be considered in the context of mechanism design. The theory of cooperative (or coalitional) games deals with some aspects of cooperation among agents. Currently collusion

among agents is hardly addressed even in the "non-computational" mechanism design literature. An introduction to both topics can be found in [81].

1.1.2 Mechanism Design

Mechanism design (a.k.a. implementation theory) is a sub-field of game theory and micro-economics which studies the design of protocols with good game theoretical properties. This theory has many applications including the design of voting schema [5], taxes [41] and auctions [43]. The theory puts an emphasis on economic applications and in particular on auction design. An introduction to this field can be found in [60, 81, 82].

Historically, this theory has stemmed from several sub-fields of micro-economics and in particular from the theory of social choice [3]. In the last few years this field has received much attention, especially due to its influence on large governmental privatizations and auctions [10, 66, 65, 62, 119].

Generally, a mechanism design problem can be described as the task of selecting from a collection of feasible games, a game that will yield desirable results for the designer. Specifically, the theory has focused on the following canonical problem: A set of agents needs to collaboratively choose an outcome x from a set $\mathcal{X}$ of possible outcomes. Each agent i has a *privately known* utility function $u^i : \mathcal{X} \rightarrow R$ (or more generally a preference relation). There is an output specification F that corresponds to every tuple u of utilities, a subset $F(u)$ of allowed outcomes. Given a solution concept, the goal of the designer is to define a protocol (a game) with solutions that meet the specification.

Mechanism design models differ from each other by the solution concept they require and by additional assumptions on the agents. In this work I have focused on the standard independent private value model, the quasi-linearity assumption and in particular on dominant strategy implementation.

Implementation in dominant strategies

This subsection describes some fundamental results concerning dominant strategy implementation which are relevant to this work.

In a revelation mechanism the agents are simply required to reveal their types (utilities) to the mechanism. Such a mechanism can be denoted by a pair $(k(u), p(u))$ where $k(\cdot)$ is an output function and $p(\cdot)$ a payment function. A revelation mechanism is called *truthful* or *incentive compatible* if truth-telling is dominant for all agents. A simple observation known as the revelation principle ([60, page 871]) states that in order to prove or disprove the ability to implement a given specification

function, it is enough to consider only revelation mechanisms¹. This observation greatly simplifies the analysis of mechanism design problems.

A celebrated theorem by Gibbard [25] and Satterthwaite [97] shows that without further assumptions on the type space, one cannot implement even the most basic functions (e.g. choosing between three alternatives). This theorem, tightly connected to a remarkable theorem of Arrow (in [3]), greatly shaped the course of the research in this area. In particular it caused researchers to focus on assumptions that facilitate implementation in dominant strategies and are feasible to many applications. Arguably the most important and popular among these assumptions is quasi linearity. In this assumption, also taken in this work, the mechanism can use monetary transformations and the agents' utilities are assumed to be of the form $v^i(x) + p^i$ where p^i denotes the monetary payment to the agent and $v^i(\cdot)$ depends only on $x \in \mathcal{X}$.

Under the quasi linearity assumption, any function of the form $\arg \max_{x \in \mathcal{X}} v^0(x) + \sum_i \beta_i \cdot v^i(x)$ (where β_i are non-negative weights and $v^0(\cdot)$ is any function chosen by the mechanism) can be implemented using the celebrated VCG mechanisms [112, 8, 31]. It is known that under mild conditions a mechanism's output function uniquely determines its payment rule (up to some normalization). This was first proven in [30] for a private case. A much more difficult question is the characterization of the output functions that can be extended to strategy-proof mechanisms (i.e. dominant strategy implementation). A beautiful theorem by Roberts [89] shows that if nothing apart from quasi-linearity can be assumed on the agents' type space, then VCG mechanisms are the only truthful (revelation) mechanisms. To the best of my knowledge this is the only known result regarding this question [61]. In appendix A I give a natural example where for which the family of implementable functions is very large. A deeper understanding of this question seems critical to the understanding of the ability to develop computationally feasible strategy-proof mechanisms.

1.1.3 Some fundamental concepts of theoretical computer science

This section describes, for the sake of completeness, some fundamental concepts of theoretical computer science. An introduction to this field can be found in many textbooks (e.g. [104]).

A *language* over an alphabet Σ is simply a subset of Σ^* , i.e. a set of finite sequences of letters. Much of the work in theoretical computer science can be described as an effort to understand, given a language, how hard it is to decide whether a string x belongs to it or not. Computer scientists make a sharp distinction between two fundamental classes of languages called P and NPC .

The class P contains all the "easy" languages. I.e. those which can be decided by an algorithm²

¹When demanding that all the solution points will maintain the problem's specification this statement usually remains true but not always.

²Formally an algorithm is a Turing machine. In this level of granularity however, there is not much difference

in time which is bounded by some polynomial in the length of the input. An example for such a language is one that contains all pairs (K, m) such that K is a set of integers and m the maximum among them.

The class NP contains all languages that can be decided by a polynomial-time non-deterministic algorithm. If L is an NP language then for every $x \in L$ there exists a (polynomially short) witness w , such that given w , verifying that $x \in L$ is easy. An example for an NP language is SAT – the set of all Boolean formulas (using some standard encoding) which have a satisfying assignment. Given a satisfying assignment, verifying that $x \in SAT$ can be done in polynomial time. A remarkable result ([9], and an equivalent result in [55]) states that if a polynomial-time algorithm for SAT do exist then *every* NP language can be solved in polynomial time! Languages with this property are called NP -hard. The class NPC (NP -complete) contains all NP language which are also NP -hard. Till date, no sub-exponential algorithm for any NPC languages has been found. It is widely conjectured that constructing a polynomial algorithm for any such language is indeed impossible. Unfortunately most of the problems of practical interest are NPC . [24] is considered as a canonical reference for the issue of NP -completeness.

A natural generalization of languages is the concept of optimization problems. An optimization problem is composed of a set I of *instances*, for every instance x , a set $S(x)$ of *feasible solutions*, and an *objective function* $g(x, s)$ that measures how good the solution $s \in S(x)$ is. Given an instance x , the goal of the algorithm is to find a feasible solution s that maximizes (or minimizes) $g(x, s)$. Such a problem is called NPO if recognizing that x is a valid instance, that s is a feasible solution for x and computing $g(x, s)$ can be done in time polynomial in the length $|x|$ of x . Max-3-sat is an example of an NPO : Given a CNF-formula where each term contains 3 literals (an instance), the goal of the algorithm is to find an assignment (feasible solution) that maximizes that number of satisfied terms (the objective function). It is not hard to verify that this problem is an NPO . Like in the case of languages, most problems tackled in everyday life seem to be hard (NPC).

One natural relaxation of such problems is to ask whether one can easily find a solution that is guaranteed to be within a certain factor of the optimum. For many NP -hard optimization problems the answer for this question is yes! A polynomial time algorithm which is guaranteed to be within a certain factor (or a function of $|x|$) of the optimum is called an *approximation*. In the last decade or so there has been a major breakthrough in the understanding of the possibility or impossibility of constructing approximation algorithms and many long-standing open problems were finally solved [34, 2, 1]. A recent book describing many results in this area is [35]. A constantly updated online compendium of results can be found in [42].

between all "reasonable" computational models, except the quantum ones.

1.1.4 Game theoretic considerations in computer science

The goal of this thesis is to facilitate the development of algorithms and protocols for self interested participants. While I am not aware of any work that pursued this direction, many previous works in computer science have adopted game theoretic and economic concepts. In this section I describe the main lines of such research.

Computational considerations in mechanism design

In recent years various complex auctions and mechanisms of interest were implemented on computer systems. This has led several researchers to consider the computational aspect of such mechanisms. Researchers were particularly motivated by the problem of combinatorial auctions. In such an auction the participants are interested in purchasing *combinations* of items (e.g. adjacent spectrum licensees, matching TV set and a DVD). Besides its intellectual interest, the problem has significant practical importance as governments conduct auctions which are combinatorial by nature raising billions of dollars (e.g. [62, 119]). Several researchers have studied algorithmic problems that stem from this application [53, 95, 23, 33, 75, 29]. A truthful mechanism for a sub-case of this problem which does not fall into the VCG category was constructed at [53]. Other cases where mechanism design theory has also been applied in computational settings include resource and task allocation problems [77, 117, 116], communication networks [19, 46], multi-agent systems [91, 120, 16] and others [114, 50, 20, 103, 56]. Various aspects of mechanism design were studied at [69, 68, 50, 102]. The amount of communication required by distributed mechanisms for cost sharing was studied at [19]. This recent paper investigates two ingredients absent in this thesis – a distributed model and budget balance requirement. These two fundamental aspects are surely to play a significant role in future research in this area. The implementation of mechanism design theory for Internet environments may require new assumptions. This issue is discussed at [94, 68]. Finally our work [77, 78, 79] can be classified into this category. It is worth mentioning that all the works cited in this section were focused on dominant strategy implementation.

Distributed artificial intelligence

The first community that brought game theory and micro economics into the center of its stage was arguably the Distributed Artificial Intelligence (DAI) community. DAI researchers study situations where autonomous agents interact. An agent can roughly be described as a computer program that acts on its owner's behalf. There are numerous aspects of such inter-agent communication. Among the investigated issues which are relates to game theory and economics are problems which stem from specific applications (e.g. [118, 115]), theories and platforms for agents' bargaining

contracting and negotiating (e.g. [105, 17, 47, 96, 48, 108]), agent economies and ecologies [72, 15] and market based control [20, 39, 64, 72]. Researchers from this community were also pioneering in the adoption of notions of mechanism design theory [91, 16, 120]. This line of work is covered in more detail in section 1.1.4. Extensive research is currently conducted in this area and multi-agent systems is nowadays a major subfield of artificial intelligence.

Communication Networks

Motivated by the overwhelming growth of the Internet, several researchers in the field of network design took a game theoretic approach. Research in this area includes Internet pricing ([100, 80, 111] and several papers in [40]), QoS and resource allocation [20, 93, 51, 87], and network control [45, 38, 12, 11, 7, 44, 101]. A game theoretic approach to the behavior of network users can be found in [21, 58]. A recent paper [19] studies a networking problem in the context of mechanism design.

Electronic commerce

The rapid growth of electronic commerce in recent years drew the attention of several researchers and institutions. Obviously many questions that stem from this area should take into consideration the self interest of the participants. Several papers on this topic can be found at [40, 14].

Theoretical computer science

Although tools from game-theory are essential for the analysis and even for the presentation of many problems in theoretical computer science (e.g. [71, chapter 2] [49]), computational models that involve self interested parties were hardly introduced in this community. Work on the border of theoretical computer science and game theory includes [46, 84, 85, 27, 63] and several papers from [107]. A survey of game theoretical aspects of computer science can be found in [57].

Additional topics

In this section I briefly describe two sub-fields of computer science that seem important for future research in mechanism design: distributed protocol design and cryptography.

While the canonical models of computation are central by nature, many computer systems are composed of distributed collections of computers and devices. The design of algorithms and protocols for such environments raises many fundamental questions. The field of distributed algorithm design is concerned with these issues. As many natural applications where mechanism design theory can play a significant role are distributed by nature [19, 68], distributed models are likely to

play an important role in the future research in mechanism design. [59] is a comprehensive book on this topic.

The theory of cryptography is concerned with communication and computation in the presence of adversaries [110][chapter 13]. Cryptographic protocols can ensure properties like privacy [88], authentication [28], and well behavior of the participants [6]. As information flow can greatly influence the agents' strategy space and because much of the applications of mechanism design involve a lot of money, cryptography is likely to play a prominent role. Some work on the border of mechanism design and cryptography has been recently conducted [73, 70]. An introduction to this topic can be found in e.g. [110, chapter 13] [26]. In some sense, the theories of cryptography and mechanism design are complementary. While a crypto system can guarantee that the participants behave legally, i.e. according to *some* legal input $t^i \in T^i$, the goal of mechanism design is to make sure that it will be for the agents' benefit to behave according to the *correct* one!

1.2 Overview of the results

The results in this thesis naturally divide into two main parts. The first, concerns the possibility (or impossibility) to develop strategy-proof mechanisms for problems that involve rational agents under various assumptions. The second, studies the computational aspect of such mechanisms.

This thesis shows, among the rest, that many problems that naturally stem from computer science applications can be satisfactorily handled. When the compensation and bonus mechanism developed in the first part is unified with the second chance mechanism developed in the second, one obtains a generic method for handling a wide variety of problems where verification assumptions are available. In this method the designer develops an *algorithm* for the corresponding optimization problem and automatically obtains a protocol with the following properties:

- The *only* rational strategy for the agents is to cooperate with the algorithm. (A precise definition of the solution concept appears in the second part of this work.)
- When the agents act rationally, the mechanism performs *at least as good* as the algorithm.
- If the algorithm is polynomial then so is the mechanism.
- Participation constraints are satisfied.
- The designer never pays more than her budget.
- The protocol is very simple from the agents' perspective.

1.2.1 Algorithmic Mechanism Design

This part studies the possibility or impossibility to solve problems that involve self-interested participants in dominant strategies. It investigates a representative task scheduling problem where the standard tools from mechanism design do not suffice. The results obtained in this part can be divided into three main parts:

Limitations of the classical model This part shows lower bounds on the approximation ratio that can be obtained by any deterministic strategy-proof mechanism for the representative problem.

Probabilistic mechanisms This part shows how to break the deterministic lower bounds using probability without weakening the game theoretic strength of the solution concept.

The power of verification assumptions This part studies strategy-proof mechanisms for the representative problem under additional verification assumptions which are natural to many problems. It introduces the *compensation and bonus* mechanism that can solve such problems *optimally*. Moreover the mechanism supports budget limitations. A method for developing a polynomial-time mechanisms is also suggested. This method basically works in cases where dynamic programming works.

1.2.2 Computationally Feasible VCG Mechanisms

This part studies the *computational* aspect of strategy-proof mechanisms. It is divided into two main parts:

Limitations of the existing methods This part shows that all known general methods for the construction of strategy-proof mechanisms fail to construct reasonable, computationally feasible mechanisms. In particular it shows gap phenomena for the important general problem of combinatorial auctions as well as for many cost minimization problems. A truthful mechanism constructed in the existing methods can either be optimal (and henceforth computationally infeasible) or produce unreasonable results.

The second-chance mechanism This part proposes a method to overcome this non-truthfulness. It introduces a *feasible* notion of truthfulness and shows a general way to construct mechanisms that are feasibly truthful under reasonable assumption on the agents. The mechanism has several desirable properties.

The solution concept suggested in this part is considerably different from previous work on bounded rationality and constitutes an important part of this thesis.

Chapter 2

Algorithmic Mechanism Design

This sections contains the paper “Algorithmic Mechanism Design” as it appeared in Games and Economic Behavior.

Algorithmic Mechanism Design

Noam Nisan

Institute of Computer Science, Hebrew University of Jerusalem, Givat Ram 91904, Israel and

School of Computer Science, IDC, Herzliya. Email: noam@cs.huji.ac.il

And

Amir Ronen

Institute of Computer Science, Hebrew University of Jerusalem, Givat Ram 91904, Israel.

Email: amiry@cs.huji.ac.il¹

Abstract

We consider algorithmic problems in a distributed setting where the participants cannot be assumed to follow the algorithm but rather their own self-interest. As such participants, termed agents, are capable of manipulating the algorithm, the algorithm designer should ensure in advance that the agents' interests are best served by behaving correctly.

Following notions from the field of mechanism design, we suggest a framework for studying such algorithms. In this model the algorithmic solution is adorned with payments to the participants and is termed a mechanism. The payments should be carefully chosen as to motivate all participants to act as the algorithm designer wishes. We apply the standard tools of mechanism design to algorithmic problems and in particular to the shortest path problem.

Our main technical contribution concerns the study of a representative task scheduling problem for which the standard mechanism design tools do not suffice. We present several theorems regarding this problem including an approximation mechanism, lower bounds and a randomized mechanism. We also suggest and motivate extensions to the basic model and prove improved upper bounds in the extended model.

Many open problems are suggested as well.

Journal of Economic Literature classification numbers: C60, C72, D61, D70, D80.

¹Both authors are supported by grants from the Israeli ministry of Science and the Israeli academy of sciences.

2.1 Introduction

2.1.1 Motivation

A large part of research in computer science is concerned with protocols and algorithms for interconnected collections of computers. The designer of such an algorithm or protocol always makes an implicit assumption that the participating computers will act as instructed – except, perhaps, for the faulty or malicious ones.

With the emergence of the Internet as *the* platform of computation, this assumption can no longer be taken for granted. Computers on the Internet belong to different persons or organizations and will likely do what is most beneficial to their owners. We cannot simply expect each computer on the Internet to faithfully follow the designed protocols or algorithms. It is more reasonable to expect that each computer will try to manipulate it for its owners' benefit. Such an algorithm or protocol must therefore be designed in advance for this kind of behavior! Let us sketch two example applications we have in mind:

Load balancing

The aggregate power of all computers on the Internet is huge. In a “dream world” this aggregate power will be optimally allocated online among all connected processors. One could imagine CPU-intensive jobs automatically migrating to CPU-servers, caching automatically done by computers with free disk space, etc. Access to data, communication lines and even physical attachments (such as printers) could all be allocated across the Internet. This is clearly a difficult optimization problem even within tightly linked systems, and is addressed, in various forms and with varying degrees of success, by all distributed operating systems. The same type of allocation over the Internet requires handling an additional problem: the resources belong to different parties who may not allow others to freely use them. The algorithms and protocols may, thus, need to provide some motivation for these owners to “play along”.

Routing

When one computer wishes to send information to another, the data usually gets routed through various intermediate routers. So far this has been done voluntarily, probably due to the low marginal cost of forwarding a packet. However, when communication of larger amounts of data becomes common (e.g. video), and bandwidth needs to be reserved under various quality of service (QoS) protocols, this altruistic behavior of the routers may no longer hold. If so, we will have to design protocols specifically taking the routers' self-interest into account.

2.1.2 This Work

In this paper we propose a formal model for studying algorithms that assume that the participants all act according to their own self-interest. We adopt a rationality-based approach, using notions from game theory and micro-economics, and in particular from the field of mechanism design. We assume that each participant has a well defined **utility function**² that represents its preference over the possible outputs of the algorithm, and we assume that participants act as to rationally optimize their utility. We term such rational and selfish participants **agents**³. The solutions we consider contain both an algorithmic ingredient (obtaining the intended results), and a payment ingredient that motivates the agents. We term such a solution a **mechanism**⁴.

Our contributions in this work are as follows:

1. We present a formal model for studying optimization problems. The model is based on the field of mechanism design⁵. A problem in this model has, in addition to the output specification, a description of the agents' utilities. The mechanism has, in addition to the algorithm producing the desired output, payments to the participating agents. An exposition of applying several classic notions from mechanism design in our model appears in [76].
2. We observe that the known techniques from mechanism design provide solutions for several basic optimization problems, and in particular for the shortest path problem, where each edge may belong to a different agent.
3. We study a basic problem, task scheduling, which requires new techniques and prove the following:
 - We design an n -approximation mechanism, where n is the number of agents.
 - We prove a lower bound of 2 to the approximation ratio that can be achieved by any mechanism. This bound is tight for the case of two agents, but leaves a gap for more agents. We conjecture that the upper bound is tight in general and prove it for two restricted classes of mechanisms.
 - We design a randomized mechanism that beats the deterministic lower bound.
4. We extend the basic model, formalizing a model where the mechanism has more information.

