Econ. Theory 6, 225–250 (1995)

Economic Theory

The complexity of resource allocation and price mechanisms under bounded rationality*

Eric J. Friedman[1] **and Shmuel S. Oren**[2]

[1] The Fuqua School of Business, Duke University, Durham, NC 27708-0120, USA
[2] Department of IEOR, University of California, Berkeley, CA 94720, USA

Received: June 14, 1993; revised version March 15, 1994

Summary. We develop a framework for designing and evaluating the complexity of mechanisms that allocate resources in a distributed setting to agents or processors with bounded computational ability. We discuss several mechanisms and describe the construction of efficient price based mechanisms, which exploit the decentralized aspects of the problem. These price mechanisms are polynomial in the number of resources, precision of the solution, and the logarithm of the number of agents.

1 Introduction

In this paper we consider the problem of allocating resources among a large group of processors (or agents) in a distributed setting in order to maximize the total social welfare of an economy. It can also be interpreted as maximizing total value or profit in an organization. We provide a framework for studying such mechanisms, which we then use to evaluate several different ones. Our goal is to design efficient mechanisms that can compute reasonable allocations rapidly in a distributed setting.

The design of mechanisms for the allocation of scarce resources among a large group of processors is a fundamental topic in modern economics [1, 10, 14]. This problem is also of great importance in distributed computing, where resources may refer to CPU time, network bandwidth, and data storage. In fact much recent work has involved the application of economic ideas to problems in distributed computing. (See e.g. [5, 9, 15, 17, 33, 32, 36].)

The formal study of 'efficiency' for different mechanisms was instituted by Reiter and Hurwicz [12, 24] and has produced a large literature on informationally efficient mechanisms. However, most of these analyses consider mechanisms which find optimal or Pareto efficient allocations at equilibrium [13]. The incorporation of dynamic elements has only been attempted in a limited manner [23]. However, the time required to actually find a good (or optimal) allocation has not previously

* We thank Thomas Marschak and a anonymous referee for helpful suggestions. This research was been supported by National Science Foundation Grant IRI-8902813.

been studied. Furthermore, the complexity of a mechanism has been defined as the number of messages required at equilibrium, not the computational effort or total number of messages required overall. We believe that the total time and effort required to compute an allocation should be considered when designing mechanisms, as these directly determine how well they function.

Most of the work in this field has implicitly assumed that the processors have unlimited computational ability, since they are required to exactly solve difficult optimization problems instantaneously. This viewpoint is certainly unrealistic and has recently been questioned; typically, the processors in an economy are not infinitely wise, neither is the designer of the mechanism. Such limitations have been referred to as 'bounded rationality' [37, 31].

Several approaches have attempted to account for the decision maker's bounded rationality. Simon [37] considers 'satisficing' mechanisms, where mechanisms must perform in an informally defined satisfactory manner. Mount and Reiter [25] consider the design of mechanisms when the participants are variants of finite automata.

Recently, Radner [29] has analyzed information processing in an organization from a computational viewpoint which is very similar to ours. However, he considers the problem of designing an efficient organizational structure, where we assume that the structure is predetermined and concentrate on the protocols within the structure.

In this paper we consider a very natural model of 'finite rationality'. We model the participants as 'processors' where each one has a finite 'speed of computation' and employ concepts traditionally used in the context of complexity theory for combinatorial optimization. For our definition of 'processor' we use the recent idea of 'computation over the real numbers' as discussed in [3, 22, 2]. Our 'processors' can manipulate real numbers and do the following operations in unit time: addition, subtraction, multiplication, division, and comparison. This model of computation is both elegant and useful analytically. It is also, perhaps, a more realistic model of the workings of modern computers and numerical analysis. Nonetheless, most of our discussion and results are also valid in standard models of binary computation, such as Turing machines, with only slight modifications.

We consider the problem of constructing mechanisms which compute allocations that are guaranteed to be 'good', in the sense of social utility,[1] but not 'perfect', in a reasonable amount of time. The most important aspect of these mechanisms is that they scale well with the number of processors. That is, even if the number of processors is very large our mechanism will still operate in a reasonable amount of time.[2]

---

[1] Note that by optimizing social utility we are in effect choosing a specific pareto optimal allocation. In the case of a single resource, this problem is strictly harder than finding a pareto optimal allocation, as any feasible allocation is parteto optimal. However, in the case with two or more resources, it does not seem that finding a pareto optimal solution, which is pareto superior to an initial endowment is necessarily easier than the problem we consider.

