Algorithms for Selfish Agents

Mechanism Design for Distributed Computation

Noam Nisan*

Abstract

This paper considers algorithmic problems in a distributed setting where the participants cannot be assumed to follow the algorithm but rather their own self-interest. Such scenarios arise, in particular, when computers or users aim to cooperate or trade over the Internet. 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.

This exposition presents a model to formally study such algorithms. This model, based on the field of mechanism design, is taken from the author's joint work with Amir Ronen, and is similar to approaches taken in the distributed AI community in recent years. Using this model, we demonstrate how some of the techniques of mechanism design can be applied towards distributed computation problems. We then exhibit some issues that arise in distributed computation which require going beyond the existing theory of mechanism design.

1 Introduction

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.

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 – act "selfishly". We cannot simply expect each computer on the Internet to faithfully follow the designed protocols or algorithms. It is more

*Institute of Computer Science, Hebrew U., Jerusalem and IDC, Herzliya. This research was supported by grants from the Israeli ministry of Science and the Israeli academy of sciences.

reasonable to expect that each selfish computer will try to manipulate it for its owners' benefit. An algorithm or protocol intended for selfish computers must therefore be designed in advance for this kind of behavior!

Such protocols and algorithms will likely involve payments (or other trade) between the selfish participants. One can view this challenge (of designing protocols and algorithms for selfish computers) as that of designing automated trade rules for the Internet environment. The normal practices of human trade, while clearly relevant, cannot be directly applied due to the much greater complexity involved and due to the automated nature of the trade.

The view taken in this paper is that of a systems' engineer that has certain technical goals for the global behavior of the Internet. We view the selfishness of the participants as an obstacle to our goals, and we view the trade and payments involved as a way to overcome this obstacle. In economic terms, we desire a virtual "managed economy" of all Internet resources, but due to the selfishness of the participants we are forced to obtain it using the "invisible hand" of "free markets". Our goal is to design the market rules as to ensure the desired global behavior.

We first present a formal model that allows studying these types of issues. The model relies on the rationality of the participants and is game-theoretic in nature. Specifically, it is based upon the theory of mechanism design. The model is directly taken from the author's joint work with Amir Ronen [18], and is similar in spirit to some models studied in the distributed AI community. After presenting the model we present some of the basic notions and results from mechanism design in our distributed computation setting. We do not intend here to give a balanced or exhaustive survey of mechanism design, but rather to pick and choose the notions that we feel are most applicable to our applications in distributed computation. Finally, we present some scenarios that arise in distributed computation that require going beyond the existing theory of mechanism design.

Before getting into the model, we will mention some of the application areas we have in mind, and shortly mention some of the existing work in computer science along this and similar tracks.

2 Sample Scenarios

We shortly sketch below three (somewhat related) application areas that we feel require these types "selfish algorithms". These application areas are each quite wide in their scope, involve complicated optimizations of resources, and directly involve differing goals of the participants. Most of the works cited below lie in one of these areas.

2.1 Resource allocation

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".

2.2 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.

2.3 Electronic Trade

Much trade is taking place on the Internet and much more is likely to take place on it. Such trade may include various financial goods (stocks, currency exchange, options), various information goods (video-on-demand, database access, music), many services (help desk, flower delivery, data storage), as well as real goods (books, groceries, computers) . This trade will likely involve sophisticated programs communicating with each other trying to find "the best deal". In addition, this will also raise the possibility of various brokerage services such as information providers, aggregators, and other types of agents. Clearly any system that enables such programs to efficiently trade with each other needs to offer general economic efficiency while very strongly taking into account the fact that all participants have totally differing goals.

3 Existing Work

Game theory, Economics, and Computer Science

In recent years there have been many works that tried to introduce economic or game-theoretic aspects into computational questions. The approach presented here is part of this trend, but is much narrower, taking specifically the direction of mechanism design. The reader interested in the wider view may start his exploration e.g. with the surveys [8, 13], the book [21], the web sites [3, 1, 2], or the papers in the conference [4].