We call this model a mechanism with verification and argue that it is justified in certain

²This notion from micro-economics is often used in mechanism design.

³The term is taken from the distributed AI community which have introduced the usage of mechanism design in a computational setting. We use it, however, in a much more restricted and well-defined sense.

⁴This is the standard term used in mechanism design.

⁵We are not the first to use notions from mechanism design in a computational setting. See section 1.3.

applications.

5. We study the task scheduling problem in the extended model and obtain two main results:

- An optimal mechanism with verification for task scheduling (that requires exponential computation time).
- A polynomial time $(1 + \epsilon)$ -approximation mechanism with verification for a sub-case of the problem.

A preliminary version of this paper appeared at the thirty-first annual symposium on theory of computing ([77]).

2.1.3 Extant Work

There have been many works that tried to introduce economic or game-theoretic aspects into computational questions. (See e.g. [49], [20], [39], [86], [85] and a survey by [57]). Most of these were not aimed at the problem of the cooperation of selfish entities, and those that were ([68], [84], [45] and [94]) did not pursue our direction. Many subfields of game theory and economics are also related to our work, see, e.g. [60]. We list below the research work that is most relevant to our direction.

Mechanism Design

The field of mechanism design (also known as implementation theory) aims to study how privately known preferences of many people can be aggregated towards a “social choice”. The main motivation of this field is micro-economic, and the tools are game-theoretic. Emphasis is put on the implementation of various types of auctions. In the last few years this field has received much interest, especially due to its influence on large privatizations and spectrum allocations ([62]). An introduction to this field can be found in [60], [81] and an influential web site in [67].

Distributed AI

In the last decade or so, researchers in AI have studied cooperation and competition among “software agents”. The meaning of agents here is very broad, incorporating attributes of code-mobility, artificial-intelligence, user-customization and self-interest. A subfield of this general direction of research takes a game theoretic analysis of agents’ goals, and in particular uses notions from mechanism design ([91], [94], [18] and [103]). A related subfield of Distributed AI, sometimes termed market-based computation ([117], [20] and [116]), aims to leverage the notions of free markets in order to solve distributed problems. These subfields of DAI are related to our work.

Communication Networks

In recent years researchers in the field of network design adopted a game theoretic approach (See e.g. [45]). In particular mechanism design was applied to various problems including resource allocation ([51]), cost sharing and pricing ([99]).

Scheduling

The specific problem we address is the minimization of the make-span of independent tasks on unrelated parallel machines, which was extensively studied from an algorithmic point of view. It is known that solving the problem or even approximating it within a factor of $3/2 - \epsilon$ is *NP*-hard, but a polynomial-time 2-approximation exists ([54]). For a fixed number of processors, a fully polynomial approximation scheme was presented by [36]. A survey of scheduling algorithms can be found in [35].

2.2 The Model

In this section we formally present our model. We attempt, as much as possible, to use the standard notions from both mechanism design and algorithmics. We limit ourself to the discussion of a dominant strategy implementation in quasi-linear environments.

Subsection 2.1 describes what a mechanism design problem is. In subsection 2.2 we define what a good solution is: an implementation with dominant strategies. Subsection 2.3 defines a special class of good solutions: truthful implementations, and states the well-known fact that restricting ourselves to such solutions loses no generality. For familiarization with our basic model and notations we suggest viewing the shortest paths example given in section 3.2.

2.2.1 Mechanism Design Problem Description

Intuitively, a mechanism design problem has two components: the usual algorithmic output specification, and descriptions of what the participating agents want, formally given as utility functions over the set of possible outputs.

Definition 1 (Mechanism Design Problem) *A mechanism design problem is given by an output specification and by a set of agents' utilities. Specifically:*

1. *There are n agents, each agent i has available to it some **private** input $t^i \in T^i$ (termed its type). Everything else in this scenario is public knowledge.*
2. *The output specification maps to each type vector $t = t^1 \dots t^n$ a set of allowed outputs $o \in O$.*

3. Each agent i 's preferences are given by a real valued function: $v^i(t^i, o)$, called its valuation. This is a quantification of its value from the output o , when its type is t^i , in terms of some common currency. I.e. if the mechanism's output is o and in addition the mechanism hands this agent p^i units of this currency, then its utility will be $u^i = p^i + v^i(o, t^i)$ ⁶. This utility is what the agent aims to optimize.

In this paper we only consider the important special case of optimization problems. In these problems the output specification is to optimize a given objective function. We present the definition for minimization problems.

Definition 2 (Mechanism Design Optimization Problem) *This is a mechanism design problem where the output specification is given by a positive real valued objective function $g(o, t)$ and a set of feasible outputs F . In the exact case we require an output $o \in F$ that minimizes g , and in the approximate case we require any $o \in F$ that comes within a factor of c , i.e. such that for any other output $o' \in F$, $g(o, t) \leq c \cdot g(o', t)$.*

2.2.2 The Mechanism

Intuitively, a mechanism solves a given problem by assuring that the required output occurs, when agents choose their strategies as to *maximize their own selfish utilities*. A mechanism needs thus to ensure that the agents' utilities (which it can influence by handing out payments) are compatible with the algorithm.

Notation: We will denote $(a^1, \dots, a^{i-1}, a^{i+1}, \dots, a^n)$ by a^{-i} . (a^i, a^{-i}) will denote the tuple $(a^1, \dots, a^n)$.

Definition 3 (A Mechanism) *A mechanism $m = (o, p)$ is composed of two elements: An output function $o()$, and an n -tuple of payments $p^1() \dots p^n()$. Specifically:*

1. *The mechanism defines for each agent i a family of strategies A^i . The agent can chose to perform any $a^i \in A^i$.*
2. *The first thing a mechanism must provide is an output function $o = o(a^1 \dots a^n)$.*
3. *The second thing a mechanism provides is a payment $p^i = p^i(a^1 \dots a^n)$ to each of the agents.*
4. *We say that a mechanism is an implementation with dominant strategies ⁷ (or in short just an implementation) if*

⁶This is termed "quasi-linear utility". In this paper we limit ourselves to this type of utilities.

⁷Several "solution concepts" are discussed in the mechanism design literature. In this paper we discuss only dominant strategy implementation.

- For each agent i and each t^i there exists a strategy $a^i \in A^i$, termed *dominant*, such that for all possible strategies of the other agents a^{-i} , a^i maximizes agent i 's utility. I.e. for every $a'^i \in A^i$, if we define $o = o(a^i, a^{-i})$, $o' = o(a'^i, a^{-i})$, $p^i = p^i(a^i, a^{-i})$, $p'^i = p^i(a'^i, a^{-i})$, then $v^i(t^i, o) + p^i \geq v^i(t^i, o') + p'^i$
- For each tuple of dominant strategies $a = (a^1 \dots a^n)$ the output $o(a)$ satisfies the specification.

We say that a mechanism is poly-time computable if the output and payment functions are computable in polynomial time. In this paper we purposefully do not consider the details of how the mechanism is computed in a distributed system. We view this topic as an important direction for further research.

2.2.3 The Revelation Principle

The simplest types of mechanisms are those in which the agents' strategies are to simply report their types.

Definition 4 (Truthful Implementation) *We say that a mechanism is truthful if*

1. For all i , and all t^i , $A^i = T^i$, i.e. the agents' strategies are to report their type. (This is called a *direct revelation mechanism*.)
2. Truth-telling is a dominant strategy, i.e. $a^i = t^i$ satisfies the definition of a dominant strategy above.

Definition 5 *We say that a mechanism is strongly truthful if truth-telling is the only dominant strategy.*

A simple observation, known as the *revelation principle*, states that without loss of generality one can concentrate on truthful implementations.

Proposition 2.2.1 ([60]) *If there exists a mechanism that implements a given problem with dominant strategies then there exists a truthful implementation as well.*

Proof: (sketch) We let the truthful implementation simulate the agents' strategies. I.e. given a mechanism $(o, p^1, \dots, p^n)$, with dominant strategies $a^i(t^i)$, we can define a new one by $o^*(t^1 \dots t^n) = o(a^1(t^1) \dots a^n(t^n))$ and $(p^*)^i(t^1 \dots t^n) = p^i(a^1(t^1) \dots a^n(t^n))$. $\square$

2.3 Vickrey-Groves-Clarke Mechanisms

Arguably the most important positive result in mechanism design is what is usually called the generalized Vickrey-Groves-Clarke (VCG) mechanism ([112]; [31]; [8]). We first describe these mechanisms in our notation and then demonstrate their usage in an algorithmic setting, that of shortest paths.

2.3.1 Utilitarian Functions

The VCG mechanism applies to mechanism design maximization problems where the objective function is simply the sum of all agents' valuations. The set of possible outputs is assumed to be finite.

Definition 6 *A maximization mechanism design problem is called utilitarian if its objective function satisfies $g(o, t) = \sum_i v^i(t^i, o)$.*

Definition 7 *We say that a direct revelation mechanism $m = (o(t), p(t))$ belongs to the VCG family if*

1. $o(t) \in \arg \max_o (\sum_{i=1}^n v^i(t^i, o))$.
2. $p^i(t) = \sum_{j \neq i} v^j(t^j, o(t)) + h^i(t^{-i})$ where $h^i(\cdot)$ is an arbitrary function of t^{-i} .

Theorem 2.3.1 ([31]) *A VCG mechanism is truthful.*

□

Thus, a VCG mechanism essentially provides a solution for any utilitarian problem (except for the possible problem that there might be dominant strategies other than truth-telling). It is known that (under mild assumptions) VCG mechanisms are the only truthful implementations for utilitarian problems ([30]).

Similarly, a weighted version can be implemented as well.

Definition 8 *A maximization mechanism design problem is called weighted utilitarian if there exist real numbers $\beta^1, \dots, \beta^n > 0$ such that the problem's objective function satisfies $g(o, t) = \sum_i \beta^i \cdot v^i(t^i, o)$.*

Definition 9 *We say that a direct revelation mechanism $m = (o(t), p(t))$ belongs to the weighted VCG family if*

1. $o(t) \in \arg \max_o (g(o, t))$.

2. $p^i(t) = \frac{1}{\beta^i} \cdot \sum_{j \neq i} \beta^j \cdot v^j(t^j, o(t)) + h^i(t^{-i})$ where $h^i()$ is an arbitrary function of t^{-i} .

Theorem 2.3.2 ([89]) *A weighted VCG mechanism is truthful.*

Proof: Let $d^1, \dots, d^n$ denote the declarations of the agents and $t^1, \dots, t^n$ denote their real types.

Suppose that truth telling is not a dominant strategy, then there exists d, i, t, d'^i such that

$$v^i(t^i, o(d^{-i}, t^i)) + p^i(t^i, o(d^{-i}, t^i)) + h^i(d^{-i}) < v^i(t^i, o(d^{-i}, d'^i)) + p^i(t^i, o(d^{-i}, d'^i)) + h^i(d^{-i})$$

thus

$$v^i(t^i, o(d^{-i}, t^i)) + \frac{1}{\beta^i} \cdot \sum_{j \neq i} \beta^j \cdot v^j(t^j, o(d^{-i}, t^i)) < v^i(t^i, o(d^{-i}, d'^i)) + \frac{1}{\beta^i} \cdot \sum_{j \neq i} \beta^j \cdot v^j(t^j, o(d^{-i}, d'^i))$$

Multiplying both sides by β^i we obtain

$$\sum_{j=1}^n \beta^j \cdot v^j(t^j, o(d^{-i}, t^i)) < \sum_{j=1}^n \beta^j \cdot v^j(t^j, o(d^{-i}, d'^i))$$

In contradiction with the definition of $o()$. □

Comment: An output function of a VCG mechanism is required to maximize the objective function. In many cases (e.g. combinatorial auctions ([33])), this makes the mechanism computationally intractable. Replacing the optimal algorithm with a non-optimal approximation usually leads to untruthful mechanisms – see for example section 2.5.6.

2.3.2 Example: Shortest Paths

Many algorithmic mechanism design problems can be solved using the VCG mechanism. Let us give a natural example.

Problem definition: We have a communication network modeled by a directed graph G , and two special nodes in it x and y . Each edge e of the graph is an agent. Each agent e has private information (its type) $t^e \geq 0$ which is the agent's cost for sending a single message along this edge. The goal is to find the cheapest path from x to y (as to send a single message from x to y). I.e the set of feasible outputs are all paths from x to y , and the objective function is the path's total cost. Agent e 's valuation is 0 if its edge is not part of the chosen path, and $-t^e$ if it is. We will assume for simplicity that the graph is bi-connected.

A Truthful Implementation:

The following mechanism ensures that the dominant strategy for each agent is to report its true type t^e to the mechanism. When all agents honestly report their costs, the cheapest path is chosen:

The output is obtained by a simple shortest path calculation. The payment p^e given to agent e is 0 if e is not in the shortest path and $p^e = d_{G|e=\infty} - d_{G|e=0}$ if it is. Here $d_{G|e=\infty}$ is the length of the shortest path which does not contain e (according to the inputs reported), and $d_{G|e=0}$ is the length of the shortest path when the cost of e is assumed to be zero (again according to the reported types).

Notice that the shortest path is indeed a minimization of the total cost. Also notice that the given mechanism is a VCG mechanism: $d_{G|e=\infty}$ corresponds to $h^i(t^{-i})$ and $d_{G|e=0}$ to $\sum_{j \neq i} v^j(t^j, o(t))$.

Many other graph problems, where agents are edges, and their valuations proportional to the edges' weights, can be implemented by a VCG mechanism. In particular minimum spanning tree and max-weight matching seem natural problems in this setting. A similar solution applies to the more general case where each agent holds some subset of the edges.

Open Problem: How fast can the payment functions be computed? Can it be done faster than computing n versions of the original problem? For the shortest paths problem we get the following equivalent problem: given a directed graph G with non-negative weights, and two vertices in it x, y . Find, for each edge e in the graph, the shortest path from x to y that does not use e . Using Disjktra's algorithm for each edge on the shortest path gives an $O(nm \log n)$ algorithm. Is anything better possible? Maybe $O(m \log n)$? For the similar problem with minimum spanning tree, it has been pointed out to us by Valerie King that the known fully dynamic algorithms (or alternatively the known sensitivity-analysis algorithms) for MST provide a nearly linear time solution.

2.4 Task Scheduling

In this section we analyze the task allocation problem. Subsection 4.1 formally presents the problem, subsection 4.2 gives a (weak) upper bound, subsection 4.3 provides our lower bounds, and finally in subsection 4.4 we exhibit a randomized solution that beats the lower bound.

2.4.1 The Problem

Definition 10 (Task Allocation Problem) *There are k tasks that need to be allocated to n agents. Each agent i 's type is, for each task j , the minimum amount of time t_j^i it is capable of performing this task in. The goal is to minimize the completion time of the last assignment (the make-span). The valuation of an agent i is the negation of the total time it has spent on the tasks allocated to it.*

More formally:

- The feasible outputs of the mechanism are all partitions $x = x^1 \dots x^n$ of the tasks to the agents, where x^i is the set of tasks allocated to agent i .
- The objective function is $g(x, t) = \max_i \sum_{j \in x^i} t_j^i$.
- Agent i 's valuation is $v^i(x, t^i) = - \sum_{j \in x^i} t_j^i$.

We will consider both the exact and approximate versions.

Notation: We denote a direct revelation mechanism for the task scheduling problem by $m = (x, p)$, where $x = x(t)$ is the allocation algorithm and $p = p(t)$ the payment. (These are functions of the declared types.)

2.4.2 An Upper Bound

A simple, but not very good, approximation for the task scheduling problem is to minimize the total work done. It turns out that this approximation can be used as a basis for an approximation mechanism⁸.

Definition 11 (MinWork Mechanism)

- **allocation:** each task is allocated to the agent who is capable of doing it in a minimal amount of time (tasks with equal time declarations are allocated arbitrarily).
- **payment:** the payment for each agent i is defined as $p^i(t) = \sum_{j \in x^i(t)} \min_{i' \neq i} (t_j^{i'})$ (i.e. for each task allocated to it, the agent is given payment equal to the time of the second best agent for this task).

Theorem 2.4.1 *MinWork is a strongly truthful n -approximation mechanism for the task scheduling problem.*

Proof: We prove that the MinWork mechanism is strongly truthful and that it is an n -approximation.

Claim 2.4.2 *MinWork is strongly truthful .*

Proof: We will first show that MinWork belongs to the VCG family, and therefore, by theorem 3.2.1 it is truthful. The output is an allocation that maximizes the utilitarian function $\sum_{i=1}^n v^i(t^i, x)$; Let h^{-i} be $\sum_{j=1}^k \min_{i' \neq i} t_j^{i'}$, then $\sum_{i' \neq i} v^{i'}(t^{i'}, x) + h^{-i}$ is exactly the mechanism's payment function.

⁸The mechanism can be viewed as auctioning each task separately in a Vickrey auction ([112]).

We now show that truth-telling is the only dominant strategy. We will show it for the case of a single task. The argument for $k > 1$ is similar. We note that a similar proof can be found in [112] for the analysis of the famous Vickrey auction. Let d denote the agents' declarations and t their real types. Consider the case where $d^i \neq t^i$ ($i = 1, 2$). If $d^i > t^i$, then for d^{3-i} such that $d^i > d^{3-i} > t^i$, the utility for agent i is $t^i - d^i < 0$, instead of 0 in the case of truth-telling. A similar argument holds for the case of $d^i < t^i$. $\square$

Claim 2.4.3 *MinWork is an n -approximation for the task scheduling problem.*

Proof: Let $opt(t)$ denote an optimal allocation. The proof follows immediately from the fact that $g(x(t), t) \leq \sum_{j=1}^k \min_i t_j^i$ and $g(opt(t), t) \geq \frac{1}{n} \cdot \sum_{j=1}^k \min_i t_j^i$. $\square$

The theorem is an immediate outcome of claims 2.4.2 and 2.4.3. $\square$

2.4.3 Lower Bounds

Due to the revelation principle (proposition 2.2.1) it suffices to prove the lower bound for truthful implementations. Thus for the rest of this section, $m = (x, p)$ is always assumed to be a truthful mechanism for the task scheduling problem.

Basic Properties of Truthful Implementations

We now formulate, in our settings, two basic observations from mechanism design. (Similar arguments can be found in [60].)

Proposition 2.4.4 (Independence) *Let t_1 and t_2 be type vectors, and i be an agent. If $t_1^{-i} = t_2^{-i}$ and $x^i(t_1) = x^i(t_2)$, then $p^i(t_1) = p^i(t_2)$.*

Proof: Without loss of generality assume that $p^i(t_1) < p^i(t_2)$. Then, if i 's type is t_1^i , it is better off cheating, declaring t_2^i . A contradiction to the truthfulness. $\square$

This proposition states that the payment offered to an agent does not depend on its type declaration (as long as the other agents' types and the allocation are fixed). The payment can thus be represented using the following well defined function.

Definition 12 *Let t be a type vector, i an agent. For a set X of tasks, we define the price offered for X to agent i as:*

$$p^i(X, t^{-i}) = \begin{cases} p^i(t'^i, t^{-i}) & \text{if there exists } t'^i \text{ s.t. } x^i(t'^i, t^{-i}) = X \\ 0 & \text{otherwise} \end{cases}$$

Usually it will be more convenient to describe a mechanism by its price rather than by its payment function. Note that any function of the form $h^i(t^{-i})$ can be added to the payment of each agent i without changing its considerations. We therefore assume w.l.o.g. that the payment given to an agent is zero if no tasks are assigned to it.