[2] Similar ideas can be found in Marschak [21]. He explicitly considers the running time of a mechanism, but allows processors to use infinite computation.

Previous results for such mechanisms are scarce. The case of a single resource has been well studied, and several polynomial algorithms exist [15, 8]. For multiple resources Nemirovsky and Yudin [26] provide the framework for constructing such mechanisms, but do not take full advantage of the structures inherent in the problem. In [7] we describe the construction of a mechanism for a generalization of the multi-resource allocation problem. However, none of these methods provide a decentralized mechanism for allocating multiple resources that converges rapidly when the number of processors is large. Our goal is to construct such a mechanism.

In the next section we formally define our model, and some measures of complexity and efficiency for mechanisms. We then consider quantity based mechanisms. In the context of mathematical programming these are often denoted primal algorithms. We describe two such mechanisms. These are only discussed briefly as they have been well treated elsewhere [7, 26]. However, neither of these mechanisms satisfies our definition of efficiency, which we denote polynomial-efficient in order to emphasize our reliance on the theory of computational complexity. This is a standard problem with primal algorithms as they do not appear amenable to distributed implementation.

Our main focus is on price-based mechanisms and the construction of polynomial-efficient mechanisms using prices. These can be interpreted as finite versions of primal-dual algorithms. These mechanisms are naturally implemented in a distributed system, as primal-dual algorithms typically distribute well. For example Arrow and Hurwicz use a primal dual method for constructing decentralized mechanisms in [1]. These algorithms are very efficient as they take advantage of the structure inherent in resource allocation problems.[3] For example the price mechanism for a single resource is the only polynomial-efficient distributed algorithm to our knowledge. For two resources it is the only algorithm we know which exploits the structure of this problem and is the only polynomial-efficient serial or parallel algorithm known to us. For more than two resources we conjecture that the same is true.

Finally, we note that these mechanisms can be seen as extensions of a theory of polynomial-efficient optimization expounded in [7] which is based on the work of Nemirovsky and Yudin [26]. This should be contrasted with the standard asymptotic results in nonlinear programming. In those results, the speed of convergence is guaranteed in a small neighborhood of the solution. Also, the asymptotic theory relies very strongly on the analytic properties of the functions being optimized. Polynomial-efficient results are based on convexity properties of the functions, but they are global and guarantee that the (approximate) solution of the problem will be found in a specified finite time.

The specific result of this paper is that the polynomial-efficient theory can be applied to problems with special structure, in order to reduce the number of

[3] The formal description of mechanisms reflects the interplay between the theory of mechanisms and that of mathematical programming. In fact many of the mechanisms found in the economics literature have their inception in mathematical programming, and currently many algorithms for mathematical programming are using ideas from economics for their inspiration. Our work is no exception.

computations required to find a solution. Note that we are not considering incentives or initial endowments of agents.

The paper is structured as follows. In section two we define a model of complexity for resource allocation, both with and without a center. Then section three presents two quantity based mechanisms, which are modeled after known methods of optimization. Since neither of these mechanisms are efficient we are led to the construction of new, price based, mechanisms in section four. In the first part of section four we construct a price mechanism for a single resource. Although much of this part is pedagogical, the mechanism constructed is a significant improvement over known algorithms for the distributed allocation of a single resource. In the second part we construct a price based mechanism for two resources and propose a similar construction for any number of resources. Since the proofs for this part are quite involved, we present them separately in the appendix. Section five describes the extension of our mechanisms to the important case of externalities and section six contains our conclusions.