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 [16]. An introduction to this field can be found in [15, chapter 23] [20, chapter 10], and an influential web site in [17].

Mechanism design in Computer Science

One may identify three motivations for combining mechanism design with computational questions.

> **Auction implementation:** As auctions become more popular as well as more complicated, they are often implemented using computers and computer networks. Many computational implementation questions result. These range from purely combinatorial ones regarding optimization in complex combinatorial auctions to systems questions regarding communication and performance issues in wide-scale auctions.
>
> **Leveraging Market Power:** In the "real world" the invisible hand of free markets seems to yield surprisingly good results for complex optimization problems. This occurs despite the many underlying difficulties: decentralized control, uncertainties, information gaps, limited computational power, etc. One is tempted to apply similar market-based ideas in computational scenarios with similar complications, in the hope of achieving similarly good results.
>
> **Handling Selfishness:** This is the approach taken here, and it views mechanism design introduced into computational problems as a necessary evil, required to deal with the differing goals of the participants.

Even though these motivations are different philosophically, research often combines aspects from all approaches. Below we shortly sketch some of previous work done introducing mechanism design into different branches of computer science, without attempting to further classify them.

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 [21] [22] [7]. A related subfield of Distributed AI, sometimes termed market-based computation [26] [8] [25], 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. [11]). In particular mechanism design was applied to various problems including resource allocation [12], cost sharing, and pricing [23].

4 The Model

In this section we formally present the model. It is taken from the author's joint work with Amir Ronen [18].

The model is concerned with computing functions that depend on inputs that are distributed among n different agents. A problem in this model has, in addition to the specification of the function to be computed, a specification of the goals of each of the agents. The solution, termed a mechanism, includes, in addition to an algorithm computing the function, payments to be handed out to the agents. These payments are intended to motivate the agents to behave "correctly".

Subsection 4.1 describes what a mechanism design problem is. In subsection 4.2 we define what a good solution is: an implementation with dominant strategies. Subsection 4.3 defines a special class of good solutions: truthful implementations, and states the well-known fact that restricting ourselves to such solutions loses no generality.

4.1 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 (outcomes).

Definition 1 (Mechanism Design Problem) *A* mechanism design problem *is given by an output specification and by a set of agent's 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 $t = t^1 ... t^n$ a set of allowed outcomes o.*

3. *Each agent i's preferences are given by a real valued function: $v^i(o, t^i)$, called its* valuation. *This is a quantification of its value from the outcome o, when its type is t^i, in terms of some common currency. I.e. if the mechanism's outcome is o and in addition the mechanism hands this agent p^i units of this currency, then its* utility *will be $u^i = p^i + v^i(o, t^i)$*[1]*. This utility is what the agent aims to optimize.*

In this paper we only discuss optimization problems. In these problems the outcome 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 outcome specification is given by a positive real valued objective function $g(o, t)$ and a set of feasible outcomes F. The required output is the outcome $o \in F$ that minimizes g.*

4.2 The Mechanism

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

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

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

1. *The mechanism defines for each agent i a family of* strategies A^i. *Agent i can choose to perform any $a^i \in A^i$.*

2. *The first thing a mechanism must provide is an* outcome *function $o = o(a^1 ... a^n)$.*

[1]This is termed "semi-linear utility". In this paper we limit ourselves to this type of utilities.

3. *The second thing a mechanism provides is a* payment $p^i = p^i(a^1...a^n)$ *to each of the agents.*

4. *We say that a mechanism is an* implementation with dominant strategies *(or in short just an implementation) if*

   - *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 $a = (a^1...a^n)$ the outcome $o(a)$ satisfies the specification.*

4.3 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.*

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

Proposition 4.1 *([15], page 871) 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 $(o, p^1, ...p^n)$, with dominant strategies $a^i(t^i)$, we can define a new one by $o^*(t^1...t^n) = o(a^1(t^1)...a^n(t^n))$ and $(p^*)^i(t^1...t^n) = p^i(a^1(t^1)...a^n(t^n))$. $\square$

5 Applying Existing Mechanism Design Theory

In this section we present several well known mechanisms. While these mechanisms are the usual ones one would find in a standard text on mechanism design, we present them in a distributed-computation setting. The implementations provided are all truthful ones, i.e. they follow this pattern:

1. Each agent reports its input to the mechanism.
2. The mechanism computes the desired outcome based on the reported types.
3. The mechanism computes payments for each agent.