Notation: Let i be an agent of type t^i , and let X be a set of tasks. We denote the time needed for i to perform all tasks of X , as $t^i(X) = \sum_{j \in X} t_j^i$.

Proposition 2.4.5 (Maximization) *For each type vector t and agent i ,*

$$x^i(t) \in \arg \max_{X \subseteq \{1, \dots, k\}} (p^i(X, t^{-i}) - t^i(X))$$

Proof:(sketch) Since $p^i(x^i, t^{-i}) - t^i(x^i)$ is agent i 's utility, the above statement simply states that the mechanism has to maximize the agent's benefit. Otherwise the agent will do so itself, i.e. cheat as to get the maximum benefit! $\square$

We can now prove the main theorem of this subsection.

Basic Lower Bound

Theorem 2.4.6 *There does not exist a mechanism that implements a c -approximation for the task scheduling problem for any $c < 2$.*

Proof: We start with a lemma.

Notation: Let i be an agent, t a type vector, and A and B two disjoint sets of tasks. We define the *price difference* $\Delta^i(A, B)$ to be $p^i(A \cup B, t^{-i}) - p^i(A, t^{-i})$ (suppressing the type t).

Lemma 2.4.7 *Let t be a type vector and let $X = x^i(t)$. For each set $D \neq X$ of tasks the following inequalities hold:*

1. *If $D \subset X$ then $\Delta^i(D, X - D) \geq t^i(X - D)$.*
2. *If $D \supset X$ then $\Delta^i(X, D - X) \leq t^i(D - X)$.*
3. *otherwise, let $L = D \cap X$, then $\Delta^i(L, X - L) - t^i(X - L) \geq \Delta^i(L, D - L) - t^i(D - L)$*

Moreover, if a set Y of tasks satisfies these inequalities sharply for all D 's, then $Y = X = x^i(t)$.

Proof: The fact that the above inequalities hold for $x^i(t)$ is an immediate consequence of proposition 2.4.5 (maximization) and the definition of the utility as $u^i = p^i(x^i(t), t^{-i}) - t^i(x^i(t))$. When the inequalities are strict, X is clearly the unique set of tasks that maximizes i 's utility. $\square$

We prove the theorem for the case of two agents. For $n > 2$ we can reduce to this case by having the other agents be much slower than agents 1 and 2.

Notation: Let t be a type vector, i an agent and X a set of tasks. Let $\alpha > 0$ be a real number. We denote by $\hat{t} = t(X \xrightarrow{i} \alpha)$ the type obtained by

$$\hat{t}_j^{i'} = \begin{cases} \alpha & \text{if } i' = i \text{ and } j \in X \\ t_j^{i'} & \text{otherwise} \end{cases}$$

In the same manner we define $\hat{t} = t(X^1 \xrightarrow{i_1} \alpha_1, X^2 \xrightarrow{i_2} \alpha_2, \dots)$ to be the result of a sequence of the above transformations.

Let $k \geq 3$, and t be the type vector defined by $t_j^i = 1$ for each agent i and task j . Without loss of generality we assume that $|x^1(t)| \leq |x^2(t)|$. Let $x = x^1(t)$ and let $\bar{x}$ denote its complement ($x^2(t)$).

Claim 2.4.8 *Let $0 < \epsilon < 1$, $\hat{t} = t(x \xrightarrow{1} \epsilon, \bar{x} \xrightarrow{1} 1 + \epsilon)$. Then $x(\hat{t}) = x(t)$.*

Proof: Since $n = 2$, it is enough to show that $x^1(\hat{t}) = x^1(t)$. As the type of agent 2 has not changed, the prices offered to agent 1 remain the same. For type t , $x^1(t)$ fulfills the inequalities of lemma 2.4.7. Thus, by inspection, they are strict when the type becomes $\hat{t}$, and therefore the allocation remains the same. $\square$

Assuming $|x^2(t)|$ is even, the lower bound follows since $g(x(\hat{t}), \hat{t}) = |x^2(\hat{t})| = |x^2(t)|$, but $g(\text{opt}(\hat{t}), \hat{t}) \leq \frac{1}{2} \cdot |x^2| + k \cdot \epsilon$ (for the allocation that gives agent 1, in addition to the original $x^1(t)$, half of agent 2's original allocation).

For the case of odd $|x^2(t)|$ it must be that $|x^2(t)| \geq 3$. We choose an arbitrary task $j \in x^2(t)$ and consider the type $\hat{t}(\{j\} \xrightarrow{2} \epsilon)$, which still yields the same allocation. $\square$

This lower bound is tight for the case of two agents. We conjecture that, in general, the upper bound is tight:

Conjecture 2.4.9 *There does not exist a mechanism that implements a c -approximation for the task scheduling problem with n agents for any $c < n$.*

Although we have not been able to prove this conjecture, we can show that it is correct for two natural special cases presented in the next subsection.

Tight Bounds for Special Cases

Definition 13 *A mechanism is called additive if for each agent i , type vector t and set X of tasks, $p^i(X, t^{-i}) = \sum_{j \in X} p^i(\{j\}, t^{-i})$.*

Theorem 2.4.10 *There does not exist any additive mechanism that solves the c -approximation problem for any $c < n$.*

Proof: Let $k \geq n^2$ and let $t_j^i = 1$ for each agent i and task j . Without loss of generality we assume that $|x^1(t)| \geq n$. Let $x = x^1(t)$ and let $\bar{x}$ denote its complement.

Claim 2.4.11 *Fix $0 < \epsilon < 1$ and let $\hat{t} = t(x^1 \xrightarrow{1} 1 - \epsilon, \bar{x} \xrightarrow{1} \epsilon)$. Then $x^1(\hat{t}) \supseteq x^1(t)$.*

Proof: Since t^2 has not changed, the prices offered to agent 1 remain the same. Clearly the price offered to agent 1 for x is strictly greater than the time $\hat{t}^1(x)$ required for it to perform x . Since the payment is additive, the set $x^1(\hat{t})$ which maximizes 1's utility must contain all the tasks in x . $\square$

It follows that $g(x(\hat{t}), \hat{t}) \geq |x^1| \geq n$. Like in theorem 2.4.6, we can assume w.l.o.g. that $|x^1| = n$. The lower bound is then obtained since an optimal allocation would split these tasks among the n agents. $\square$

Definition 14 *We say that a mechanism is local if for each agent i , type vector t and set X of tasks, the price $p^i(X, t^{-i})$ depends only on the agents' values on the tasks in X (i.e. $\{t_j^{l \neq i} | j \in X\}$).*

Theorem 2.4.12 *There does not exist a local mechanism that solves the c -approximation problem for any $c < n$.*

Proof: We start with a simple lemma that will enable us to turn the inequalities of lemma 2.4.7 into sharp ones.

Lemma 2.4.13 *For each type vector t and $\epsilon > 0$, there exists a type vector t' such that $\|t - t'\| < \epsilon$ and where the sets that maximize the agents' utility are unique for all agents.*

Proof:(sketch) The lemma is proved using a simple measure-theoretic argument. Let i be an agent, $A \neq B$ two sets of tasks. Because of the independence property (proposition 2.4.4), the following set has a zero measure on the type-space of agent i :

$$E^i(A, B, t^{-i}) = \{t^i | p^i(A, t^{-i}) - t^i(A) = p^i(B, t^{-i}) - t^i(B)\}$$

From this we obtain that for almost every type vector t' , the inequalities in lemma 2.4.7 (for all agents) are strict. $\square$

Let $k \geq n^2$ and let $t_j^i = 1$ for each agent i and task j . By lemma 2.4.13, we assume w.l.o.g. that $x^i(t)$ uniquely maximizes i 's utility for all agents i . Without loss of generality we assume that $|x^1(t)| \geq n$.

Claim 2.4.14 *Let $x = x^2(t)$ and $\hat{t} = t(x \xrightarrow{2} \epsilon)$ for some $0 < \epsilon < 1$. Then $x(\hat{t}) = x(t)$.*

Proof: Clearly $x^2(\hat{t}) = x^2(t)$. Consider another agent $i \neq 2$. The mechanism must allocate to agent i a set $x^i(\hat{t})$ that maximizes i 's utility among all the sets X which are *disjoint* from $x^2(t)$. But since the mechanism is local, these prices have not changed from t to $\hat{t}$. Therefore $x^i(t)$ remains the unique set that maximizes i 's utility. $\square$

By the same argument the allocation for the type $\hat{t} = t(x^2(t) \xrightarrow{2} \epsilon, \dots, x^n(t) \xrightarrow{n} \epsilon)$ remains $x(t)$.

Like in theorem 2.4.6 we can assume that $|x^1(t)| = n$ and thus the lower bound is obtained since an optimal allocation will split these tasks among the n agents. $\square$

2.4.4 The Power of Randomization

In section 2.4.3 we showed that no mechanism can achieve a better than 2-approximation for the task scheduling problem. Here we show that *randomized* mechanisms can do better. The model of randomization that we use does not weaken the demands of dominant strategies at all: Although the agents choose their strategies without knowing the results of the random coin tosses, we require the strategy to be dominant for *all* possible tosses.

Definition 15 (A Randomized Mechanism) *A randomized mechanism is a probability distribution over a family $\{m_r | r \in I\}$ of mechanisms, all sharing the same sets of strategies and possible outputs.*

The outcome of such a mechanism is a probability distribution over outputs and payments; the problem specification must specify what output distributions are required. For the case of optimization problems, the objective function on such a distribution is taken to be the expectation, i.e. $g(a, t) = E_{r \in I}(g(o_{m_r}(a), t))$.

Definition 16 (Universally Dominant Strategy) *A strategy a^i is called universally dominant (in short, dominant) for agent i if it is a dominant strategy for every mechanism in the support of the randomized mechanism. A randomized mechanism is called universally truthful (in short, truthful) if truth-telling is a dominant strategy, and strongly truthful if it is the only one.*

We will design a strongly truthful randomized mechanism that achieves better performance than the deterministic lower bound. The randomized mechanism will be a distribution over *biased min work* mechanisms defined in figure 1.

Lemma 2.4.15 *For all parameter values, the biased min work mechanism is strongly truthful.*

Parameters: A real number $\beta \geq 1$ and a bit vector $s \in \{1, 2\}^k$. Input: The reported type vectors $t = (t^1, t^2)$. Output: An allocation $x = (x^1, x^2)$, and a payment $p = (p^1, p^2)$. Mechanism: $x^1 \leftarrow \emptyset; x^2 \leftarrow \emptyset; p^1 \leftarrow 0; p^2 \leftarrow 0$. For each task $j = 1 \dots k$ do Let $i = s_j$ and $i' = 3 - i$ If $t_j^i \leq \beta \cdot t_j^{i'}$ Then $x^i \leftarrow x^i \cup \{j\}; p^i \leftarrow p^i + \beta \cdot t_j^{i'}$ Else $x^{i'} \leftarrow x^{i'} \cup \{j\}; p^{i'} \leftarrow p^{i'} + \beta^{-1} \cdot t_j^i$
--

Figure 2.1: the biased min work mechanism (for two agents)

Proof: Since the overall utility of each agent can be described as the sum of the utilities aggregated in each step, it is enough to consider the case of $k = 1$. In this case the mechanism is equivalent to a weighted VCG (definition 9) with weights $\{1, \beta\}$ or $\{\beta, 1\}$ (depending on s_j). $\square$

Definition 17 (The Randomly Biased Min Work Mechanism) *The randomly biased min work mechanism is the distribution on biased min work mechanisms given by $\beta = 4/3$, and a uniform distribution of $s \in \{1, 2\}^k$.*

Theorem 2.4.16 *The randomly biased min work mechanism is a (polynomial time computable) strongly truthful implementation of a 7/4-approximation for task scheduling with two agents.*

The proof of the theorem is immediate from the following two lemmas.

Lemma 2.4.17 *The randomly biased min work mechanism is strongly truthful .*

This is immediate from lemma 2.4.15.

Lemma 2.4.18 *The allocation obtained by the randomly biased min work mechanism is a 7/4-approximation for the task scheduling problem.*

Comment: Our original analysis yielded a bound of 1.823. Daniel Lehmann ([52]) provided us with a tighter case analysis, improving the bound to 7/4. With Daniel's permission we include his refined analysis in our proof.

Proof: Let $opt(t)$ denote an optimal allocation algorithm. Let t_{opt} denote its make-span, and let t_{bmw} denote the (expected) make-span of the randomly biased min work mechanism.

We call a task j a k -task if one of the agents is considerably more efficient than the other on it (i.e. $t_j^1/t_j^2 > \beta$ or $t_j^2/t_j^1 > \beta$); otherwise we call it an l -task. Note that the mechanism allocates each k -task to the agent which is *efficient* on it, and *randomly* allocates the l -tasks.

Claim 2.4.19 *It is enough to consider the following case:*

1. *For each k -task, the efficiency discrepancy between the agents is arbitrarily close to β (therefore we shall assume that it equals β).*
2. *If opt allocates an l -task j to agent i , then $t_j^{3-i}/t_j^i = \beta$.*
3. *Under opt both agents have the same finishing time.*
4. *One of the agents is more efficient than the other on all k -tasks (w.l.o.g. let it be agent 1).*
5. *There are at most four tasks, where at most one k -task and at most one l -task is allocated by opt to each agent.*

Proof:

1. Since the mechanism always allocates each k -task j to the agent i which is more efficient on it, reducing t_j^{3-i} down to $\beta \cdot t_j^i$ can only help opt and leaves t_{bmw} unchanged.
2. If opt allocates an l -task j to agent i , then increasing t_j^{3-i} will not affect t_{opt} but will increase t_{bmw} .
3. Otherwise, w.l.o.g. assume that agent 1 finishes δ -time before agent 2. Adding an l -task j such that $t_j^1 = \delta$ and $t_j^2 = \beta \cdot \delta$ does not change t_{opt} but increases t_{bmw} .
4. Assume that there are two k -tasks a and b such that $t_a^2/t_a^1 = t_b^1/t_b^2 = \beta$. W.l.o.g. $t_a^1 \geq t_b^2$. If a is replaced by two k -tasks a' and a'' such that $t_a^i = t_{a'}^i + t_{a''}^i$ then t_{bmw} remains the same while t_{opt} can only decrease. In particular we can choose $t_{a'}^1 = t_b^2$.

The mechanism allocates both tasks a' and b to the agent which is efficient on them. If opt is doing the same then clearly removing both a' and b can just increase the ratio t_{bmw}/t_{opt} . Obviously, opt cannot allocate a' to agent 2 and b to agent 1. Therefore it is enough to consider the case where opt allocates both tasks to one of the agents. One can verify that in this case replacing both tasks with equivalent l -tasks (i.e. l -tasks with the same computational times as the original ones) does not affect t_{opt} but will increase t_{bmw} by at least $\frac{\beta-1}{2} \cdot t_{a'}^1$.

5. Let a and b be two k-tasks that opt allocates to the same agent i . Recall that $t_a^{3-i}/t_a^i = t_b^{3-i}/t_b^i = \beta$. Clearly, replacing both tasks with a single task c such that $t_c^i = t_a^i + t_b^i$, does not affect opt nor the mechanism. We now consider the case where a and b are both l-tasks. Again, t_{opt} does not change as a consequence of such a replacement. We will show that t_{bmw} can only increase. Let Y be an allocation of all the tasks except a and b ; let $t_{Y,a,b}$ denote the expected make-span when all other tasks are allocated according to Y and a and b are randomly allocated; let $t_{Y,c}$ denote the expected make-span when a and b are replaced by c which is allocated randomly. Clearly, it is enough to show that $t_{Y,a,b} \leq t_{Y,c}$.

Let T^1 and T^2 denote the finishing times of both agents respectively when the allocation is Y . If one of the agents i finishes after the other regardless of how a and b are allocated, then clearly $t_{Y,a,b} = T^i + \frac{t_a^i + t_b^i}{2} = t_{Y,c}$. Otherwise, if agent i finishes last iff both a and b are allocated to it, then $t_{Y,a,b} = \frac{T^i + t_a^i + t_b^i}{4} + \frac{T^{3-i} + t_a^{3-i}}{4} + \frac{T^{3-i} + t_b^{3-i}}{4} + \frac{T^{3-i} + t_a^{3-i} + t_b^{3-i}}{4}$. Since $T^{3-i} \leq T^i + t_a^i + t_b^i$, we obtain that $t_{Y,a,b} \leq \frac{T^i + t_a^i + t_b^i}{2} + \frac{T^{3-i} + t_a^{3-i} + t_b^{3-i}}{2} = t_{Y,c}$. Finally w.l.o.g. assume that $t_a^i \geq t_b^i$ (for $i = 1, 2$) and consider the case where the agent to which a is allocated finishes last. In this case $t_{Y,a,b} = \frac{T^1 + t_a^1 + t_b^1}{4} + \frac{T^1 + t_a^1}{4} + \frac{T^2 + t_a^2 + t_b^2}{4} + \frac{T^2 + t_a^2}{4} \leq \frac{T^1 + t_a^1 + t_b^1}{2} + \frac{T^2 + t_a^2 + t_b^2}{2} = t_{Y,c}$.

□

Following the above claim we prove the lemma for the case of four tasks k_1, k_2, l_1, l_2 , such that k_i and l_i denote the k-task and l-task which are allocated to agent i by opt (cases in which there are less than four tasks can be represented by zero times). The reduced case is described in figure 2.2.

	t_j^1	t_j^2	opt-alloc	bmw-alloc
k_1	a	$\beta \cdot a$	1	1
k_2	b	$\beta \cdot b$	2	1
l_1	c	$\beta \cdot c$	1	rnd
l_2	$\beta \cdot d$	d	2	rnd

Figure 2.2: the reduced case

Since both agents have the same finishing time in opt , $t_{opt} = a + c = \beta \cdot b + d$. We show that $\frac{t_{bmw}}{t_{opt}} \leq 7/4$ by considering three separate sub-cases.

Case 1: $a + b + \beta \cdot d \leq \beta \cdot c$.

Considering all four possible allocations of the mechanism we obtain that $t_{bmw} = 1/4 \cdot ((a + b + c + \beta \cdot d) + (a + b + c) + (\beta \cdot c) + (\beta \cdot c + d))$. Substituting $\beta = 4/3$ we get $t_{bmw} = 1/2 \cdot a + 1/2 \cdot b + 7/6 \cdot c + 7/12 \cdot d$ and one can verify that $t_{bmw} \leq 7/4 \cdot (a + c) = 7/4 \cdot t_{opt}$.

Case 2: Otherwise, $a + b + \beta \cdot d \geq \beta \cdot c$. Consider the case where $a + b \leq \beta \cdot c + d$.

In this case $t_{bmw} = 1/4 \cdot ((a + b + c + \beta \cdot d) + (a + b + c) + (a + b + \beta \cdot d) + (\beta \cdot c + d))$. Substituting β we get $t_{bmw} = 3/4 \cdot a + 3/4 \cdot b + 5/6 \cdot c + 11/12 \cdot d$ and it is not difficult to verify that $t_{bmw} \leq 7/4 \cdot (a + c) = 7/4 \cdot t_{opt}$.