2 Model

We consider the problem of allocating r resources among n processors. The basic data for an economy is the environment or set of utility functions for the processors. This is denoted by $\mathbf{U} = (U_1, U_2, \ldots, U_n) \in \Upsilon$, where Υ is the set of possible environments. We will assume that $\Upsilon = \Upsilon_1 \times \Upsilon_2 \times \cdots \times \Upsilon_n$ where each Υ_i is the set of all C^2 nondecreasing concave (utility) functions $U_i: \mathfrak{R}^r \to \mathfrak{R}^+$ with $\left|\frac{\partial U_i}{\partial x_i^j}\right| \leq 1$, and $U_i(0) = 0$. Note that since U_i is concave, the condition that $\left|\frac{\partial U_i}{\partial x_i^j}\right| \leq 1$ is quite mild and is essentially a normalization.[4]

As is typical, we assume that U_i is private information – it is only known to processor i. However, we assume a more specific form of privacy than is usual. We assume that the processor has access to a utility function oracle which can be queried and supply the processor with local information about the utility function such as values and derivatives of $U_i(\mathbf{x}_i)$ for specific values $\mathbf{x}_i$ provided by the processor. Each query requires one unit of time. This is an attempt to capture the bounded rationality of the processors' knowledge as manifested by the local information available to it, since global information would allow it to 'smuggle' information and computation that would not be accounted for.

For ease of exposition we will assume that there is 1 unit of each resource. The set of feasible allocations is

$$F = \left\{ \mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \mid \mathbf{x}_i \in \mathfrak{R}^r, \sum_{i=1}^{n} x_i^j = 1, \quad j = 1, \ldots, r \right\}.$$

These restrictions are easily relaxed at the cost of additional notation.

[4] The restriction to differentiable functions is not really necessary, and could be removed at the cost of more elaborate proofs.

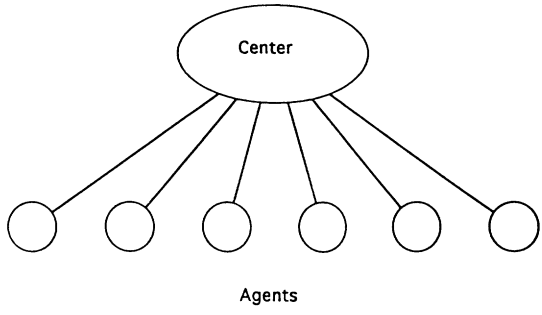

Figure 1. A central mechanism

The goal of a mechanism will be to rapidly compute an allocation that approximately maximizes the total utility[5] $U(\mathbf{x})$ subject to $\mathbf{x} \in F$, where $U(\mathbf{x}) = \sum_i U_i(\mathbf{x}_i)$.

We will consider both central and distributed mechanisms. Central mechanisms are very common in economics. For motivation consider the standard Walrasian process consists of a center shouting prices to a large group of processors. Central mechanisms can also arise in distributed computing when many processors are sharing a single resource. However, center-less mechanisms are often more relevant for distributed computing, in which a large collection of processors (such as workstations or personal computers) are interconnected on a large network. We show that any of the central mechanisms we design are easily implemented in a distributed setting with a negligible loss of efficiency.

2.1 Central mechanisms

We will assume that there are many processors interacting with a computationally powerful center. Thus we assume that the 'center' has computing speed proportional to the number of processors, and each processor has a unit speed of computing. The center communicates with the processors by broadcasting a single real number to all the processors simultaneously. The processors can respond by each one sending a single real number to the center. (See figure 1.)

We define a mechanism as a vector of computer programs (or algorithms)

$$\mathbf{m} = (m_1, \ldots, m_n, m_c) \in M_1 \times \cdots \times M_n \times M_c$$

where each m_i represents a program that computer i follows. Let

$$\mathbf{x}(\mathbf{m}, \mathbf{U}) = (x(m_1, U_1), \ldots, x(m_n, U_n))$$

be the output of mechanism $\mathbf{m}$ in environment $\mathbf{U}$. (We will only consider the case of mechanisms which have feasible outcomes $\mathbf{x}(\mathbf{m}, \mathbf{U}) \in F$.)

---

[5] Note that this socially optimal allocation is also Pareto Efficient. In the case of a single resource this problem is inherently more difficult than the standard problem in an exchange economy. However, when there are two or more resources it is not clear which problem is more difficult, or more relevant. This is an important open problem.