The challenge in these examples is to determine these payments as to ensure that the truth is indeed a dominating strategy for all agents.

5.1 Maximum

Story:[2]

A single server is serving many clients. At a certain time, the server can serve exactly one request. Each client has a private valuation t^i for his request being served. (The valuation is 0 if the request is not served.) We want the most valuble request to be served.

Failed attempts:

One might first attempt to simply ignore all payments (i.e. set $p^i = 0$ for all i). This however is clearly insufficient since it motivates each agent to exaggerate his valuation, as to get his request executed. The second attempt would be to let the winning agent pay his declaration. I.e. set $p^i = -t'^i$ for the agent i that declared the highest t'^i (and $p^i = 0$ for all others). This also fails since the agent with highest t^i is motivated to reduce his declaration to slightly above the second highest valuation offered. This will result in his request still being served, and his payment reduced. In case agent i has imperfect information about the others this strategic behavior may lead him to accidently declare a lower value than the second valuation, which will result in a sub-optimal allocation.

Solution:

The agent that offers the highest valuation for his request pays the second highest price offered. I.e. $p^i = -t^j$, where i offers the highest price and j the second highest. All other agents have $p^k = 0$.

Analysis:

To see why this is a truthful implementation, consider agent i and consider a lie $t'^i \neq t^i$. If this lie does not change the allocation, then nothing is gained or lost by agent i since his payment is also unaffected by his own declaration. If this lie gets his request served, then $t'^i > t^j > t^i$ and he gains t^i of utility from his valuation of the served request, but he loses t^j on payments, thus his total utility would be $t^i - t^j < 0$, as opposed to 0 in the case of the truth. On the other hand, if his lie makes him lose the service, then his utility is now 0, as opposed to a positive number which it was in the truthful case.

[2]This is an auction and the solution presented is Vickrey's well-known second price auction [24].

5.2 Threshold

Story:[3]

A single cache is shared by many processors. When an item is entered into the cache, all processors gain faster access to this item. Each processor i will save t^i in communication costs if a certain item X is brought into the cache. (I.e its valuation of loading X is $t^i > 0$, and of not loading it, 0.) The cost of loading X is a publicly known constant C. We want to load X iff $\sum_i t^i > C$.

Failed attempts:

We may first attempt to just divide the total cost between the n participating agents, i.e. set $p^i = -C/n$ for all i. This however motivates any agent with $t^i > C/n$ to announce his valuation as greater than C, and thus assure that X is loaded. We may, as a second attempt, let each agent pay the amount declared (or perhaps something proportional to it.) In this case, however, we will be faced with a free-rider problem, where agents will tend to report lower valuation than the true ones so as to reduce their payments. This, when done by several agents, may result in the wrong decision of not loading X.

Solution:

In case X is loaded, each agent pays a sum equal to the minimum declaration required from him in order to load X, given the other's declarations. I.e. the only case where $p^i \neq 0$, is when $\sum_{j \neq i} t^j \leq C < \sum_j t^j$, in which case $p^i = \sum_{j \neq i} t^j - C$ (a negative number).

The analysis is left to the reader. Alternatively, this example may be seen to be a special case of the example below.

This example can be generalized to the case where t^i can be negative as well.

5.3 Shortest Path

Story:

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 outcomes are all paths from x to y, and the objective function is the path's total cost. Agent e's valuation is 0 if his 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.