Case 3: Otherwise, $a + b + \beta \cdot d \geq \beta \cdot c$ and $a + b \geq \beta \cdot c + d$. In this case $t_{bmw} = 1/4 \cdot ((a + b + c + \beta \cdot d) + (a + b + c) + (a + b + \beta \cdot d) + (a + b))$. Substituting β we get $t_{bmw} = a + b + c/2 + 2/3 \cdot d$ and again it can be verified that $t_{bmw} \leq 7/4 \cdot (4/3 \cdot b + d) = 7/4 \cdot t_{opt}$. $\square$

This completes the proof of the theorem. $\square$

2.5 Mechanisms with Verification

The basic mechanism design model assumes that each agent can follow any of its strategies, independently of its type. Thus the mechanism cannot use any “real-world” information about the agents. This is the norm in mechanism design and it models well the negotiation stage in which agents do nothing but communicate. In many settings in distributed computation though, one could take advantage of the fact that computers actually act (execute a task, route a message, etc.) to gain extra information about the agents’ types and actions.

A simple type of modification to the model suggests itself: a problem definition may limit the set of strategies A^i available to each agent as a function of its type t^i . Many variants are possible, with different types of information available at different stages of the mechanism. In this paper we concentrate on what we feel is a very natural model. We distinguish between two stages of the mechanism: a declaration phase in which agents “talk” and which results in a decision (e.g. allocation), and then an execution phase where the agents actually execute the agreed output. The payments need only to be given *after* the execution. Intuitively we view the execution part as allowing the mechanism to verify in some sense the agents’ declarations, and “punish” them for lying.

For the task scheduling problem we assume that by the end of the execution the mechanism **knows** the exact execution time of each task. A reformulation of the problems is introduced in section 2.5.2. We then (in sections 2.5.3 and 2.5.4) present a family of optimal mechanisms⁹ for this problem. In section 2.5.5 we show that versions of our mechanism can operate under a limited budget and can guarantee that the profit for a truthful agent is always non-negative. Since these mechanisms require optimal scheduling algorithms they are computationally intractable. In section 2.5.6 we discuss polynomial-time mechanisms. We define a sub-case of the scheduling problem for which we present a polynomial-time approximation schema. The existence of a (better than n)

⁹Although these mechanisms are presented for the scheduling problem, they can be generalized for many situations.

polynomial time approximation mechanism for the general problem is left open.

2.5.1 Mechanisms with Verification

Definition 18 (Mechanism with Verification)

- An agent's strategy a^i is composed of two separate parts: a declaration d^i and an execution e^i .
- Each declaration d^i is chosen by the agent, based on its type t^i , in an unrestricted manner.
- The decision k of the mechanism must be a function of just the declarations $d^1, \dots, d^n$.
- The agent's execution e^i may depend on t^i as well as on k . The problem specification specifies, for each t^i , the possible $e^i()$'s an agent of type t^i may choose.
- The output of the mechanism is the result of the decision k and the agents' executions $e^1(k), \dots, e^n(k)$. The output function $o(k, e)$ is a part of the problem specification.
- The output o , determines both the objective function $g(o, t)$ and the agents' valuations $v^i(t^i, o)$.
- The payment p^i that the mechanism provides depends on both, the declarations $d^1, \dots, d^n$ and the executions $e^1(k) \dots e^n(k)$.

Definition 19 A mechanism with verification is called truthful if

1. The agents' declarations are simply to report their types.
2. For each agent i of type t^i , there is a dominant strategy of the form $a^i = (t^i, e^i())$.

We say that the mechanism is strongly truthful if it is the only dominant strategy.

Note that by applying the revelation principle 2.2.1 to the declaration part, we can limit the discussion to mechanisms where the agents are simply requested to reveal their types.

Notation: We denote a mechanism with verification by a pair $m = (k(d), p(d, e))$, where $k()$ is the decision and $p()$ the payment function.

2.5.2 A Reformulation of Task Scheduling

Definition 20 (Task Scheduling with Verification) The problem is the same as the task allocation problem (definition 10), except that the mechanism is allowed to pay to the agents after the tasks have been performed. We assume that the times which the tasks were actually performed in are known to the mechanism.

More formally:

- A feasible output of the mechanism is denoted by a pair $(x, \tilde{t})$, where $x = x^1, \dots, x^n$ denotes the allocation of the tasks to the agents, and $\tilde{t} = \tilde{t}_1, \dots, \tilde{t}_k$ denotes the actual times that they were performed in. If $j \in x^i(t)$, then it must be that $\tilde{t}_j \geq t_j^i$.
- A strategy for an agent is composed of two parts: a declaration of its type and an execution of the tasks allocated to it. An agent may lie or choose to perform any task j allocated to it, in any time $\tilde{t}_j \geq t_j^i$.
- The objective function is $g(x, \tilde{t}) = \max_i \sum_{j \in x^i} \tilde{t}_j$ (the make-span).
- Agent i 's valuation is $v^i(x, \tilde{t}) = - \sum_{j \in x^i} \tilde{t}_j$.
- A mechanism is a pair (x, p) such that $x(t) = x^1(t), \dots, x^n(t)$ is the allocation function, and $p(t, \tilde{t}) = p^1(t, \tilde{t}), \dots, p^n(t, \tilde{t})$ is the payment.

2.5.3 The Compensation-and-Bonus Mechanism

The *Compensation-and-Bonus Mechanism* is composed of an optimal allocation algorithm, together with a well chosen payment function. The payment function is the sum of two terms, one is called the compensation, and the other the bonus.

Definition 21 (Compensation) *The function*

$$c^i(t, \tilde{t}) = \sum_{j \in x^i(t)} \tilde{t}_j$$

is called the compensation function for agent i .

The bonus function uses the following notion:

Definition 22 (Corrected Time Vector) *Let i be an agent, x an allocation. Given the agents' declarations t and the vector of actual times $\tilde{t}$, we define the corrected time vector for agent i as:*

$$\text{corr}^i(x, t, \tilde{t})_j = \begin{cases} \tilde{t}_j & \text{if } j \in x^i \\ t_j^l & \text{if } j \in x^l \text{ and } l \neq i \end{cases}$$

We define $\text{corr}^(x, t)$ of x and t to be the (unique) vector that satisfies $\text{corr}^*(x, t)_j = t_j^i$ for all i and $j \in x^i$.*

Definition 23 (Bonus) *The function*

$$b^i(t, \tilde{t}) = -g(x(t), \text{corr}^i(x(t), t, \tilde{t}))$$

is called the bonus function for agent i .

The bonus is calculated according to the *declarations* of the other agents and the *actual* times that the agent performed its assignments in.

Definition 24

(Compensation-and-Bonus Mechanism) *The Compensation-and-Bonus mechanism is given by an optimal allocation algorithm with the payment functions $p^i(t, \tilde{t}) = c^i(t, \tilde{t}) + b^i(t, \tilde{t})$.*

Theorem 2.5.1 *The Compensation-and-Bonus mechanism is a strongly truthful implementation of the task scheduling problem.*

Proof: We show that the only dominant strategy for each agent is to reveal its true type and to execute its tasks in minimal time.

Claim 2.5.2 *The Compensation-and-Bonus mechanism is strongly truthful.*

Proof: Let i be an agent, t^i its type and let d^{-i} denote the declarations for the other agents (note that the allocation and bonus given to i depend on d^{-i} but not on the actual execution times of the others). Let $t = (d^{-i}, t^i)$. Observing that the utility of an agent equals its bonus, and that for every allocation x the bonus for an agent i is maximized when executing its assignments in minimal time, it is enough to show that $-g(x(t), \text{corr}^*(x(t), t)) \geq -g(x(t^i, d^{-i}), \text{corr}^*(x(t^i, d^{-i}), t))$ for each t^i . This is immediately implied by the *optimality* of the allocation algorithm.

Clearly, when an agent does not follow this strategy, there are circumstances where this will increase the make-span and therefore decrease the agent's bonus. Therefore, the above strategy is the only dominant one. $\square$

When all agents follow their dominant strategies, the best possible make-span is obtained due to the optimality of the allocation algorithm. $\square$

An Example

	j_1	j_2	j_3
A_1	10	30	45
A_2	100	60	100

Figure 2.3: a type matrix for two agents

Consider the type matrix in figure 2.3. Assume first that both agents are truthful. The optimal allocation in this case is $\{\{j_1, j_3\}\{j_2\}\}$ and the make-span is 60, therefore the bonus given to each agent is -60 . Consider the case where agent 1 tries to "loose" j_3 declaring t_3^1 as 200. The "optimal" make-span therefore reduces to 100 and consequently the bonus for each agent reduces to -100 .

Similarly, when agent 1 tries to "gain" j_2 declaring for example t_2^1 to be 4, its bonus is reduced to -85 . If agent one is "lazy" executing its tasks in 100 units of time instead of 55, then its bonus reduces from -60 to -100 .

2.5.4 The Generalized Compensation-and-Bonus Mechanism

We now generalize the Compensation-and-Bonus mechanism:

Definition 25 (Generalized Compensation) *The function $c^i(t, \tilde{t})$ is called a generalized compensation for agent i if*

- $c^i(t, \tilde{t}) \leq \sum_{j \in x^i(t)} \tilde{t}_j$ for all t and $\tilde{t}$.
- Equality exists in case that the agent is truthful.

Definition 26 (Generalized Bonus) *Let $m^i(t^{-i}, w)$ be any positive real-valued function, that is strictly monotonically increasing in w . The function $b^i(t, \tilde{t}) = m^i(t^{-i}, -g(x(t), \text{corr}^i(t, \tilde{t})))$ is called a generalized bonus for agent i .*

Definition 27 (Generalized Compensation and Bonus Mechanism) *A Generalized Compensation and Bonus mechanism is a pair $m = (o, p)$ where:*

- $o()$ is an optimal allocation algorithm.
- $p^i(t, \tilde{t}) = c^i(t, \tilde{t}) + b^i(t, \tilde{t})$, where $c^i()$ and b^i are generalized compensation and bonus functions respectively.

Arguments, similar to 2.5.1 lead to the following theorem:

Theorem 2.5.3 *The Generalized Compensation and Bonus mechanism is a strongly truthful implementation of the task scheduling problem.*

□

2.5.5 Budget Considerations

Theorem 2.5.3 allows a lot of freedom in the design of a generalized Compensation-and-Bonus mechanism. In this section we take advantage of this freedom in order to satisfy two additional requirements: participation constraints and budget limits (see [60] for a detailed discussion).

Definition 28 (Participation Constraints) We say that a mechanism satisfies participation constraints if whenever an agent is truth-telling, its utility is non-negative. More formally: for each t and $\tilde{t}$ and for each agent i , if $\tilde{t}_j = t_j$ for all $j \in x^i(t)$, then $p^i(t, \tilde{t}) + v^i(x(t), \tilde{t}) \geq 0$.

Note: If a truthful approximation mechanism satisfies the property that whenever an agent is not allocated any tasks then its payment is non-negative, it needs to satisfy participation constraints as well. (Because by declaring high enough values the agent can force the mechanism to allocate no tasks to it.)

Theorem 2.5.4 *There exists a strongly truthful mechanism that satisfies participation constraints for the task scheduling problem.*

Proof: We define $g^{-i}(t^{-i})$ to be the optimal make-span among the allocations which do not include agent i . We define the *contribution* of agent i as $cont^i(t^{-i}, \tilde{t}) = g^{-i}(t^{-i}) - g(x(t), corr^i(x(t), t, \tilde{t}))$. A generalized Compensation-and-Bonus mechanism where the bonus of each agent equals its contribution clearly satisfies participation constraints. $\square$

So far we have not limited the amount of money that a mechanism can pay to the agents. When considering mechanisms with limited budget one cannot ignore the fact that such a mechanism may not be able to allocate the tasks at all. We therefore change the problem definition.

Definition 29 (Constrained Task Scheduling) *The problem is the same as in definition 20 except that there is another possible output $\perp$ (indicating that the tasks are not allocated). $v^i(\perp)$ is taken to be zero and $g(\perp)$ is taken to be ∞ .*

Definition 30 (Limited Budget) *We say that a mechanism $m = (o, p)$ has a limited budget b if for any tuple of strategies $s = s^1, \dots, s^n$, $\sum_i p^i(s) \leq b$. We call a revelation mechanism b -constrained if it satisfies participation constraints and has a limited budget b .*

Note that if $Y(t)$ is the allocation of a b -constrained mechanism then it must be that $\sum_i \sum_{j \in Y^i} t_j^i \leq b$.

Definition 31 *Let $m = (o, p)$ and $m' = (o', p')$ be two mechanisms for the same minimization problem. We say that m is as good as m' if for any dominant strategies $d = d^1, \dots, d^n$ for the agents in m and $s = s^1, \dots, s^n$ for the agents in m' , $g(o(d)) \leq g(o'(s))$.*

Theorem 2.5.5 *Let $b, h > 0$. Then there exists a truthful mechanism m for the constrained task scheduling problem such that*

1. m is $(b + h)$ -constrained.
2. If m' is b -constrained, then m is as good as m' .

Proof: Consider the following mechanism $m = (o, p)$:

- The output algorithm $o(t)$ finds an optimal allocation among the allocations y such that $\sum_i \sum_{j \in y^i} t_j^i \leq b$ (or outputs $\perp$ if none exists).
- Let $m()$ be a bounded strictly monotone increasing function such that $0 \leq m() < h/n$. We define the contribution $cont^i$ of the agent as in 2.5.4 (except that the algorithm and the objective function are different). We define the bonus $b^i()$ as $m(cont^i())$ and the payment to be given by $p^i = c^i() + b^i()$ where $c()$ is a compensation function. The payment is defined to be 0 if the output is $\perp$.

The total compensation is bounded by b and the total bonus by h , therefore the budget is bounded by $b + h$. Arguments, similar to 2.5.1 show that the mechanism is truthful and that the only dominant strategies for an agent i are to reveal the truth (when $t_j^i \geq b$ the agent may declare on any $d_j^i \geq b$) and to perform its tasks as efficient as it can.

Recalling that any b -constrained mechanism m' needs to choose an allocation z such that $\sum_i \sum_{j \in z^i} t_j^i \leq b$, the theorem is proved. $\square$

2.5.6 Poly-Time Mechanisms

While the Compensation-and-Bonus mechanisms are optimal, note that they are intractable from a computational point of view due to their use of the exponential-time optimal allocation algorithm. One would be tempted to take a known polynomial-time approximation algorithm for the problem and base a mechanism upon it, obtaining a polynomial-time approximation mechanism. Unfortunately, this is not so simple and we do not know how to do this in general. In this section we first show that replacing the optimal allocation algorithm with a non-optimal approximation in the Compensation-and-Bonus mechanism does not preserve truthfulness. A similar argument can be made for the important case of VCG mechanisms (section 2.3.1). We then define a sub-case of the scheduling problem, where the number of agents is fixed and there are known bounds for the execution times t_j^i . This problem is still NP -hard and the lower bounds presented in sections 2.4.3 and 2.4.3 can be applied to it (with slightly weaker constants depending on the bounds). Nevertheless, for any $\epsilon > 0$ we are able to present a $1 + \epsilon$ polynomial approximation mechanism for this variant. Our approximation mechanism is based on a rounding technique developed by [36].

Definition 32 Let $x()$ be an allocation algorithm. The Compensation-and-Bonus mechanism based on x is the same as 2.5.3 except that the optimal algorithm is replaced by $x()$.

Theorem 2.5.6 Let $x()$ be a non-optimal approximation algorithm for task scheduling. Let $m = (x, p)$ be the Compensation-and-Bonus mechanism based on $x()$. Then m is not truthful.

Proof: Assume by contradiction that it is truthful. For an allocation y and a type t , let $g(y, t)$ denote the make-span $-\max_i \sum_{j \in y^i} t_j^i$; let $opt(t)$ denote an optimal allocation.

Let t be a type such that $g(opt(t), t) < g(x(t), t)$, and let t'^1 be a type for agent 1 such that

$$t_j'^1 = \begin{cases} t_j^1 & \text{if } j \in opt^1(t) \\ \infty & \text{otherwise} \end{cases}$$

where ∞ stands for an arbitrary high value.

Claim 2.5.7 Let $t' = t'^1, t^2, \dots, t^n$. Then $g(x(t'), t') \geq g(x(t), t)$.

Proof: Otherwise, in the case where agent 1's type is t'^1 , it would be more beneficial for it to “pretend” to be t'^1 (note that this cannot be verified!). This contradicts the truthfulness of the mechanism. $\square$

Corollary 2.5.8 Let s be a type such that

$$s_j^i = \begin{cases} t_j^i & \text{if } j \in opt^i(t) \\ \infty & \text{otherwise} \end{cases}$$

Then $g(x(s), s) \geq g(x(t), t)$.

$\square$

Since $g(x(s), s) \geq g(x(t), t) > g(opt(t), t) = g(opt(s), s)$, we obtain that $x(s) \neq opt(s)$. Thus there exists an agent who is allocated an ∞ job, in contradiction to the approximation ratio of $x()$. $\square$

Definition 33 (Bounded Scheduling Problem) The problem is the same as in definition 20, except that the number of agents n is fixed to a constant and there exist fixed $b > a > 0$ such that for all i, j $a \leq t_j^i \leq b$.

The *rounding* algorithm presented in [36] provides a $(1 + \epsilon)$ -approximation for bounded scheduling and runs in polynomial time. It basically works as follows: The entries t_j^i are first rounded up to integer multiples of δ (a parameter chosen as a function of a and ϵ). It then exactly solves this rounded problem using dynamic programming (in polynomial time). The solution of the rounded problem is shown to be a $1 + \epsilon$ approximation to the original one.

We will attach to this algorithm a carefully chosen payment function as to obtain our mechanism. The idea is to use the exact times for the compensation function, but the rounded ones for the bonus function.

Notation: For a vector t , let $\hat{t}$ denote the vector where all entries are rounded up to an integer multiple of δ . Denote also $\hat{g}(x, \hat{t}) = g(x, \hat{t})$, where g is the make-span objective function.

Definition 34 (Rounding Mechanism) *The rounding mechanism is defined as follows:*

- *The allocation algorithm is the rounding algorithm of [36] sketched above.*
- *The payment function is given by: $p^i(t, \tilde{t}) = c^i(t, \tilde{t}) + b^i(t, \tilde{t})$, where*

$$c^i(t, \tilde{t}) = \sum_{j \in x^i(t)} \tilde{t}_j$$

$$b^i(t, \tilde{t}) = -\hat{g}(x(\hat{t}), \text{corr}^i(x(t), t, \tilde{t}))$$

The rounding mechanism compensates the agents according to their actual work, but computes the bonus according to the rounded declarations and execution times.

Theorem 2.5.9 *For every fixed $\epsilon > 0$ the rounding mechanism is a polynomial time mechanism with verification that truthfully implements a $1 + \epsilon$ approximation for the bounded task scheduling problem.*

Proof:(sketch) When the types and the actual computation times are rounded, $\hat{g}$ is exactly the make-span, and the rounding algorithm is optimal. Arguments, similar to those in 2.5.1, therefore show that the only dominant strategies for agent i are to declare on a type t'^i such that t'^i and t^i have the same rounded value, and to execute its tasks such that after rounding, $\text{corr}^i(x(t), t, \tilde{t})$ equals $\text{corr}^*(x(t), t)$. Clearly, when all agents follow such strategies, the result is a $1 + \epsilon$ approximation. In particular, truth-telling is among the dominant strategies. $\square$

2.6 Conclusions and Further Research

In this paper we suggested a framework for studying optimization problems that involve selfish participants. We studied a representative task scheduling problem under two main models: a basic mechanism design based model and a model that allows more information to be incorporated into the mechanism. Under the assumptions of the basic model we showed that the problem cannot be approximated within a factor of $2 - \epsilon$. Then, under the second model assumptions, we introduced several novel mechanisms including optimal, constrained optimal and polynomial-time

approximation mechanisms. We have also shown that worst case behavior can be improved using randomness without weakening the “game-theoretic” requirements of the mechanism.