Define the error of mechanism $\mathbf{m}$ in environment $\mathbf{U}$ to be

$$\varepsilon(\mathbf{m}, \mathbf{U}) = |U(\mathbf{x}(\mathbf{m}, \mathbf{U})) - U(\mathbf{x}^*(\mathbf{U}))|$$

where

$$\mathbf{x}^*(\mathbf{U}) = \operatorname{argmax}_{\mathbf{x} \in F} U(\mathbf{x})$$

So $\varepsilon(\mathbf{m}, \mathbf{U})$ is the error of $\mathbf{m}$ on $\mathbf{U}$. Now define the error of a mechanism to be

$$\varepsilon(\mathbf{m}) = \sup_{\mathbf{U} \in \Upsilon} \varepsilon(\mathbf{m}, \mathbf{U}).$$

We are interested in parametrized families of mechanisms, $\{\mathbf{m}^{\varepsilon,n,r}\}$, where $\mathbf{m}^{\varepsilon,n,r}$ is a mechanism that for n processors and r resources has $\varepsilon(\mathbf{m}^{\varepsilon,n,r}) \leq \varepsilon$ for all $\mathbf{U} \in \Upsilon$.

Our mechanism must perform three different actions: computation, communication, and utility function evaluation. We represent the time required for each of these operations symbolically. Let X be the amount of time required for a single (real number) computation by a processor and X/n for the center, I be the time for the communication of a number, and Ω the time required for a single query of a utility function. Note that the processors and the center can each perform a single action simultaneously.

We will denote the complexity of a mechanism to be the time required by that mechanism to compute an allocation. We will compare the behavior of complexities by considering this time

$$T(\mathbf{m}^{\varepsilon,n,r}) = \mathcal{O}(G(\varepsilon, n, r))$$

where $G(\cdot)$ is a (symbolic) function containing X, I, Ω's such that

$$T(\mathbf{m}^{\varepsilon,n,r}) \leq \kappa G(\varepsilon, n, r)$$

for some constant κ and all ε, n, r.

Thus we can compare different mechanisms by comparing their complexities. As in standard complexity theory of parallel computing [18], an algorithm will be deemed 'polynomial-efficient' if it is bounded by a polynomial in r, $\log n$, and $\log(1/\varepsilon)$. This is based on the standard definitions used in computer science (there are a reasonable number of resources, a very large number of processors) and the condition that the dependence on ε be equivalent to an algorithm that is at least linearly convergent.

2.2 Distributed mechanisms

Distributed mechanisms can easily duplicate central mechanisms with only a slight loss of efficiency. In a distributed mechanism there is no center. We imagine that each processor can do one computation in time X. We also assume that the processors are connected on a network. Thus there is a graph $G = (V, E)$ where each vertex corresponds to a processor $V = \{1, 2, \ldots, n\}$ and each edge connects two processors $(i, j) \in E$. However, we require that the graph not have too many edges, thus preventing everyone from talking to everyone else, which would not be realistic. A useful (and non-restrictive) assumption is that each processor is connected to a small number of other processors.

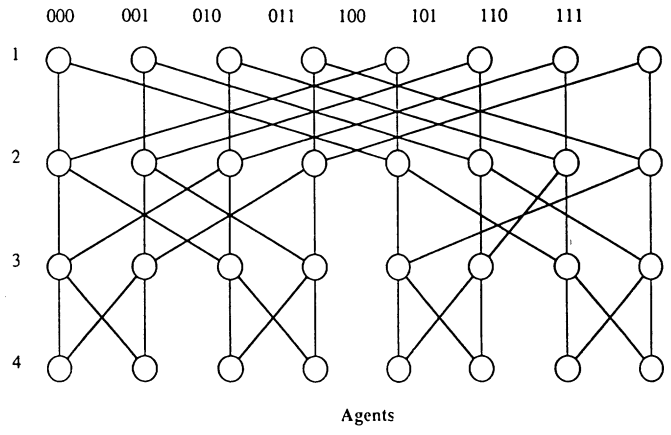

Figure 2. Agents organized on a Butterfly network

For concreteness we assume that the processors are connected via a butterly network.[6] (See figure 2.) In a butterfly network each processor is connected to four other processors in an array that (somewhat) resembles a butterfly. This network allows many distributed computations to be performed rapidly. For example sums of numbers and matrix operations can be performed in $\mathcal{O}(\log k)$ time, where k is the number of real numbers involved in the computation [18].