Solution:

The following mechanism ensures that the dominant strategy for each agent is to report his true type t^e to the mechanism. When all agents honestly report

[3] This is known as the "public project" problem, and the solution is known as the Clarke tax [5].

their costs, the cheapest path is chosen: The outcome 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 $p^e = d_{G-e} - (d_G - t'^e)$ if it is. Here t'^e is the agents' reported input (which may be different from its actual one), d_G is the length of the shortest path (according to the inputs reported), and d_{G-e} is the length of the shortest path that does not contain e (again according to the reported types).

Analysis:
First notice that if the same shortest path is chosen with t'^e as with t^e then the payment and thus utility of the agent does not change. A lie $t'^e > t^e$ will cause the algorithm to choose the shortest path that does not contain e as opposed to the (correct one) which does contain it iff $d_{G-e} - d_G < t'^e - t^e$. This directly implies that e's utility would have been positive had e been chosen in the path (as opposed to 0 when its not chosen), thus the truth is better. A similar argument works to show that $t'^e < t^e$ is worse than the truth.

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.

Algorithmic 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 $O(m \log n)$?

5.4 Utilitarian Functions

Arguably the most important positive result in mechanism design is what is usually called the generalized Vickrey-Groves-Clark (VCG) mechanism [24] [10] [5]. All previous examples are, in fact, VCG mechanisms. In this section we present the general case.

The VCG mechanism applies to mechanism design optimization problems where the objective function is simply the sum of all agents' valuations.

Definition 5 *An optimization mechanism design problem is called* utilitarian *if its objective function satisfies* $g(o,t) = \sum_i v^i(o, t^i)$.

Definition 6 *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. $p^i(t) = \sum_{j \neq i} v^i(o(t), t^i) + h^i(t^{-i})$ *where* $h^i()$ *is an arbitrary function of* t^{-i}.

Theorem 5.1 *(Groves [10]) A VCG mechanism is truthful.*

Proof: (sketch) Let $d^1, \ldots, d^n$ denote the declaration of the agents and $t^1, \ldots, t^n$ denote their real types. Suppose that truth telling is not a dominant strategy, then there exists d, i, t, d'^i such that

$$v^i(t^i, o(d^{-i}, t^i)) + p^i(t^i, o(d^{-i}, t^i)) + h^i(d^{-i}) <$$

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

But then

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

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

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 are the only truthful implementation for utilitarian problems ([9]).

5.5 More Issues in Mechanism Design

The examples presented here demonstrate only the most basic notions from the field of mechanism design. Many more issues addressed by the theory of mechanism design are applicable to the distributed computation setting. We briefly mention just some of the issues commonly studied by mechanism design (and other branches of game theory) that we feel may find applications in distributed computation.

Bayesian-Nash equilibrium: Our notion of a solution was very strong, requiring dominant strategies. Weaker notions of equilibrium are also often considered, in particular Bayesian-Nash equilibrium.

Non semi-linear utilities: We assumed that the utility of each agent is additive in the money. More general types of utilities may be considered, where money influences the utility in an arbitrary manner.

Budgets: We did not put any requirements on the sums of money involved in a mechanism. At least two types of constraints are widely studied: constraining the total money spent by the mechanism (either to as large a negative amount as possible, or to 0 – budget balance), and considering budget limitations of the agents.

Common value models: We assumed that each agent has a known valuation function that is independent from the others. One may alternatively assume a valuation that is common to all agents but is not fully known by them.

Repeated Games: We only considered a single instance of a problem. One may clearly consider repeated instances.

Coalitions: We only considered manipulation by a single agent. Clearly one may study coalitions of agents.

6 Beyond Existing Mechanism Design

We feel that the application of existing mechanism design in distributed computation, as demonstrated above, is just a first step. Many of the considerations of distributed computation are quite different from the ones usually considered in mechanism design. Addressing these considerations will thus require new research. In this section we exhibit several scenarios in distributed computation that raise questions that indeed go beyond the current scope of mechanism design.