We believe that our work is only a first step towards understanding the notion of algorithmic cooperation among selfish agents. There are clearly many open problems and research directions, and we are far from a situation where we could design, analyze, and implement protocols and algorithms that directly take into account the participants’ differing goals.

We divide the basic issues for further research into three main categories: questions directly coming out of our work, game-theoretic extensions to our model and distributed computation issues.

Several questions directly stem from our work. For example, there are large gaps between the upper and lower bounds for both task scheduling without verification and for poly-time task scheduling with verification.

Many game-theoretic extensions to our model are possible. For example one may consider different settings (e.g. repeated games), different solution concepts (e.g. Bayesian-Nash), and different assumptions (e.g. partial verification).

Finally, in this work we have treated the mechanism as a black box, and have not considered how its function is actually carried out in a distributed manner. A whole set of open problems comes from trying to “open up” this black box, and analyze the steps taken in implementing the mechanism from a distributed point of view. For example when communication costs are considered, even the revelation principle breaks up; non complete network topology may be exploited by the agents to extract information about others and to cooperate; cryptography may be introduced and distributed handling of the payments may be considered.

Acknowledgments: We thank Dov Monderer, Motty Perry and Ori Regev for helpful discussions at various stages of this work. We thank Daniel Lehmann, Ofer Neyman, Dana Peer, Inbal Ronen and Moshe Tennenholtz for comments on earlier drafts of this paper.

Chapter 3

Computationally Feasible VCG Mechanisms

This sections contains the paper “Computationally Feasible VCG Mechanisms” as submitted to EC00.

Computationally Feasible VCG Mechanisms

Noam Nisan

Institute of Computer Science, Hebrew University of Jerusalem, Givat Ram 91904, Israel.

Email: noam@cs.huji.ac.il

And

Amir Ronen

Institute of Computer Science, Hebrew University of Jerusalem, Givat Ram 91904, Israel.

Email: amiry@cs.huji.ac.il¹

Abstract

One of the major achievements of mechanism design theory is the family of truthful (incentive compatible) mechanisms often called VCG (named after Vickrey, Clarke and Groves). When applying VCG mechanisms to complex mechanism design problems such as combinatorial auctions a problem emerges: even finding optimal outcomes is computationally intractable. A striking observation is that if the optimal outcome is replaced by the results of computationally tractable approximation algorithms or heuristics then the resulting mechanism (termed VCG-based) is no longer necessarily truthful!

The first part of this paper considers this problem in depth and shows that it is almost universal. Specifically, we prove that essentially all reasonable approximations or heuristics for combinatorial auctions as well as a wide class of cost minimization problems yield non-truthful VCG-based mechanisms.

The second part of this paper proposes a method for handling this non-truthfulness. We introduce a notion of *feasible truthfulness* that captures the limitation on agents imposed by their own computational limits. We then show that under reasonable assumptions on the agents, it is possible to turn *any* VCG-based mechanism into a feasibly truthful one, using an additional appeal mechanism.

The resulting mechanism also satisfies participation constraints.

¹Both authors are supported by grants from the Israeli ministry of Science and the Israeli ministry of science

3.1 Introduction

The theory of mechanism design may be described as studying the design of protocols under the assumption that the participants behave according to their own goals and preferences and not necessarily as "instructed" by the protocol. This theory has been traditionally used in economic settings such as auctions of various kinds (for an introduction see e.g. [60, 81]). Recently, with the emergence of the Internet as *the* distributed computing platform, the theory of mechanism design has been applied in computational settings such as distributed resource and task allocation [77, 117, 116], communication networks [19, 46], multi-agent systems [91], and others.

The canonical mechanism design problem can be described as follows: A set of rational agents need to collaboratively choose an outcome o from a finite set O of possibilities. Each agent i has a privately known valuation function $v^i : O \rightarrow R$ quantifying the agent's benefit from each possible outcome. The agents are supposed to report their valuation functions $v^i(\cdot)$ to some centralized mechanism that chooses an outcome o that maximizes the total welfare $\sum_i v^i(o)$.

The main difficulty is that agents may choose not to reveal their true valuations but rather report carefully designed lies in an attempt to influence the outcome to their liking. The tool that the mechanism uses to motivate the agents to reveal the truth is monetary payments – these payments are to be designed in a way that ensures that rational players always reveal their true valuations – making the mechanism, so called, incentive compatible or truthful.

There is only one general technique known for designing such a payment structure, sometimes called the generalized Vickrey auction [112], the Clarke pivot rule [8] the Groves mechanism [31], or, as we will, VCG. In certain senses this payment structure is unique [30, 89].

In recent years mechanisms have become quite complicated, requiring implementation on computer systems. Cases in point include combinatorial auctions [95, 23, 33, 53, 109, 66, 75, 10, 62], where multiple items are concurrently sold in an auction, as well as various computational task and resource allocation problems [77, 117, 116, 51, 19]. For many of these applications the space of possible outcomes is huge and finding an outcome that maximizes the declared total welfare is computationally infeasible (NP-complete). In these cases any realistic mechanism will not be able to obtain the optimal outcome.

A natural general approach for the development of mechanisms for complex problems would be to use a computationally feasible approximation algorithm or heuristic for obtaining a near-optimal outcome. The payments in such an approach would presumably be derived by applying the VCG payment rules to the underlying algorithm. We term such mechanisms *VCG-based*.

The starting point of this paper is the surprising observation, noticed already by some re-

searchers ([53, 78]), that VCG-based mechanisms are not necessarily truthful! I.e. if the choice of outcome is sub-optimal, then applying the VCG payment rules will not necessarily motivate the agents to reveal the truth – rational agents may lie taking advantage of quirks in the outcome choice algorithm.

The first part of this paper exams this phenomena in depth and shows that it is near universal: essentially *all* reasonable VCG-based mechanisms are *not* truthful. We first point our attention at combinatorial auctions and characterize the class of truthful VCG-based mechanisms for this problem. This characterization yields, in particular, the following corollary. Call an allocation algorithm for combinatorial auctions reasonable if whenever an item is valued by only a single agent, then this agent receives that item.

Theorem: Any reasonable VCG-based mechanism for combinatorial auctions is not truthful (unless it uses the – computationally intractable – optimal allocation algorithm).

We then study a family of problems termed "cost minimization allocation problems". We call an algorithm for such a problem degenerate if it produces results which are arbitrarily far from the optimal.

Theorem: Any non-degenerate VCG-based mechanism for any cost minimization allocation problem is not truthful (unless it uses the optimal allocation algorithm).

We then examine carefully why VCG-based mechanisms are non-truthful, and observe that the only reason for an agent to lie about her valuation is to "help" the algorithm to improve the overall result. However, if finding a better outcome is hard, then we would not expect an agent to be computationally able to do so! This motivates us to define a feasible notion of truthfulness, a notion that takes into account the computational limits of the agents.

In the second part of this paper we first define this "feasible" notion of truthfulness. We then introduce a variant on VCG-based mechanisms, called the second-chance mechanism² and show that under reasonable assumptions, any such mechanism is indeed "feasibly truthful".

Our notion of truthfulness is based on the view that each agent has some *strategic knowledge* and that her range of actions is determined by this knowledge. This knowledge may represent the computational limitations of the agent or some structural limitations on her information or behavior. We say that an action is feasibly dominant for the agent if her strategic knowledge does not contain awareness of *any* circumstances where another action is better for her. We say that a mechanism is feasibly truthful if the truth is feasibly dominant for all agents.

²The mechanism is patent pending.

Given *any* algorithm for the corresponding optimization problem we define the *second-chance* mechanism based on it. This mechanism is a modification of the VCG-based mechanism where in addition to their type declarations, the agents are allowed to submit *appeal functions* as well. An appeal function is an encapsulation of the agents' algorithmic ability to help the mechanisms' underlying outcome determination algorithm by means of input modification.

We show that under reasonable assumptions on the agents, the addition of this appeal mechanism is enough to ensure feasible truthfulness. Specifically we prove several variants of theorems of the following schema. The different theorems differ from each other in the exact definition of computational limitation.

Theorem Schema: If all agents' strategic knowledge functions are computationally limited and the outcome determination algorithm is computationally limited then the second chance mechanism is feasibly truthful. Furthermore, the appeal functions are all computationally limited, and the mechanism's outcome is at least as good as that of the underlying algorithm. The resulting mechanism satisfies participation constraints as well.

We present our work in the context of VCG-mechanisms. However our results are more general. In particular all they can be applied to the compensation and bonus mechanism [78] and to weighted versions of the VCG method.

The rest of this paper is organized as follows: In section 3.2 we introduce our model and notations. Limitations of truthful VCG-based mechanisms are studied in section 3.3. In section 3.4 we introduce the second-chance mechanism and show some of its properties.

3.2 Preliminaries

In this section we formally present our model. We attempt, as much as possible, to use the standard notions from both mechanism design and computational complexity. We focus on dominant strategy implementation under quasi-linear environments.

In subsection 3.2.1 we describe mechanisms and mechanism design problems³. In subsection 3.2.2 we describe the celebrated VCG mechanisms and present our VCG-based mechanisms.

In subsection 3.2.3 we discuss the computational burden of VCG mechanisms and briefly describe some fundamental notions of computational complexity.

As an example, we formulate, in subsection 3.2.4, the important problem of combinatorial auctions.

³In this paper we discuss only what are called utilitarian problems. Without loss of generality we limit ourselves to direct revelation mechanisms.

3.2.1 Mechanism design problems

Definition 35 (Utilitarian mechanism design problem) A (utilitarian) mechanism design problem is described by the following:

1. A finite set O of allowed outputs.
2. Each agent $i = (1, \dots, n)$ has a real function $v^i(o \in O)$ called its valuation or type. This is a quantification of its benefit from each possible output o in terms of some common currency. v^i is **privately** known to agent i . The space V^i of all possible valuation functions is called the type space of the agent.
3. If the mechanism's output is o and in addition the mechanism hands the agent p^i units of this currency, then her utility u^i equals⁴ $v^i(o) + p^i$. This utility is what the **agent** aims to optimize.
4. The goal of the **mechanism** is to select an output $o \in O$ that maximizes the total welfare $g(v, o) = \sum_i v^i(o)$.

An example for such a problem can be found in section 3.2.4.

In a direct revelation mechanism, the participants are simply asked to reveal their types to the mechanism. Based on these declarations the mechanism then computes the output o and the payment p^i for each of the agents.

Definition 36 (A mechanism) A (direct revelation) mechanism is a pair $m = (k, p)$ such that:

- The output function k accepts as input a vector $w = (w^1, \dots, w^n)$ of declared valuation functions⁵ and returns an output $k(w) \in O$.
- The payment function $p(w) = (p^1(w), \dots, p^n(w))$ returns a real vector. This is the payment handed by the mechanism to each of the agents (e.g. if $p^i = -2$ then the agent pays two units of currency to the mechanism).

A revelation mechanism computes its output according to the type declarations of the agents. As they may **lie** to the mechanism, it should be carefully designed such that it will be for the benefit of each agent to reveal her true type to the mechanism.

Notation: We denote the tuple $(a^1, \dots, a^{i-1}, a^{i+1}, \dots, a^n)$ by a^{-i} . We let (a^i, a^{-i}) denote the tuple $(a^1, \dots, a^n)$.

⁴This is called the quasi-linearity assumption.

⁵We do not consider the issue of how to represent the valuations.

Definition 37 (truthful mechanism) *A mechanism is called truthful if truth-telling is a dominant strategy. I.e. for every agent i of type v^i and for every type declaration w^{-i} for the other agents, the agent's utility is maximized when she declares her real valuation function v^i .*

As an example consider the famous Vickrey auction [112]: A seller wishes to sell one item in an auction. There are n buyers, each privately knowing her valuation v^i for this item. In a Vickrey auction each of the buyers is simply asked for her valuation. The item is allocated to the buyer with the highest bid for the price equal the second highest bid. The reader may verify that this mechanism is truthful, i.e. that it is always for the agent's benefit to declare her true type v^i . Another example for a truthful mechanism can be found in section 3.2.4.

3.2.2 VCG-based mechanisms

In this subsection we present the celebrated VCG mechanisms. Intuitively these mechanisms solve utilitarian problems by identifying the utility of truthful agents with the declared total welfare.

Definition 38 (VCG mechanism) *A mechanism $m = (k, p)$ belongs to the VCG family if:*

- $k(w)$ maximizes the total welfare according to w .
- The payment is calculated according to the VCG formula: $p^i(w) = \sum_{j \neq i} w^j(k(w)) + h^i(w^{-i})$ ($h^i()$ is an arbitrary function of w^{-i}).

Theorem 3.2.1 ([31]) *A VCG mechanism is truthful.*

□

It is worth notifying that weighted VCG mechanisms are possible as well (see e.g. [89] [78]).

For many applications, the task of finding an output $k(w)$ that maximizes the total welfare is computationally infeasible. In this paper we consider mechanisms where the optimal algorithm is replaced by a sub-optimal but computationally feasible one.

Definition 39 (VCG-based mechanism) *Let $k(w)$ be an algorithm that maps type declarations into allowable outputs. We call $m = (k(w), p(w))$ a VCG mechanism based on k if p is calculated according to the VCG formula: $p^i(d) = \sum_{j \neq i} w^j(k(w)) + h^i(w^{-i})$ (where $h^i()$ is an arbitrary function of w^{-i}).*

Obviously, a VCG-based mechanism that is based on an optimal algorithm is a VCG mechanism.

3.2.3 Computational considerations in mechanism design

Computer scientists make a sharp distinction between two major classes of computational problems. The first class is called P and contains all the easy problems. I.e. problems that can be solved in time polynomial in the bit-size of the input. The second class is called NP -Complete and contains problems which are provably hard. To date, no sub-exponential algorithm for any NP -Complete problem has been found. Unfortunately, many problems of interest (e.g. finding an allocation that maximizes the total welfare in a combinatorial auction) are NP -Complete. Fortunately, for many of these problems, various polynomial time heuristics and approximation algorithms are available. An introduction to the theory of computational complexity can be found in [24] [83].

We define these two principal notions in terms of mechanisms:

Definition 40 *A mechanism (k, p) is called polynomial time computable if both $k(w)$ and $p(w)$ run in polynomial time (using a standard encoding of w).*

Note that a VCG-based mechanism is polynomial iff its output algorithm is polynomial. We sometimes call polynomial algorithms and mechanisms computationally feasible.

Definition 41 *A mechanism design problem is called NP -complete if the problem of finding an output that maximizes the total welfare is NP -complete.*

3.2.4 Example: Combinatorial auctions

The problem of combinatorial auction has been extensively studied in recent years (see e.g. [53] [95] [23] [33] [75]). The importance of this problem is twofold. First, several important applications rely on it (e.g. the FCC auction [62]). Second, it is a generalization of many other problems of interest, in particular in the field of electronic commerce.

The problem: A seller wishes to sell a set S of items (radio spectra licenses, electronic devices, etc.) to a group of agents who desire them. Each agent i has, for every subset $s \subseteq S$ of the items, a number $v^i(s)$ that represents how much s is worth for her. We assume that $v^i(\cdot)$ is privately known to the agent.

We take two standard additional assumptions on the type space of the agents:

No externalities The valuation of each agent depends only on the items allocated to her. I.e. $\{v^i(s) | s \subseteq S\}$ completely represents the agent's valuation.

Free disposal Items have non-negative values. I.e if $s \subseteq t$ then $v^i(s) \leq v^i(t)$. Also $v^i(\emptyset) = 0$.

Note that the problem allows items to be complementary, i.e. $v^i(S \cup T) \geq v^i(S) + v^i(T)$ or substitutes i.e. $v^i(S \cup T) \leq v^i(S) + v^i(T)$ (S, T disjointed). For example a buyer may be willing to pay \$200 for T.V set, \$150 for a VCR, \$450 for both and only \$200 for two VCRs.

When an agent payment is p^i for a set of items s^i then her overall utility is $p^i + v^i(s^i)$. Note that here the payment is non-positive. This utility is what each agent tries to optimize. For example an agent prefers to buy a \$1000 valued VCR for \$600 gaining \$400 to buying a \$1500 valued VCR for \$1250.

In a VCG mechanism for such an auction, the participants are first required to reveal their valuation functions to the mechanism. The mechanism then computes, according to the declarations of the agents, an allocation s that maximizes the total welfare. The payment for each of the agents is calculated according to the VCG formula. By theorem 3.2.1, the utility $u^i = p^i + v^i(s^i)$ of each of the agents is maximized when she reveals her true valuation to the mechanism. When all agents are truthful, the mechanism maximizes the total welfare.

Consider however the computational task faced by such a mechanism. After the types are declared, the mechanism needs to select, among all possible allocations, one that maximizes the total welfare. This problem is known to be NP-Complete. Therefore, unless the number of agents and items is very small, such a mechanism is computationally infeasible. Note that even the problem of finding an allocation that approximates the optimal allocation within a reasonable factor is NP-Complete (under the common complexity assumption that $RP \neq Co - NP$, see e.g. [95, 53]). Nevertheless, various heuristics and tractable sub-cases have been analyzed in the literature [95] [23] [33] [75] [109].

3.3 Limitations of truthful VCG-based mechanisms

In this section we study the limitations of truthful VCG-based mechanisms. In subsection 3.3.1 we characterize these mechanisms for important problem of combinatorial auctions (see subsection 3.2.4). This characterization precludes the possibility of obtaining truthfulness by applying VCG rules to many of the proposed heuristics for combinatorial auctions, e.g. the greedy algorithms in [53] and [75]. Moreover we show that any truthful non-optimal VCG-based mechanism for combinatorial auctions suffers from abnormal behavior. In subsection 3.3.2 we show that for many natural cost minimization problems, any truthful VCG-based mechanism is either optimal or produces results which are arbitrarily far from the optimal. As a result, when such a problem is computationally intractable, any truthful computationally feasible VCG-based mechanism will produce unreasonable results.

3.3.1 Truthful VCG-based mechanisms for combinatorial auctions

In this section we characterize the class of truthful VCG-based mechanisms for combinatorial auctions (subsection 3.2.4).

Definition 42 Let $k(w)$ be an algorithm that maps type declarations into allowable outputs. Let V' be a subspace of the space of all possible types $V \stackrel{\text{df}}{=} \prod_{i=1}^n V^i$. Let $\mathcal{O}$ denote the range of k at V' , i.e. $\mathcal{O} = \{k(w) | w \in V'\}$. We say that k is maximal in its range at V' if for every type $w \in V'$, $k(w)$ maximizes g over $\mathcal{O}$. We say that k is maximal in its range if it is maximal in its range at V .