Now note that any central mechanism can be implemented as a distributed mechanism by simply designating some processor to act as the center. However, typically this would increase the complexity by a factor of $\mathcal{O}(n)$ as the processor computes more slowly than the center does, which would make any polynomial-efficient central mechanism into a polynomial-inefficient distributed one.

This can be avoided by exploiting the power of distributed computations by the network. All of the mechanisms which we describe can be implemented in a distributed manner. In each of these mechanisms the complexity is increased by a factor of $\log n$ in the number computations and information exchanges. This does not effect the polynomial-efficiency of any of the mechanisms.

3 Quantity mechanisms

Quantity based mechanisms operate on the allocation space and have the property that (after a period of 'initialization') the intermediate allocation is always feasible. Therefore if the mechanism must be terminated before its completion, it will still supply a feasible (and reasonably good) allocation.

These mechanisms, however, have shortcomings. The only quantity based mechanisms with finite polynomial-efficiency that we are aware of fall into two classes. These are 'generalized bisection' methods, and 'descent' methods, each of which has its own shortcomings.

---

[6] Actually any expander graph will suffice. (See [18].)

Generalized bisection methods which are represented by the ellipsoid mechanism,[7] do not distribute well, and use very little of the processors' computing power.

Gradient methods as represented by the mirror descent mechanism [26], can be distributed effciently. However they are not polynomial in $\log 1/\varepsilon$ as they converge sublinearly.

This can be seen in the following theorems which are based on the work of Nemirovsky and Yudin [26].

Theorem 1 *A mechanism based on the ellipsoid method can be constructed which has complexity*

$$T(\mathbf{m}^{\varepsilon,n,r}(ellipsoid)) = \mathcal{O}\left(r^3 n^2 \log(rn/\varepsilon)\left[X\left(1 + \frac{\log 1/\varepsilon}{rn} \right) + I + \Omega \right] \right)$$

for the resource allocation problem.

The proof is omitted for brevity. It is a straightforward application of the algorithm described in [26, 28]. (Note that Nemorovsky and Yudin call this algorithm the modified method of center of gravities.)

Theorem 2 *A mechanism based on the method of mirror descent can be constructed which has complexity*

$$T(\mathbf{m}^{\varepsilon,n,r}(md)) = \mathcal{O}\left(r^3 \frac{\log(rn)}{\varepsilon^2}\left[X\left(1 + \frac{\log 1/\varepsilon}{rn} \right) + I + \Omega \right] \right)$$

for the resource allocation problem.

The proof is omitted for brevity. It is a straightforward application of the algorithm described in [26].

Thus, primal mechanism do not satisfy our definition of polynomial-efficiency. The ellipsoid mechanism fails because the computation is not truly distributed and the center must do a disproportionate share of the work. The Mirror Descent Mechanism also fails due to its poor dependence on ε, i.e. $\mathcal{O}(1/\varepsilon^2)$.

Even for the case of one resource, primal algorithms with known bounds on their complexity [15, 8] fail our criteria of polynomial-efficiency as they distribute poorly. In the next section we describe a primal-dual approach that overcomes the weaknesses of primal mechanisms.

4 Price mechanisms

The idea of a price mechanism is based on the idea of a Walrasian process, which is described in detail in [40]. Thus, our process is based on the idea of a center announcing a sequence of prices, and processors picking a consumption bundle

[7] The ellipsoid method was developed by Nemirovsky and Yudin [26] based on an idea of Levin [19]. It is most well known from its use by Khachian [16] to prove the polynomiality of linear programming. However, its original purpose was for convex programming. While it is useful theoretically for linear programming it does not seem to be of practical value; however, it may actually be useful for convex programming [4]. In fact, recent modifications that increase the efficiency of the ellipsoid method have found useful applications. (See [39, 34].)

that maximizes $U_i(x_i) - p \cdot x_i$. This process continues until a feasible allocation is reached.