6.1 Task Scheduling

Story:

A computer has k tasks it wishes to execute, and can execute each of them on any one of n servers. Each server i knows, for every task j, the time t_j^i it requires to execute this task. Each server's cost is proportional to the time it spends on executing the tasks assigned to it. Our goal is to have all tasks completed as soon as possible (i.e. to minimize the completion time of the last task.)

This problem was considered in [18]. Here are some of the issues raised by this problem and addressed there. Similar issues arise in many other problems in distributed computation.

Issues:

Non-utilitarian Problem: The goal in this example is non-utilitarian. Thus, the VCG mechanism cannot be applied and new mechanisms need to be invented.

Impossibility: It is possible to prove that no mechanism perfectly solves this problem. As is common in Computer Science, one should try to overcome this impossibility. In particular, the following approaches may be considered (and were all studied in [18]):

Approximation: Find a mechanism that approximates the optimal solution as well as possible.

Randomization: In Computer Science as well as in game theory randomization often helps. In turns out that for this problem, randomized mechanisms can provably do better than deterministic ones.

Model Extensions: Every model is an imperfect abstraction of reality. One may incorporate useful attributes of reality into the model as to make an impossible result possible. In [18] the model was extended by assuming that the mechanism need only compute the payments $after$ the tasks were actually executed, giving it additional information.

Computational Intractability: Even from a purely algorithmic point of view, the task scheduling problem is intractable (NP-complete). When adding the requirements of a mechanism things only get worse. In particular, standard ways of overcoming the computational intractability (such as tractable approximations) have complicated interactions with the requirements of mechanism design.

6.2 Maximum Independent Set

Story:

There are n processors connected in a linear array (i.e. each processor i is connected to $i-1$ and to $i+1$). Each processor wants to execute a single job, and values it at $t^i \geq 0$. The problem is that executing the job requires exclusive access to the common link with each of its neighbors. Thus no two consecutive processors can execute their job. Our goal is to execute the set of tasks with maximal valuation, i.e. to find an independent set S of processors that maximizes $\sum_{i \in S} t^i$.

Model Restriction:

In this story we want to find a decentralized solution. I.e. we want to design a protocol, that runs on *these computers*, using only the *available communication links*, and without assuming any central trusted computer, or any other communication links.

Solution:

Our protocol has two phases a left-to-right phase and a right-to-left phase. In the left-to-right phase, each processor places a bid R^i for link on its right. These offers are computed by each processor in turn as follows: $R^1 = t^1$, and for $1 < i < n$, $R^i = max(t^i - R^{i-1}, 0)$. In the right-to-left phase each processor places a bid L^i on the link to its left as follows: $L^n = t^n$, and for $1 < i < n$, $L^i = max(t^i - L^{i+1}, 0)$. Processor i wins the left link iff $L^i > R^{i-1}$ and wins the right link iff $R^i \geq L^{i+1}$. It can execute its task (i.e. is chosen to be in S) if it has won both links. In this case its payment is $-p^i = R^{i-1} + L^{i+1}$ (i.e. the second price on each of links it has won).

Analysis:

There are many issues to consider here:

Algorithmic correctness: One may verify that R^i is the difference between the weight of the maximum weight independent set in $1...i-1$ and the weight of the maximum weight independent set in $1...i$. Similarly, L^i is the difference between of the weights of the maximum weight independent sets in $i+1...n$ and $i...n$. Clearly i should be chosen to be in S if $t^i > L^{i+1}+R^{i-1}$ (ties can be broken arbitrarily), which is exactly what this protocol does. This protocol can be viewed as a dynamic programming solution of this problem.

Domination of the Truth: Assume that the players' strategies are limited to acting according to some fixed valuation t'^i. Such a model may be called the "honest but selfish" case. In this case one may observe that the protocol achieves the VCG mechanism that is a solution since the problem is indeed utilitarian.

Dishonesty: A more general model would allow all strategies made possible by the protocol. In this case the processors could act according to a different t'^i in each phase. One may verify that in this model the truth is no longer dominant. Yet, truth is still a Nash equilibrium.