As an example consider an algorithm for combinatorial auctions that allocates all the items (the set S) to the agent with the highest valuation $v^i(S)$. Clearly, this computationally efficient algorithm is maximal in its range. The result of this algorithm is never worse than $1/n$ and also $1/|S|$ times the optimal result (n denotes the number of the agents).

Proposition 3.3.1 A VCG-based mechanism with an output algorithm that is maximal in its range is truthful.

Proof: Such a mechanism is a VCG mechanism where the set of allowable outputs is the range of its output algorithm. By theorem 3.2.1 such a mechanism is truthful. $\square$

We show that this almost characterizes the class of truthful VCG-based mechanisms for the combinatorial auction problem.

Notation: We let $\tilde{V}$ denote the space of all types $v = (v^1, \dots, v^n)$ such that for any two different allocations x and y , $g(v, x) \neq g(v, y)$. (Recall that $g(\cdot)$ denotes the total welfare.)

It is not difficult to see that $\tilde{V}$ contains almost all the types, i.e. $V - \tilde{V}$ is zero measured.

Theorem 3.3.2 If a VCG-based mechanism for the combinatorial auction problem is truthful then its output algorithm is maximal in its range at $\tilde{V}$.

Proof: Assume by contradiction that $m = (k, p)$ is truthful but k is not maximal in its range at $\tilde{V}$. Without loss of generality assume that $p^i(w) = \sum_{j \neq i} w^j(k(w))$ (i.e. we let $h^i(\cdot)$ in the VCG formula to be zero). The utility of a truthful agent i , when the declarations of the other agents are w^{-i} , equals $g((v^i, w^{-i}), k(v^i, w^{-i}))$.

Let $\mathcal{O}$ denote the range of k at $\tilde{V}$ and let $v \in \tilde{V}$ be a type such that $k(v)$ is not optimal over $\mathcal{O}$. Let $y = \text{opt}_{\mathcal{O}}(v)$. Among the optimal allocations in $\mathcal{O}$ we also require that $y = (y^1, \dots, y^n)$

allocates all the items to the agents, i.e. $\bigcup_i y^i = S$. Note that this is possible due to the free disposal assumption. Finally let $w \in \tilde{V}$ be a type such that $y = k(w)$.

Define a type z by

$$z^i(s) = \begin{cases} v^i(s) & \text{if } s \not\supseteq y^i \\ \alpha & \text{if } s \supseteq y^i \end{cases}$$

where α stands for a sufficiently large number. In other words, each agent i strongly desires the set y^i . We assume that $z \in \tilde{V}$, otherwise we could add sufficiently small “noise” $\epsilon^i(s)$ to z such that all the following claims remain true.

Lemma 3.3.3 $y = k(z)$.

Proof: Define a sequence of type vectors by:

$$\begin{aligned} w_0 &= (w^1, \dots, w^n) \\ w_1 &= (z^1, w^2, \dots, w^n) \\ &\vdots \\ w_n &= (z^1, \dots, z^n) \end{aligned}$$

We assume that $w_j \in \tilde{V}$ for all j . It is not difficult to see that z could be defined in a way that guarantees this.

Claim 3.3.4 $k(w_1) = y$.

Proof: Assume by contradiction that this is false. From the definition of $\tilde{V}$ we obtain that $g(w_1, k(w_1)) \neq g(w_1, y)$. Consider the case where agent 1’s type is z^1 and the types of the others are $w^2, \dots, w^n$. By declaring w^1 , agent 1 can force the algorithm to decide on y . Therefore it must be that $g(w_1, k(w_1)) > g(w_1, y)$. In particular since α is large, it must be that $k^1(w^1) \supseteq y^1$. Thus, $\alpha + \sum_{j=2}^n w^j(k(w_1)) > \alpha + \sum_{j=2}^n w^j(y)$. As, due to the free disposal assumption, $w^1(k^1(w^1)) \geq w^1(y)$, we obtain that $w^1(k^1(w^1)) + \sum_{j=2}^n w^j(k(w_1)) > w^1(y) + \sum_{j=2}^n w^j(y)$ i.e. $g(w_0, k(w^1)) > g(w_0, y)$. Therefore when the type of agent 1 is w^1 , she is better off declaring z^1 in contradiction to the truthfulness of the mechanism. $\square$

Similarly we obtain that $k(w_j) = y$ for all j . This completes the proof of lemma 3.3.3. $\square$

Consider the following sequence of type vectors:

$$\begin{aligned} v_0 &= (v^1, \dots, v^n) \\ v_1 &= (z^1, v^2, \dots, v^n) \\ &\vdots \\ v_n &= (z^1, \dots, z^n) \end{aligned}$$

Claim 3.3.5 *For all v_j , y maximizes g on $\mathcal{O}$.*

Proof: y maximizes g for v_0 . Due to the free disposal assumption, an optimal allocation for v_j allocates y^i for every agent $i \leq j$. It must also to allocate the rest of the items optimally among the rest of the agents. This is also done according to y . $\square$

Claim 3.3.6 $k(v_{n-1}) = y$.

Proof: Otherwise, when agent n 's type is v^n , she is better off declaring z^n obtaining $g(v_{n-1}, y)$. This contradicts the truthfulness of the mechanism. $\square$

Similarly, we obtain that $k(v_0) = y$. A contradiction. This completes the proof of theorem 3.3.2. $\square$

This result characterize the output algorithms that could be incorporated in a VCG-based mechanisms accept maybe a measure zeroed subset of the types. We can now give a complete characterization:

Corollary 3.3.7 *Consider a VCG-based mechanism for a combinatorial auction with an output algorithm k . If the mechanism is truthful then there exists an output algorithm $\tilde{k}$, maximal in its range, such that for every v , $g(v, k(v)) = g(v, \tilde{k}(v))$.*

Proof: Let $g_k(v) \stackrel{df}{=} g(v, k(v))$. It is not difficult to see that $g_k(v)$ must be contiguous in v . As $\tilde{V}$ is dense in V our corollary is proved. $\square$

This result gives rise to several interesting algorithmic and combinatorial questions. For example given an approximation factor $c \geq 1$, what is the minimal size of a sub-family $\mathcal{O} \subseteq \mathcal{O}$ such that for every v , $\max_{y \in \mathcal{O}} g(v, y) \geq c \cdot g_{opt}(v)$.

We now show that non-optimal truthful VCG-based mechanisms suffer from the following disturbing abnormal behavior:

Definition 43 (reasonable mechanism) *A mechanism for combinatorial auctions is called reasonable if whenever there exists an item j and an agent i such that*

- *For all S , if $j \notin S$ then $v^i(S \cup \{j\}) > v^i(S)$.*
- *For every agent $l \neq i$, $v^i(S \cup \{j\}) = v^i(S)$.*

then j is allocated to agent i .

In other words, when one of the agents desires some item and the others do not, then the “right one” gets it.

Theorem 3.3.8 *Any non-optimal truthful VCG-based mechanism for combinatorial auctions is not reasonable.*

Proof: By theorem 3.3.7, let $S = (S^1, \dots, S^n)$ be an allocation which is not in the range of the output algorithm. Define a type vector v by $v^i(X) = |X \cap S^i|$. Clearly each item j is desired by a single agent. As the mechanism’s allocation is not S , there exist at least one item which is not allocated to the “right” agent. $\square$

3.3.2 Truthful VCG-based mechanisms for cost minimization problems

We now show that for many natural cost minimization problems, any truthful VCG-based mechanism is either optimal or produces results which are arbitrarily far from the optimal.

We start with an example.

Multicast transmissions: A communication network is modeled by a directed graph $G = (V, E)$. Each edge e is a privately owned link. The cost t_e of sending a message along that edge is *privately* known to its owner. Given a source $s \in V$ and a set $T \subseteq V$ of terminals, the mechanism must select a subtree rooted in s that covers all the terminals. The message is then broadcasted along this tree. We assume that no agent owns a cut in the network.

Naturally, the goal of the mechanism is to select, among all possible trees, a tree R that minimizes the total cost: $\sum_{e \in R} t_e$. The natural goal of each agent however is to maximize her *own* profit: $p^i - \sum_{(e \in R \text{ owned by } i)} t_e$. It is not difficult to see that this is a utilitarian mechanism design problem.

This example was introduced at [19] (under a different model). It is motivated by the need to broadcast long messages (e.g. movies) over the Internet. We now generalize this example.

Definition 44 (cost minimization allocation problem)

A cost minimization allocation problem (CMAP) is a mechanism design problem described by:

Type space The type of each agent i is described by a vector $(v_1^i, \dots, v_{m_i}^i)$. We let $m = \sum_i m_i$.
(In our multicast example the v_j^i s correspond to the negation of the costs t_e .)

Allowable outputs Each output is denoted by a bit vector $x = (x_1^1, \dots, x_{m_1}^1, \dots, x_1^n, \dots, x_{m_n}^n) \in \{0, 1\}^m$. We denote $(x_1^i, \dots, x_{m_i}^i)$ by x^i . There may be additional constraints on the set O of

allowable outputs. (In our example x should correspond to a tree the network's graph where x_j^i equals 1 iff the corresponding edge is in the tree.)

such that the following conditions are satisfied:

Non-boundedness if $v^i = (v_1^i, \dots, v_{m_i}^i)$ describes a type for agent i and $w^i \leq v^i$ (as vectors), then w^i also describe a type.

Independence and monotonicity Each valuation v^i depends only on i 's bits x^i . (In our example, the agent valuation of a given tree depends only on her own edges in it.) If for all j , $w_j^i \leq v_j^i$ then for every output x , $w^i(x^i) \leq v^i(x^i)$.

Forcing condition For every type v , an allowable output x and a real number α , define a type $v[\alpha]$ by

$$v[\alpha]_j^i = \begin{cases} v_j^i & \text{if } x_j^i = 1 \\ \alpha & \text{otherwise} \end{cases}$$

The forcing condition is satisfied if for every allowable output $y \neq x$, $\lim_{\alpha \rightarrow -\infty} g(t(\alpha), y) = -\infty$.

Many natural problems in which the goal is to minimize the total cost under given constraints belong to this class. In particular the reader may verify that our multicast example is of this kind.

Notations: For a type v we let $g_{opt}(v)$ denote the optimal value of g . We denote $g(v, k(v))$ by $g_k(v)$.

Definition 45 (degenerate algorithm) An output algorithm k is called degenerate if the ratio $r_k(v) = \frac{g_k(v) - g_{opt}(v)}{|g_{opt}(v)| + 1}$ is unbounded. I.e. there exist v 's such that $r_k(v)$ is arbitrarily large.

A degenerate algorithm is arbitrarily far from optimal both additively and multiplicatively. Note that this should not be confused with the standard notion of an approximation ratio, as our definition corresponds to a single problem. In particular the number of agents is fixed.

Theorem 3.3.9 If a VCG-based mechanism for a CMAP is truthful then its output algorithm is either optimal or degenerate.

Proof: Let $m = (k, p)$ be a non-optimal truthful VCG-based mechanism for a CMAP. Like in theorem 3.3.2 assume that $p^i(w) = \sum_{j \neq i} w^j(k(w))$. Let v be a type (description) such that $k(v)$ is not optimal and let $y = opt(v)$ be an optimal output.

We define a type z by:

$$z_j^i = \begin{cases} v_j^i & \text{if } y_j^i = 1 \\ -\alpha & \text{otherwise} \end{cases}$$

where α is arbitrarily large.

Consider the type sequence:

$$\begin{aligned} v_0 &= (v^1, \dots, v^n) \\ v_1 &= (z^1, v^2, \dots, v^n) \\ &\vdots \\ v_n &= (z^1, \dots, z^n) \end{aligned}$$

Claim 3.3.10 *For all j , $y = \text{opt}(v_j)$.*

Proof: y is optimal for v_0 . From the independence condition, $g(v_j, y) = g(v_0, y)$. From the monotonicity, $g(v_j, x) \leq g(v_0, x)$ for all x . $\square$

Claim 3.3.11 $g(v_1, k(v_1)) < g(v_1, y)$

Proof: Otherwise $g(v_1, k(v_1)) = g(v_1, y) = g(v_0, y) > g(v_0, k(v_0))$. Therefore, when agent 1's type is v^1 and the declarations of the other agents are $v^2, \dots, v^n$, agent 1 is better off declaring z^1 . This contradicts the truthfulness of the mechanism. $\square$

Similarly, we obtain that $g(z, k(z)) < g(z, y) = g(v_0, y)$. By the forcing condition, $g(z, k(z)) \rightarrow -\infty$ when $\alpha \rightarrow \infty$. By that the theorem is proved. $\square$

When a CAMP is NP-Complete, an optimal algorithm is infeasible from a computational point of view. Thus, any computationally feasible truthful VCG-based mechanism for such a problem will produce unreasonable results. If we insist on solving problems under dominant strategies we are therefore in trouble. Another interesting class of mechanisms is one where the agents first declare their types, then calculate the output according to some protocol and finally calculate the payments according to the VCG formula. An immediate outcome of theorem 3.3.9 is that if such a mechanism has a non-optimal ex-post Nash equilibrium, then there are equilibria which are arbitrarily far from optimal. Finally we comment that we do not know how to prove such theorems for Bayesian models.

3.4 Computationally feasible VCG mechanisms

To date, VCG is the only known general method for the development of strategy-proof mechanisms. Therefore the results in the previous section do not leave much hope for the development of satisfying, computationally feasible strategy-proof mechanisms.

Our goal in this section is to develop feasible mechanisms that provide solution concepts of the same spirit as strategy-proofness.

Taking into consideration the computational limitations of the agents, we define the concept of feasible dominance. It is meant to be used in the context of revelation games. We assume that each of the agents chooses her action according to some *knowledge* she has at the beginning of the game. This knowledge may represent the computational limitations of the agent or some structural limitations on her information or behavior. We say that an action is feasibly dominant for the agent if she is not *aware* of *any* circumstances where another action is better for her. We argue that if an agent has such actions available, then it is irrational for her not to choose one of them.

When VCG-based mechanisms are carefully inspected one sees that the only reason for an agent to lie about her type is to help the algorithm to improve the overall result. This leads to the intuition that if the agents cannot improve upon the underlying algorithm then they can do no better than being truthful.

Given any algorithm for the corresponding optimization problem we define the second-chance mechanism based on it. This mechanism is a modification of the VCG-based mechanism where in addition to their type declarations, the agents are allowed to submit appeal functions as well. We show that under reasonable assumptions on the agents, truth-telling is feasibly dominant. When the agents act rationally and report their true types, the mechanism's result is *at least as good* as the underlying algorithm's. Our mechanism also satisfies participation constraints.

The reader is noted that our solution concept and the reasoning of why an agent should tell the truth to the mechanism are essentially different from previous works on bounded rationality. (For a recent book on this topic see [92].)

Subsection 3.4.1 describes the concept of feasibly dominant actions. The second-chance mechanism is introduced in subsection 3.4.2. In subsection 3.4.3 we describe why agents should tell the truth to our mechanism. Additional implementation issues are discussed in subsection 3.4.4. Our method is summarized in subsection 3.4.5.

3.4.1 Feasibly dominant actions

The basic models of equilibria in game theory are justified by the (hidden) assumption that the agents are capable of computing their best response functions.

In many natural games however, the action space is huge and this function is too complex to be calculated or even to be approximated within a reasonable amount of time. In such situations this assumption seems no longer valid.

In this section we re-formulate the concept of dominant actions (strategies) under the assump-

tion that agents have a limited capability of computing their best response. Our concept is meant to be used in the context of revelation games. We assume that each of the agents chooses her action according to some **knowledge** she has at the beginning of the game. We define an action to be feasibly dominant if, when the agent has chosen her action, she was not *aware* of *any* circumstances where another action is better for her.

Notations: We denote the action space of agent i by A^i . Given a tuple $a = (a^1, \dots, a^n)$ of actions chosen by the agents, we denote the utility of agent i by $u^i(a)$.

In a direct revelation mechanism (definition 36) $A^i = V^i$ and $u^i(w) = v^i(k(w)) + p^i(w)$. Note that the agent's type is suppressed.

Definition 46 (strategic knowledge) Strategic knowledge (or knowledge for short) of agent i is a partial function $b^i : A^{-i} \rightarrow A^i$.

Knowledge is a function by which the agent describes (for herself!) how she would like to respond to any given situation. The semantics of $a^i = b^i(a^{-i})$ is “when the others' actions are a^{-i} the best action I can think of is a^i ”.

Naturally we will assume that each agent is capable of computing her own knowledge and henceforth that b^i can be computed in a reasonable amount of time. (This assumption is formulated in section 3.4.4.)

Definition 47 (feasible best response) An action a^i for agent i is called feasible best response to a^{-i} if either a^{-i} is not in the domain of the agent's knowledge b^i or $u^i((b^i(a^{-i}), a^{-i})) \leq u^i(a)$.

In other words, other actions may be better against a^{-i} but at least when choosing her action the agent was not aware of these.

Note that when the agent's knowledge b^i is optimal, i.e. it is a best response function, this definition corresponds to the standard one. Note also that if a^{-i} is not in the domain of b^i , then any action a^i is a feasible best response to it.

The definition of feasibly dominant actions now follows naturally.

Definition 48 (feasibly dominant action) An action a^i for agent i is called feasibly dominant if it is a feasible best response against any a^{-i} . We also call such an action *fda*.

A dominant action is obviously feasibly dominant.

To exemplify the concept, consider a match between two chess-players, White and Black, where each of the opponents is required to submit a computer program that plays on her behalf. Black's

knowledge for example specifies what program she would like to submit if she knew White's. Of course one cannot expect such knowledge to be optimal. A program A_b is a feasible best response to A_w if Black does not know how to play better against A_w , more precisely, if when submitting A_b Black was not aware of any better way to play against A_w . Note that since this is a revelation game, we do not refer to knowledge that may be acquired during the game. A_b is feasibly dominant for Black if she cannot think of any change in the program that would help her against any possible opponent! We argue that if an agent has feasibly dominant actions available, then it is irrational not to choose one of them.

Our goal is to define a mechanism where truth-telling is feasibly dominant.

3.4.2 The second-chance mechanism

Although VCG-based mechanisms may not be truthful, it is not difficult to see that the only reason for an agent to lie about her type is to help the algorithm to improve the overall result. An agent who believes that the declarations of the others are such that she may better lie to the algorithm, might be wrong and henceforth cause damage to the overall result.

In order to prevent this we introduce the usage of appeal functions. The semantics of an appeal l is: “when the agents’ type is $v = (v^1, \dots, v^n)$ I believe that the output algorithm k produces a better result if it is given $l(v)$ instead of the actual input v ”. Given the agents’ declarations w , the mechanism then computes $k(w)$ and the results of all the appeals $k(l^1(w)), \dots, k(l^n(w))$ and selects the best output among these results.

The idea is that instead of declaring a falsified type, it is better for the agent to report it *truthfully* and to ask the mechanism to *check* whether the false declaration would have lead to better results. When the agents are truth-telling, the output chosen by the mechanism is at least as good as $k(v)$.

Definition 49 (appeal function) Let $V = \prod_i V^i$ denote the type space of the agents. An appeal is a function $l : V \rightarrow V$.

We now define the second-chance mechanism.

Definition 50 (second-chance mechanism) Given an output algorithm k , the second chance mechanism based on k is the following game:

1. Each agent sends a type declaration w^i and an appeal function l^i to the mechanism.
2. Let $w = (w^1, \dots, w^n)$. The mechanism computes $k(w), k(l^1(w)), \dots, k(l^n(w))$ and chooses among these outputs the one that maximizes the total welfare (according to w). I.e. the mechanism tries all the appeals and chooses the one that yields the best result.