The importance of this process is demonstrated by Hurwicz [11]. He shows that this process defines an optimal mechanism, in the sense of having the minimal message space of any static mechanism which implements a Pareto optimal allocation. However, this mechanism is not stable [35]. A modification of it based on the 'Global Newton Method' [38] is stable, but requires a larger message space. These mechanisms assume that processors can provide their exact optimal consumption bundle for a given set of prices. Thus processors are able to instantly compute the optimum of a difficult optimization problem, while any such computation requires an arbitrarily large number of computational steps. The issue of accuracy and computation ability of processors in this situation has been neglected in the literature.

In this section we show how to construct finite 'computational' mechanisms based on the Walrasian mechanism. These mechanisms are polynomial-efficient and naturally distributed.

4.1 One resource

In this section we present a price based algorithm for allocating a single resource. The algorithm we present is not the simplest possible, but it is the one most amenable to generalization. Most of the ideas in its construction are extendible to the multi-resource problem. Thus, our goal here is to provide a basis for constructing other price based mechanisms.

This mechanism operates in two stages. (Recall that we are trying to maximize $U(\mathbf{x})$ over all feasible allocations, given a prespecified amount of the resource.) In the first stage the center announces a price and the processors compute a resource utilization that approximately maximizes

$$L_i(x_i, p) = U_i(x_i) - p(x_i - 1/n)$$

to a given accuracy. This is done using a finite accuracy version of a binary search to locate the zero of the derivative

$$L_i'(x_i, p) = U_i'(x_i) - p.$$

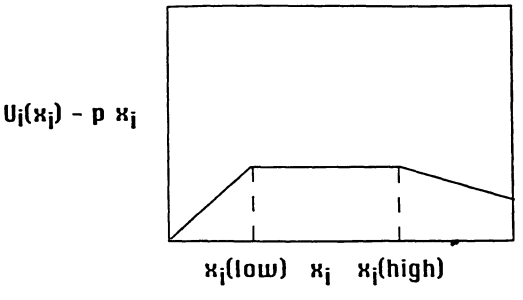

Figure 3. Multiple allocations can result from a single price. Note any allocation between x(low) and x(high) is of the same value to consumer i

Using this information the center computes a new price and the process continues until the center has found a price $\hat{p}$ that approximately minimizes

$$L(\mathbf{x}, p) = \sum_i L_i(x_i, p).$$

However, the current allocation may not be feasible for two reasons. First, even if the price was the correct price p^*, the resulting allocation might not be feasible if the processor's utility function was flat at this price, and its marginal utility constant, thus allowing for a wide choice of possible consumption choices all with the same net utility. (See figure 3.) Another problem is that the price $\hat{p}$ may be arbitrarily far from p^* due to inter-processor effects.

This problem is remedied in stage 2. Basically, the center asks the processors for their maximum and minimum consumption choices x_i that are reasonably good (within ε of optimal), given the price $\hat{p}$, which the processors compute approximately using a finite accuracy binary search. By noting that a feasible allocation must occur for some x with the consumption by each processor in the interval between these two points, we see that some convex combination of these points must give a feasible allocation. This combination $\hat{x}$ is then used as the allocation. While this allocation may be arbitrarily different from the optimal allocation ($\|\hat{x} - x^*\|$ may be quite large) it is still an accurate one in the sense of $|U(x^*) - U(\hat{x})|$ as it is the convex combination of two other good allocations and convex combinations preserve the accuracy of solutions.

Note the interplay between the processors and the center. Essentially the center is computing a good approximate allocation by asking questions of the processors, who are computing approximate answers. Thus there are two levels of computation and two levels of approximation.