Ensuring Honesty: There are various ways to augment the model as to force the processors to be consistent in both phases, and thus essentially force the "honest but selfish" situation. In particular, if processors $i-1$ and $i+1$ can communicate with each other then they can catch i's dishonesty. Such communication may alternatively be implicitly achieved by using cryptographic signatures.

Decentralized Payments: The payments in this solution were to be given to some party outside of the n involved processors. It would have been nice to have a mechanism where the payments are only transferred between connected processors.

6.3 Decentralized Auction

Story:

A single item is to be auctioned over the Internet among n humans (each with his own computer).

Restriction:

There is no trusted entity. In particular we do not trust the auctioneer to faithfully execute the auction rules or to keep any secrets. In the absence of such a trusted entity we would like to ensure two goals:

- The auction is executed according to the published auction rules (e.g. second price).
- No information about bids is leaked to any participant, beyond the results of the auction which become public knowledge. I.e. only the identity of the winner (but not his bid), and the amount of the second highest bid (but not the identity of the bidder) become known.

Solution:
The celebrated "oblivious circuit evaluation" cryptographic protocols [19, 14, 6] exactly achieve this goal (as long as not too many of the participants collude to lie). These cryptographic protocols can faithfully carry out any distributed computation without leaking any information to the participants. What cannot, in principle, be ensured by cryptography is that the participants reveal their inputs. This, however, is ensured by the mechanism. We should note that these cryptographic protocols, while theoretically tractable, are quite impractical.

7 Acknowledgments

The notions expressed in this paper are derived from my joint work with Amir Ronen, who has also helped with the writing of this paper. I thank Dov Monderer, Motty Perry, and Moshe Tennenholtz for helpful discussions.

References

[1] Comet group technical reports. Web Page: http://comet.ctr.columbia.edu/publications/techreports.html.

[2] The information economy. Web Page: http://www.sims.berkeley.edu/resources/infoecon/.

[3] Market-oriented programming. Web Page: http://ai.eecs.umich.edu/people/wellman/MOP.html.

[4] First international conference on information and computation economies ice-98. Web Page: http://www.cs.columbia.edu/ICE-98/, October 1998.

[5] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, pages 17–33, 1971.

[6] C. Crepeau D. Chaum and I. Damgard. Multiparty unconditionally secure protocols. In *20th STOC*, 1988.

[7] 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.

[8] 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.

[9] J. Green and J.J. Laffont. Characterization of satisfactory mechanism for the revelation of preferences for public goods. *Econometrica*, pages 427–438, 1977.

[10] T. Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.

[11] 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.

[12] 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.

[13] Nathan Lineal. Game theoretic aspects of computing. In *Handbook of Game Theory*, volume 2, pages 1339–1395. Elsevier Science Publishers B.V, 1994.

[14] S. Golwasser M. Ben-Or and A. Wigderson. Completeness theorems for fault-taulerent distributed computing. In *20th STOC*, 1988.

[15] A. Mas-Collel, W. Whinston, and J. Green. *Microeconomic Theory*. Oxford university press, 1995.

[16] J. McMillan. Selling spectrum rights. *Journal of Economic Perspectives*, pages 145–162, 1994.

[17] Market design inc. Web Page: http://www.market-design.com.

[18] Noam Nisan and Amir Ronen. Algorithmic mechanism design. Avilable at http://www.cs.huji.ac.il/˜amiry.

[19] S. Micali O. Goldreich and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *27th FOCS*, 1986.

[20] M. J. Osborne and A. Rubistein. *A Course in Game Theory.* MIT press, 1994.

[21] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers.* MIT Press, 1994.

[22] 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.

[23] S. Shenkar, Clark D. E., and Hertzog S. Pricing in computer networks: Reshaping the research agenda. *ACM Computational Comm. Review*, pages 19–43, 1996.

[24] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, pages 8–37, 1961.

[25] 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.

[26] 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.