3. Let $\hat{o}$ denote the chosen output. The mechanism then calculates the payments according to the VCG formula: $p^i = \sum_{j \neq i} w^j(\hat{o}) + h^i(w^{-i})$ (where h^i is any real function).

The actual implementation of the appeal functions is discussed in subsection 3.4.4. Note that $h^i(w^{-i})$ is independent of the agents' appeals. Until section 3.4.4 we can simply assume that it is always zero.

An action in a second chance mechanism is a pair (w^i, l^i) where w^i is a type declaration and l^i is an appeal function.

Definition 51 (truthful action) *An action $a^i = (w^i, l^i)$ is called truthful if $w^i = v^i$, i.e. the agent reveals her true type to the mechanism. A mechanism is called feasibly truthful if truth-telling is feasibly dominant for all the agents.*

The following observation is a key property of the mechanism.

Proposition 3.4.1 *Consider a second-chance mechanism with an output algorithm k . For every type $v = (v^1, \dots, v^n)$, if all agents are truth-telling, the output chosen by the mechanism is at least as good as $k(v)$.*

□

In other words, when the agents are truth-telling, the mechanism is at least as good as the underlying algorithm. In particular, when the algorithm is guaranteed to be within a certain constant from the optimal, then so is the mechanism.

We will show that under reasonable assumptions on the agents, the mechanism is indeed feasibly truthful. We also need to guarantee that all the computations can be carried out in a reasonable amount of time.

An example

To exemplify the strength of the mechanism consider Alice who participates in a combinatorial auction for houses. She is interested in a particular pair of adjacent cottages and is willing to pay up to \$200,000 for them. She is also willing to pay up to \$60,000 for each one separately.

Alice's type is therefore $v^i = \{200, 60, 60\}$ (in thousands, regarding only these particular two houses). After experimenting with the mechanism Alice notices that in many situations a declaration of $w^i = \{200, 0, 0\}$ causes the algorithm to produce a better result.

In a VCG-based mechanism Alice may choose to declare w^i instead of v^i . Sometimes, depending on the type declarations of the others, it would indeed improve the overall result and would therefore

increase her own profit as well. On other occasions it would reduce the quality of the overall result and therefore decrease Alice's profit.

In the second chance mechanism Alice can report her true valuation v^i and define $l^i(v^i, w^{-i})$ as (w^i, w^{-i}) . This way she can enjoy the best of both worlds: In cases where declaring w^i would improve the overall result, the mechanism will choose $k(l^i(v))$. In cases where lying to the algorithm would reduce the quality of the overall result, the mechanism will prefer $k(v)$, and thus prevent the damage.

3.4.3 Existence of computationally efficient feasibly dominant truth-telling actions

In this subsection we will show that under reasonable assumptions, computationally efficient feasibly dominant truth-telling actions do exist for the agents. Note that our assumptions are different from what is customary in game theory.

We first comment that if no time limit is enforced on the appeal functions then the agents have dominant actions available: Consider an agent i participating in a second chance mechanism. Let k denote the output algorithm. Given the type declarations w^{-i} of the other agents, there exists a type vector $\hat{w} \stackrel{df}{=} l^i(w^{-i})$ that maximizes $g((v^i, w^{-i}), k(\hat{w}))$. It is not difficult to see that (v^i, l^i) is dominant and henceforth feasibly dominant action for the agent. However, if the original problem is too hard to compute then so is l^i .

When the agents are limited to submit only computationally limited appeal functions, the existence of feasibly dominant truth-telling actions is less obvious. In this section we show that such actions are available under reasonable assumptions on the way that the agents' knowledge is *obtained*. We argue that in practice this knowledge will not "fall out of the blue", but there will be a few reasonable ways for the agents to obtain it.

We focus on two types of knowledge that look to us the most natural ones:

The first is obtained by an agent who simply explores the output algorithm. Such an agent ignores potential appeals for the other agents. Therefore knowledge of this kind is actually knowledge about the algorithm. The justification behind the assumption that agents will have this kind of knowledge is that the space of possible appeals is so huge and complex that it is *beyond the agents' capability* to reason about it.

The second type of knowledge is obtained by an agent who explores, in addition to the output algorithm, a small family of potential appeals for the other agents.

In both cases we show the existence of computationally efficient feasibly dominant truth-telling actions for the agent.

Clearly, when an agent lies about her type, there exist cases where she will consequently loose. Therefore the agents are further motivated to be truthful.

We begin with a formal definition for computationally limited algorithms and mechanisms.

Definition 52 (algorithm of degree d) *We say that an output algorithm is of degree d if its running time is bounded by some polynomial of degree d in the number of agents.*

The definition for appeal functions, actions, knowledge and mechanisms are similar.

Proposition 3.4.2 *If the output algorithm of a second-chance mechanism is of degree d then the mechanism is of degree $d + 1$.*

□

In subsection 3.4.4 we discuss ways to force the agents to submit only appeal functions of degree d .

We now refer to the first type of knowledge obtained by an agent who simply explores the output algorithm:

Definition 53 (appeal independent knowledge) *Knowledge b^i is called appeal independent if it is of the form $b^i : V^{-i} \rightarrow A^i$. Such knowledge is called declaration based if it is of the form $b^i : V^{-i} \rightarrow V^i$.*

The semantics of declaration based knowledge is: “when the others declare w^{-i} , I would like to declare $b^i(w^{-i})$ ”. Alice in our example has such kind of knowledge. The semantics of $(w^i, l^i) = b^i(w^{-i})$ where b^i is appeal independent is: “when the others declare w^{-i} , I would like to declare w^i and to submit the appeal l^i ”.

Theorem 3.4.3 *If b^i is a declaration based knowledge for agent i then (v^i, b^i) is feasibly dominant for the agent.*

Proof: Assume by contradiction that it is not feasibly dominant. Therefore there exists a vector of actions a^{-i} for the others such that $(w^i, \phi) \stackrel{df}{=} b^i(a^{-i})$ is strictly better for the agent than playing (v^i, b^i) . Note that i ’s appeal is always empty. Since b^i is declaration based, we can assume that all the appeals of the other agents are also empty. Denote a^{-i} by (w^{-i}, ϕ) . Consider the case where the agent’s action is (v^i, b^i) . If the mechanism “rejects the appeal” and chooses the type (v^i, w^{-i}) then $g((v^i, w^{-i}), k(v^i, w^{-i})) \geq g((v^i, w^{-i}), k(w^i, w^{-i}))$. By declaring w^i the agent forces the mechanism to choose $k(w^i, w^{-i})$ which can only reduce the utility of the agent. If the mechanism “accepts the appeal” and chooses the type (w^i, w^{-i}) , then it does the same when the agent declares w^i . □

Theorem 3.4.4 *If the output algorithm is of degree d and the agent has appeal independent knowledge of degree d then there exists a truthful feasibly dominant action of degree d for this agent.*

Proof: Let b^i be a knowledge of degree d for agent i . Define an appeal l^i as follows: Given w^{-i} , let $(w^i, \psi^i) = b^i(w^{-i})$. The appeal computes $k(w^i, w^{-i})$ and $k(\psi((w^i, w^{-i})))$ and takes the better result according to (v^i, w^{-i}) . Obviously l^i is feasibly dominant and of degree d $\square$

We now consider a more general scenario, where the agent explores, in addition to the output algorithm, a small family of potential appeals for the other agents.

Definition 54 (d -obtainable knowledge) *Knowledge b^i is called d -obtainable if the following holds:*

1. b^i is of degree d .
2. Every appeal function that appears, in the domain or in the range of b^i , is of degree d .
3. There are at most n^d appeal functions that appear in the domain or in the range of b^i . Moreover there exists a representative family L^i of no more than n^d $(n-1)$ -tuples of appeals such that for every tuple ϕ^{-i} that appears in the domain of b^i there exists a $\psi^{-i} \in L^i$ such that for all w^{-i} , $b^i((w^{-i}, \phi^{-i})) = b^i((w^{-i}, \psi^{-i}))$.

The second condition is justified by the assumption that an agent can not reason about functions that she cannot compute. The representative family in the third condition stands for all the cases that the agent has checked. The underlying assumption is that an agent cannot reason about functions and tuples that she has not checked at least once. Naturally we assume that an agent can check only a reasonable number of cases.

Theorem 3.4.5 *If the output algorithm is of degree d and an agent has a d -obtainable knowledge, then she has a truthful feasibly dominant action of degree $3 \cdot d$.*

Proof: Let b^i be a d -obtainable knowledge for agent i . Given w^{-i} we shall define an appeal l^i as follows: Let L be the family of all appeals that appear in the domain or in the range of b^i . Let L^i be a representative family and let $W = \{w^i | \exists \psi^{-i} \in L^i, \phi^i \text{ s.t. } (w^i, \phi^i) = b^i((w^{-i}, \psi^{-i}))\}$. Obviously $|W|, |L|$ are bounded by a polynomial of degree d . l^i computes for every pair $(w^i \in W, l \in L)$, $k(l((w^i, w^{-i})))$ and takes the best output according to (v^i, w^{-i}) . It is not difficult to see that (v^i, l^i) is feasibly dominant and that l^i is of degree $3 \cdot d$. $\square$

When the agents' knowledge is obtained in one of the ways mentioned above then if the time limit enforced on the appeals is not too small, the agents are strongly motivated to report their true types to the mechanism. We argue that this will be the case for many applications of interest.

Truthful feasibly dominant actions may not exist in situations where an agent has significant knowledge about properties of the potential appeals of the others. Such situations may arise in repeated settings (e.g an auction that runs every week). This may be solved by merging each appeal that improves the result of the output algorithm into it and letting the agents update their appeal functions.

3.4.4 Other implementation issues

Implementation of the appeal functions

We see two natural ways to implement the concept of appeal functions. The first is simply to let the agents compute their appeals by themselves and to send the results to the mechanism in a second round. In this method however, the type of each agent is revealed to all the others. This is undesirable for many applications. Another natural way is to supply the agents with a language for the description of their appeals and to perform the computation on the mechanism's machine.

In both methods it is not difficult to impose a time limit on the agents or to ask them to pay for additional computational time. However an arbitrary time limit may prevent the existence of feasible dominant actions.

An interesting direction is to impose on the description of the appeals a structure that reflects the agents' knowledge of the output algorithm. One possible structure is a decision tree where the agents are required to supply, for each leaf α , a type vector t_α such that the algorithm's result is strictly improved when it is given $l(t_\alpha)$ instead of the actual input t_α . In this method the computational time of the mechanism is naturally limited by the computational power of the agents themselves.

Participation constraints

One desirable property of mechanisms is that the utility of a truthful agent is always non-negative. This is often called participation constraints (see e.g. [60]). In this section we construct a second-chance mechanism that satisfies this property.

We shall assume that for each agent i , there exists a type $\underline{v}^i$ such that for every $v = (v^1, \dots, v^n)$, $g_{opt}((\underline{v}^i, v^{-i})) \leq g_{opt}(v)$. In a combinatorial auction for example, this type is defined by $\underline{v}^i(s) = 0$ for every combination s of items.

The Clarke mechanism ([8]) is a VCG mechanism with payments $p^i(w) = \sum_{j \neq i} w^j(opt(w)) -$

$g_{opt}(\underline{v}^i, (w^{-i}))$. It is not difficult to check that the mechanism satisfies participation constraints. When the optimal algorithm in the Clarke mechanism is replaced by a sub-optimal one, the result of the algorithm may be improved when w^i is replaced by $\underline{v}^i$. Therefore participation constraints may no longer be satisfied. However this is not difficult to fix.

Definition 55 *We say that an output algorithm k satisfies participation constraints if for every type $v = (v^1, \dots, v^n)$ and agent i , $g_k(v) \geq g_k(\underline{v}^i, v^{-i})$.*

Proposition 3.4.6 *If the output algorithm k is of degree d , then there exists an algorithm $\tilde{k}$ of degree $d + 2$ that satisfies participation constraints such that for every type v , $g_{\tilde{k}}(v) \geq g_k(v)$.*

Proof: Given v we define $\tilde{k}(v)$ as follows. At the beginning we let $k_0 = k$. In the $(j + 1)$ th phase, if for every agent i , $g_{k_j}(v) \geq g_{k_j}(\underline{v}^i, v^{-i})$ then fix $\tilde{k}(v) = k_j(v)$ and stop; Otherwise fix the type of one of the agents i who does not satisfy the above condition to be $\underline{v}^i$ and continue. It is not difficult to verify that $\tilde{k}$ satisfies the required conditions. $\square$

Given an algorithm $\tilde{k}$ that satisfies participation constraints we shall define the payment function of the second chance mechanism based on it as $p^i = \sum_{j \neq i} w^j(o) - g_{\tilde{k}}(\underline{v}^i, (w^{-i}))$ where w denotes the agents' declarations and o the chosen output. Since $g(w, o) \geq g_{\tilde{k}}(w)$, participation constraints are satisfied.

3.4.5 Summary of the method

We summarize the necessary steps the designer of a mechanism needs to take when using our method. First the designer needs to develop an algorithm k for the corresponding optimization problem. This could be done without any constraints on the global behavior of the algorithm. The designer can automatically transform k into an algorithm $\tilde{k}$ that satisfies participation constraints. The next step is to find a reasonable time limit on the appeals or better to develop a language (API) for the appeals that reflects the agents' knowledge of the algorithm. After this is done the designer can give the mechanism to the agents to experiment with and to learn $\tilde{k}$. Finally, after the participants are ready, the second-chance mechanism could be executed. In repeated settings the designer may want to merge (automatically) successful appeals into $\tilde{k}$ and let the agents update their appeal functions afterwards. Note that all the steps required for turning the algorithm k into a mechanism are either automatic or common for all mechanism design problems. Using our method, the development of a mechanism for a mechanism design problem is therefore reduced to the development of a good algorithm for the corresponding optimization problem.

Acknowledgments: We thank Abraham Newman and Motty Perry for helpful discussions at various stages of this work. We thank Ron Lavi, Ahuva Mu'alem, Elan Pavlov and Inbal Ronen for comments on earlier drafts of this paper.

Chapter 4

Discussion

4.1 Conclusions and further research

This thesis studies algorithmic problems that involve self-interested agents. It assumes that the agents are rationally following their *own* self interest. It studies such problems in the framework of mechanism design theory and focuses on dominant strategy implementation.

The thesis shows that many problems that naturally stem from computer science applications can be satisfactorily handled. The suggested solution, which is a combination of the *compensation and bonus* and the *second chance* mechanisms (definitions 24, 50), let the designer focus on the algorithmic side of the problem and constructs the protocol automatically. The protocol uses monetary transformations in order to motivate the agents to behave well. Although revenues and payments are not optimized in this work, the protocol never pays more than the designer's budget. Results in this thesis can roughly be divided into the following categories: Limitations of the classical model, probabilistic mechanisms, the usage of verification assumptions, the computational limitations of existing mechanisms and the second-chance mechanism.

Beyond the concrete results, there are several principal insights that emerge from this work:

The importance of information The information, a-priory or post-priory, which is available to the mechanism can make a crucial difference on the mechanism's ability to handle such problems. In particular, the importance of verification assumptions was shown in this thesis.

The need for the relaxation of the game theoretic requirements Requirements like the truthfulness of the mechanism, seem to impose constraints on the global behavior of the corresponding algorithms, which are very difficult and sometimes even impossible to satisfy. In many cases, a slight *relaxation* of such requirements can greatly facilitate the construction of computationally feasible mechanisms. This phenomenon is likely to occur when considering other game theoretic requirements such as budget balance.

The power of randomization Randomization can play a major role in the development of efficient mechanisms. Its contribution may be twofold. First, randomized algorithms are in many cases simpler, faster and more efficient than deterministic ones. This was already exploited in this work. Secondly, randomization of the mechanism can greatly influence the strategy space of the agents precluding strategies which are bad for the designer. This aspect is yet to be explored.

This work is only a first step towards an understanding of the ability to solve computational problems that involve selfish agents. There are clearly numerous directions for further research on this topic. I mention only few basic ones: The computational power of strategy-proof mechanisms is far from being understood, even in the classical model, probabilistic models and further relaxation of the concept of dominant strategies are yet to be explored and more concrete connections between mechanism design and computational complexity are yet to be discovered. Moreover, in practice, mechanisms will have to handle additional problems like irrationality and collusion among agents. Issues like revenue maximization, risk, reserved prices and budget balance are very important to many applications. In a lot of cases, mechanisms will have to be carried out in a "real" distributed environment. Finally, solution concepts other than dominant strategy implementation (e.g. Bayesian Nash) may be considered.

I am certain that all these issues and a lot of others will provide researchers with many beautiful research problems in the years to follow!

4.2 Applications

Problems that involve self interested participants arise from several sub fields of computer science and economics. In this section I describe some example applications which may benefit from results obtained in this work and make some additional comments about how to use these results.

Electronic Commerce: Combinatorial Auctions

The problem of combinatorial auctions (section 3.2.4) can be addressed by a second-chance mechanism to maximize the total welfare. Besides the major obstacle of the computation of the allocation, this mechanism can also be used for handling another major problem: that even the description of the valuation functions is intractable. In the proposed setting the agents will send two structures that represents their valuation functions. One is an oracle $v^i(\cdot)$ that accepts a subset s and returns the agent's valuation $v^i(s)$. The other structure $x^i(\cdot)$ represents the valuation function in a way that is convenient for the allocation algorithm to work with (e.g. a short list of bids or a description in an intermediate language [75]). The second chance mechanism will use $x^i(\cdot)$ for

the computation of the allocation and will query $v^i(.)$ for the computation of the total welfare. The agents will be requested to prepare the representing structure according to the designer's instructions (e.g. submit a list of the 100 most valuable sets which are not contained one in each other). An agent's appeals will refer to the $x^i(.)$ s but be measured according to the $v^i(.)$ s. In any case where an agent consider a deviation from the instructions, she can rather follow them and ask the mechanism to check whether she is better off deviating.

Networking: broadcasting long messages

Consider a network where the links are privately owned and the costs of sending messages are privately known to their owners. Consider the problem of broadcasting (e.g. movies) in such a network that appeared at section 3.3.2. The a second-chance mechanism could be used here as well. The companies that owe the links can weekly update their costs and appeals. Successful appeals may be incorporated into the mechanism as suggested in section 3.4.3.

Networking: broadcasting many short messages

Consider such a network as before. This time, whenever someone sends a message we would like to send it through the network. Obviously, there is no point in applying a central mechanism that finds an optimal route for each message. However, a solution that makes local decisions in every node which are based on a Vickery auction with random weights (as in section 2.4.4) may be considered. The randomness here is used to spread the load over the system. Although such a mechanism is not necessarily truthful it may has some interesting stochastic "strategy proofness" properties.

Virtual communities

My initial practical motivation when starting this research was to enhance cooperation among people over the Internet. Consider for example a virtual community where people cooperate and help each other in solving problems (as an example consider [37] or just a good old news-group). A naive incentive model (such as a new-group's) has two potential major problems: first there may not be enough motivation for the participants to help each other and second, there is nothing that guarantees the efficiency of the allocation of problems to participants. My personal feeling is that indeed, news-group participants hardly devote a considerable amount of time for problems which are posed in news-groups and that the amount of answers for various requests is im-balanced and inefficient. For some communities the compensation and bonus mechanism might help. Money, real or virtual, may motivate people to invest more time helping each-other and the mechanism will make sure that problems are efficiently allocated among the participants. Although verification of the costs does not seem natural for such an application, it may simply be ignored. Many additional

interesting questions can be asked about such an application.

Privatization processes

Several privatization processes in recent years were conducted using various sorts of auctions (e.g. the California day-ahead auction [119]). As the verification of both costs and actions seems feasible for such applications, the compensation and bonus mechanism might offer an alternative to the auction approach. The mechanism enables a government to determine a precise *policy* of what is desired for the public and to pay the companies involved according to their *contribution* to the public's welfare.

Bibliography

- [1] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proc. of the 37th Annual Symposium on Foundations of Computer Science*, 1996.
- [2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science*, pages 24–27, 1992.
- [3] K. J. Arrow. *Social choice and individual values*. New York: Wiley, 1963.
- [4] R. J. Auman and Hart S., editors. *Handbook of Game Theory*. Elsevier Science B.V., 1995.
- [5] S. Barbera and Peleg B. Strategy-proof voting schemes with continuous preferences. *Social choice and welfare*, 7:31–38, 1990.
- [6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, 1988.
- [7] B. Braden, D. Clark, Crowcroft. J., B. Davie, S. Deering, Estrin D., Floyd S., Jacobson V., Minshall G., PartridgeC., Peterson L., Ramakrishnan K., Shenker S., Wroclawski J., and Zhang L. Recommendations on queue management and congestion avoidance in the internet. Request for Comments (Informational) 2309, Internet Engineering Task Force, 1998.
- [8] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, pages 17–33, 1971.
- [9] S. A Cook. The complexity of theorem proving procedures. In *Proc. of 3rd Ann ACM symposium on theory of computing*, pages 70–72, 1970.
- [10] Peter Cramton. The fcc spectrum auction: an early assessment. *Journal of Economics and Management Strategy*, 6:431–495, 1997.
- [11] A. Demers, S. Keshav, and Shenker S. Analysis and simulation of a fair queuing algorithm. *Journal of Internetworking*, pages 3–20, 1990.

- [12] Altman E. Flow control using the theory of zero sum markov games. *IEEE transactions on automatic control*, 39:814–818, 1994.
- [13] Zermelo E. Uber eine anwendung der mengnlehre auf die theorie des schachspiels. In *Proc. of the fifth international congress of mathematicians*, pages 501–504, 1913.
- [14] *Proceedings of the first ACM conference on electronic commerce (EC99)*, 1999.
- [15] Bernardo Huberman (ed.). *The Ecology of Computation*. Elsevier Science Publishers/North-Holland, 1988.
- [16] Eithan Ephrati. *Planning and consensus among autonomous agents*. PhD thesis, Hebrew University of Jerusalem, Departement of Computer Science, 1993.
- [17] Eithan Ephrati, Sarit Kraus, and Daniel Lehmann. Negotiatio in a non-cooperative environment. *of Experimental and Theoretical Artificial Intelligence*, 3:255–282, 1991.
- [18] Eithan Ephrati and Jeffrey S. Rosenschein. The clarke tax as a concensus mechanism among automated agents. In *Proceedings of the national Conference on Artificial Intelligence*, pages 173–178, July 1991.
- [19] Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. In *Thirty-Second Annual ACM Symposium on Theory of Computing (STOC00)*, May 2000.
- [20] Donald F. Ferguson, Christos Nikolaou, and Yechiam Yemini. Economic models for allocating resources in computer systems. In Scott Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1995.
- [21] Eric Friedman and Scott Shenker. Learning and implementation on the internet. To appear.
- [22] Drew Fudenberg and Jean Tirole. *Game Theory*. Mit Press, 1991.
- [23] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI-99*, 1999.
- [24] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [25] Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, 41:587–601, 1973.

- [26] Oded Goldreich. *The Foundations of Cryptography*. Springer Verlag, 2000. Forthcoming.
- [27] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.
- [28] Shafi Goldwasser, Silvio Micali, and Andy Yao. Strong signature schemes. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 25–27, 1983.
- [29] Rica Gonen and Daniel Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. To appear.
- [30] J. Green and J.J. Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, pages 427–438, 1977.
- [31] T. Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.
- [32] J. C. Harsanyi. Games with incomplete information played by 'bayesian' players, parts i ii and iii. *Management science*, 14, 1967-68.
- [33] R. M. Harstad, Rothkopf M. H., and Pekec A. Computationally manageable combinatorial auctions. Technical Report 95-09, DIMACS, Rutgers university, 1995.
- [34] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *37th Annual Symposium on Foundations of Computer Science*, 1996.
- [35] Dorit S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS publishing company, 1997.
- [36] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the association for computing machinery*, pages 317–327, 1976.
- [37] HotDispatch. Available via <http://www.hotdispatch.com/sun>.
- [38] M.-T. T Hsiao and Lazar A. A. Optimal decentralized flow control. *Performance Evaluation*, 13:181–204, 1991.
- [39] Brnardo A. Huberman and Tad Hogg. Distributed computation as an economic system. *Journal of Economic Perspectives*, pages 141–152, 1995.
- [40] *Proceedings of The first international conference on information and computation economies (ICE-98)*, 1998.

- [41] Mirrlees J. An exploration in the theory of optimal income taxation. *Review of economic studies*, 38:175–208, 1971.
- [42] Viggo Kann and Pierluigi Crescenzi. A compendium of np optimization problems. Available via <http://www.nada.kth.se/~viggo/problemelist/compendium.html>.
- [43] Paul Klemperer. Auction theory: a guide to the literature. *Journal of economic surveys*, pages 227–286, 1999.
- [44] Y. Korilis and A. Lazar. On the existence of equilibria in noncooperative optimal flow control. *Journal of the ACM*, pages 584–613, 1995.
- [45] Y.A Korilis, A. A. Lazar, and A. Orda. Architecting noncooperative networks. *IEEE Journal on Selected Areas in Communication (Special Issue on Advances in the Fundamentals of Networking)*, 13(7):1241–1251, September 1991.
- [46] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *STACS 99, the 16th Annual Symposium on Theoretical Aspects of Computer Science*, March 1999.
- [47] Sarit Kraus. Agents contracting tasks in non-collaborative environments. In *Eleventh National Conference on Artificial Intelligence*, pages 243–248, 1993.
- [48] Sarit Kraus. *Strategic Negotiation in Multi-Agent Environments*,. MIT Press,, 2001. To appear.
- [49] Leslie Lamport, Robert E. Shostak, and C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, 1982.
- [50] Ron Lavi and Noam Nisan. Competitive analysis of online auctions. To appear.
- [51] A.A. Lazar and N. Semret. The progressive second price auction mechanism for network resource sharing. In *8th International Symposium on Dynamic Games*, Maastricht, The Netherlands, July 1998.
- [52] Daniel Lehmann. Private communication, 1999.
- [53] Daniel Lehmann, Liadan O’Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *ACM Conference on Electronic Commerce (EC-99)*, pages 96–102, November 1999.

- [54] Jan Karel Lenstra, David B. Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science*, pages 217–224. IEEE, 1987.
- [55] L. Levin. Universal sorting problems. *Problems of information transmission*, 9:265–266, 1973. Translated from Russian.
- [56] Kevin Leyton-Brown, Moshe Tennenholtz, and Yoav Shoham. Bidding clubs: Institutionalized collusion in auctions. To appear.
- [57] Nathan Lineal. Game theoretic aspects of computing. In *Handbook of Game Theory*, volume 2, pages 1339–1395. Elsevier Science Publishers B.V, 1994.
- [58] Rajan M. Lukose and Bernardo A. Huberman. Surfing as a real option. In *The first international conference on information and computation economies (ICE-98)*, pages 45–51, 1998.
- [59] Nancy Lynch. *Distributed Algorithms (Data Management Series)*. Morgan Kaufmann, 1997.
- [60] A. Mas-Collel, Whinston W., and J. Green. *Microeconomic Theory*. Oxford university press, 1995.
- [61] Eric Maskin. Private communication, 2000.
- [62] J. McMillan. Selling spectrum rights. *Journal of Economic Perspectives*, pages 145–162, 1994.
- [63] Nimrod Megiddo and Avi Wigderson. On play by means of computing machines. In *1st Conf. on Theoretical aspects of reasoning about knowledge*, 1986.
- [64] Welleman Michael. Market-oriented programming. Web Page: <http://ai.eecs.umich.edu/people/welman/MOP.html>.
- [65] P. Milgrom. *Auction Theory for Privatization*. Cambridge university press, 1998.
- [66] Paul Milgrom. Putting auction theory to work: the simultaneous ascending auction. Technical Report 98-002, Dept. of Economics, Stanford University, 1998.
- [67] Market design inc. Web Page: <http://www.market-design.com>.
- [68] Dov Monderer and Moshe Tennenholtz. Distributed games. To appear in *Games and Economic Behaviour*.

- [69] Dov Monderer and Moshe Tennenholtz. Optimal auctions revisited. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.
- [70] Dov Monderer and Moshe Tennenholtz. Distributed games: From mechanisms to protocols. In *Proceedings of AAAI-99*, 1999.
- [71] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge Univ Press, 1995.
- [72] Miller M.S. and Drexler K. E. *The Ecology of Computation*, chapter Markets and Computation: Agoric open systems. North Hollan, 1988.
- [73] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conf. on Electronic Commerce (EC99)*, pages 129–138, 1999.
- [74] J. F. Nash. Equilibrium points in n-person games. In *Proc. of the national academy of sciences of the united states of america*, pages 48–49, 1950.
- [75] Noam Nisan. Bidding and allocation in combinatorial auctions. To appear.
- [76] Noam Nisan. Algorithms for selfish agents. In *To appear in Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, Trier, Germany, March 1999.
- [77] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *The Thirty First Annual ACM symposium on Theory of Computing (STOC99)*, pages 129–140, May 1999.
- [78] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 2000. To appear.
- [79] Noam Nisan and Amir Ronen. Computationally feasible vcg mechanisms. In *Proceedings of the Second ACM Conference on Electronic Commerce (EC00)*, 2000. To appear.
- [80] Andrew Odlyzko. Paris metro pricing for the internet. In *ACM conference on electronic commerce (EC99)*, pages 140–147, 1999.
- [81] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT press, 1994.
- [82] T. R. Palfrey. *Handbook of Game Theory*, chapter Implementation Theory. Elsevier Science B.V., 1995.
- [83] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing, 1994.

- [84] Christos H. Papadimitriou. Computational aspects of organization theory. In *Proceedings of the 1996 European Symposium on Algorithms*. Springer LNCS, 1996.
- [85] Christos H. Papadimitriou and Mihalis Yannakakis. On the value of information in distributed decision-making (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 61–64, Montreal, Quebec, Canada, August 1991.
- [86] Christos H. Papadimitriou and Mihalis Yannakakis. Linear programming without the matrix (extended abstract). In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 121–129, San Diego, California, May 1993. ACM.
- [87] Kihong Park, Meera Sithram, and Shaogang , Chen. Quality of service provision in noncooperative networks: hetogenous preferences, multi-dimentional qos vectors and burstiness. In *The first international conference on information and computation economies (ICE-98)*, pages 111–125, 1998.
- [88] R. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Comm. ACM*, pages 120–126, 1978.
- [89] Kevin Roberts. The characterization of implementable choise rules. In Jean-Jacques Laffont, editor, *Aggregation and Revelation of Preferences*, pages 321–349. North-Holland, 1979. Papers presented at the first European Summer Workshop of the Econometric Society.
- [90] Amir Ronen. Algorithms for rational agents. In *Proc. of the 27th Annual Conference on Current Trends in Theory and Practice of Informatics*, 2000.
- [91] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, 1994.
- [92] Ariel Rubinstein. *Modeling Bounded Rationality*. MIT Press, 1998.
- [93] J. Sairamesh and Y. Ferguson, D.and Yemini. An approach to pricing, optimal allocation and quality of service provisioning in high-speed networks. In *IEEE INFOCOM95*, pages 1111–1119, 1995.
- [94] Tuomas W. Sandholm. Limitations of the vickrey auction in computational multiagent systems. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, pages 299–306, Keihanna Plaza, Kyoto, Japan, December 1996.

- [95] Tuomas W. Sandholm. Approaches to winner determination in combinatorial auctions. *Decision Support Systems*, to appear.
- [96] Thoumas Sandhulm. An implementation of the contract net protocol based on marginal calculations. In *Eleventh National Conference on Artificial Intelligence*, pages 256–262, 1993.
- [97] Mark Allen Satterthwaite. Strategy-proofness and arrow’s condition: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, pages 187–217, 1975.
- [98] T. J Schaffer. Complexity of some two person perfect information games. *J. of Computer systems*, pages 185–225, 1978.
- [99] S. Shenkar, Clark D. E., and Hertzog S. Pricing in computer networks: Reshaping the research agenda. *ACM Computational Comm. Review*, pages 19–43, 1996.
- [100] S. Shenker, D. Estrin D. Clarck, and Herzog S. Pricing in computer networks: reshaping the research agenda. *Telecommunication Policy*, pages 183–201, 1996.
- [101] Scott Shenker. Making greed work in networks: a game theoretic analysis of switch service disciplines. In *ACM SIGCOMM94*, pages 47–57, 1994.
- [102] Yoav Shoham and Tennenholtz Moshe. Rational computation and the coomunication complexity of auctions. To appear.
- [103] Yoav Shoham and Katsumi Tanaka. A dynamic theory of incentives in multi-agent systems (preliminary report). In *Proceedings of the Fifteenth International Joint Conferences on Artificial Intelligence*, pages 626–631, August 1997.
- [104] Michael Sipser. *Introduction to the Theory of Computation*. Pws Pub Co, 1996.
- [105] Reid G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, pages 1104–1113, 1980.
- [106] Joel Spencer. Randomization, derandomization and antirandomization: three games. *Theoretical Computer Science*, 131:415–429, 1994.
- [107] *Proceedings of the Twenty-Sixth annual ACM symposium on the theory of computing (STOC94)*, 1994.
- [108] K. Sycara. Resolving goal conflicts via negotiation. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 245–250, 1988.

- [109] Moshe Tennenholtz. Some tractable combinatorial auctions. In *the national conference on artificial intelligence (AAAI-2000)*, 2000.
- [110] Jean Van Leeuwen, editor. *Handbook of Theoretical Computer Science*. Mit Press, 1990.
- [111] Hal R. Varian. The economics of the internet, information goods intellectual property and related issues. To Appear.
- [112] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, pages 8–37, 1961.
- [113] J. von Neumann and Morgenstern O. *The theory of games and economic behaviour*. John Weily and Sons, 1944.
- [114] Nir Vulkan and Ken Binmore. Applying game theory to automated negotiation. To appear.
- [115] W. Walsh and M. Wellman. Efficiency and equilibrium in task allocation economies with hierarchical dependencies. In *In Sixteenth International Joint Conference on Artificial Intelligence*, pages 520–526, 1999.
- [116] W.E. Walsh and M.P. Wellman. A market protocol for decentralized task allocation: Extended version. In *The Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, 1998.
- [117] W.E. Walsh, M.P. Wellman, P.R. Wurman, and J.K. MacKie-Mason. Auction protocols for decentralized scheduling. In *Proceedings of The Eighteenth International Conference on Distributed Computing Systems (ICDCS-98)*, Amsterdam, The Netherlands, 1998.
- [118] Michael Wellman. A general-equilibrium approach to distributed transportation planning. In *AAAI92*, 1992.
- [119] Robert Wilson. Activity rules for the power exchange, report to the california trust for power industry restructuring. Available at <http://www.market-design.com>, 1997.
- [120] Gilad Zlotin. *Mechanisms for Automated Negotiation among Autonomous Agents*. PhD thesis, Hebrew University of Jerusalem, Departement of Computer Science, 1994.

Appendix A

A note about single object allocations

This section gives an example for a natural problem where the family of functions that can be extended into a truthful mechanism is very large. This is in a sharp contrast to [89].

One of the most basic problems of mechanism design is an auction on a single object. Formally:

Definition 56 (single object allocation) *A single object allocation problem is a mechanism design problem of the following form:*

- *There are $n + 1$ possible outputs. An output $i > 0$ means that the object is allocated to agent i . 0 means that the object is not allocated to any agent.*
- *The type of each agent is any strictly positive number.*

We say that a mechanism for this problem is *good* if it is truthful, satisfies participation constraints (definition 3.4.4) and is normalized such that the payment of the agents who do not get the object is always zero.

We shall characterize the family of good mechanisms for this problem.

Definition 57 (proper function) *An output function $x : R_{++}^n \rightarrow \{0, \dots, n\}$ is called proper if for every type vector $t = (t^1, \dots, t^n)$, and an agent i such that $i = x(t)$, if $s^i \geq t^i$ then $i = x((s^i, t^{-i}))$.*

In other words, an agent wins the object with type t^i shall keep if her type increases.

Definition 58 (implied payment) *Let $x(\cdot)$ be a proper function. We say that a payment function $p(\cdot)$ is implied by $x(\cdot)$, if $-p^i(t)$ is 0 when $i \neq x(t)$ or equals $\inf\{s^i | i = x((s^i, t^{-i}))\}$.*

In other words, agent i , when wins, pays an amount which equals the minimal type that enables her to still win the object, or zero otherwise. It is not difficult to see that the infimum exists.

Proposition A.0.1 *If $m = (x, p)$ is a revelation mechanism for the single object allocation problem (definition 56), such that $x(\cdot)$ is a proper function and $p(\cdot)$ is implied by $x(\cdot)$, then m is a good mechanism.*

Proof: Let t be a type vector, i an agent. Clearly $p^i(\cdot)$ does not depend on t^i . If $i \neq x(t)$ then, since $x(\cdot)$ is proper and from the definition of $p(\cdot)$ it must be that $t^i \leq p^i$. Henceforth i can only loose if she wins by cheating. If $i = x(t)$ then from the definition of p , $t^i \geq p^i$ and therefore i can only loose by cheating. $\square$

Proposition A.0.2 *If a revelation mechanism $m = (x, p)$ is good then x is proper and p is implied by x .*

Proof: From the basic properties proven in section 2.4.3 we known that the mechanism should offer two possible payments to the agent: A payment which equals zero if she loses (this is because the mechanism is good), or a payment of p^i if she wins. We know (proposition 2.4.4) that p^i is independent of t^i . From the maximization property (proposition 2.4.5) we also know that if $t^i > p^i$ then i must win the object and if $t^i < p^i$ then she must not win. Clearly this implies that x is proper and that p^i is implied by it. $\square$

We have therefore characterized the family of good mechanisms for this problem. A similar result holds for a discrete setting.

It is not difficult to see that if $x(\cdot)$ obeys the following natural independence condition:

For every type vector t , an agent i and a type s^i , if $x(t) \neq i$ and $x((s^i, t^{-i})) \neq i$ then $x(t) = x((s^i, t^{-i}))$.

then it can be represented as $x(t) = \arg \max_i \psi^i(t^i)$ where $\psi^i(\cdot)$ are monotonically increasing. Obviously, every function of this form obeys the independence condition.

It is not hard to construct an example which does not satisfy this independence condition and henceforth can not be represented as a maximum of monotone functions. Thus, in a contrast to problems where only maximizing weighted sums is implementable [89], the family of truthful mechanisms for this problem is very large.