Price mechanism for one resource

Center's program—$\mathbf{m}_c^{\varepsilon,n,1}$(*price*)
Stage 1:

1. Let $p_l = 0, p_h = 1$, and $\hat{\varepsilon} = \varepsilon/3n$.
2. Repeat $N_c = [\log(1/\hat{\varepsilon})]$ times.
   (Perform a binary search for the minimum of $L(\mathbf{x}(p), p)$ where $\mathbf{x}(p)$ is computed approximately by the processors.)
   (a) Let $p_m = (p_l + p_h)/2$
   (b) Let $\hat{p} = p_m$.
   (c) For all i: transmit $p_m + \hat{\varepsilon}/2$ to processor i.
   (d) For all i: receive $l_i^+ = L_i(x_i^+, p_m + \hat{\varepsilon}/2)$ from processor i.
   (e) For all i: transmit $p_m - \hat{\varepsilon}/2$ to processor i.
   (f) For all i: receive $l_i^- = L_1(x_i^-, p_m + \hat{\varepsilon}/2)$ from processor i.
   (g) Compute $l^{\pm} = \sum_i l_i^{\pm}$.
   (h) If $|l^+ - l^-| \leq \varepsilon$ goto 'Stage 2'.
   (i) Else if $l^+ > l^-$ let $p_h = p_m$.
   (j) Else let $p_l = p_m$.
3. Continue.

Stage 2:

1. For all i: transmit $\hat{p}$ to processor i.
2. For all i: receive $x_i^{\pm}$ from processor i.
3. Compute $s^{\pm} = \sum_i x_i^{\pm}$. (The maximum and minimum consumption amounts that are ε-accurate.)
4. Let $\alpha = (s^+ - 1)/(s^+ - s^-)$. (The coefficient for combining $\mathbf{x}^+$ and $\mathbf{x}^-$.)
5. For all i: transmit α to processor i.

Processor's program—$\mathbf{m}_i^{\varepsilon,n,1}(price)$

Stage 1:

1. Receive p.
2. Let $x_l = 0$ and $x_h = 1$.
3. Let $\varepsilon' = \hat{\varepsilon}^2$.
4. Repeat $N_a = \lceil \log(1/\varepsilon') \rceil$ times.
   (Perform a binary search to find the optimal consumption at price p.)
   (a) Let $x_m = (x_l + x_h)/2$.
   (b) Let $v_m = U_i'(x_m)$.
   (c) If $v_m < p$ let $x_l = x_m$.
   (d) Else let $x_h = x_m$.
5. Continue.
6. Transmit $L_i(x_m, p)$ to the center.
7. Go to 1.

Stage 2:

1. Receive $\hat{p}$ from center.
2. Do for $w = \pm 1$.
   (a) Let $x_l = 0$ and $x_h = 1$.
   (b) Set $\hat{l} = L_i(\hat{x}_i, \hat{p})$.
   (c) Repeat $[\log(\hat{\varepsilon})]$ times.
   (Do a binary search to find the maximum (resp. minimum) value of x_i such that $|L_i(x_i, p) - L(\hat{x}_i, \hat{p})| \leq \hat{\varepsilon}$.)
      i. Let $x_m = (x_l + x_h)/2$.
      ii. Let $l_m = L_i(x_m, \hat{p})$ and $v_m = U_i'(x_m)$.
      iii. If $|\hat{l}_m - l_m| < \hat{\varepsilon}$ let $x_l = x_m$ if $w = +1$ otherwise let $x_h = x_m$.
      iv. Else if $v_m < p$ then let $x_l = x_m$.
      v. Else let $x_h = x_m$.
3. Continue.
4. Transmit $x^{\pm}$ to center.
5. Receive α from center.
6. Compute $x_i = \alpha x_i^+ + (1 + \alpha) x_i^-$. (This is the allocation to processor i.)

Theorem 3 *The* 1 *resource price mechanism* $\mathbf{m}^{\varepsilon,n,1}(price)$ *has complexity*

$$T(\mathbf{m}^{\varepsilon,n,1}) = \mathcal{O}\left(\left(\log\frac{n}{\varepsilon}\right)^2 [X + I + \Omega]\right).$$

Proof: It is easy to see that stage 1 dominates stage 2. During stage 1 the center performs $\log(1/\hat{\varepsilon})$ iterations, and each iteration requires the computation of a sum,

a few simple computations, and two calls to the processors to compute their own optimization problem. During these calls each processor performs $\log(1/\varepsilon')$ iterations, and each iteration requires one call to the oracle and several basic computations. This shows that the complexity is as stated.

To prove that the algorithm computes a correct solution consider the Lagrangian

$$L(\mathbf{x}, p) = \sum U_i(x_i) - p\cdot\left(\sum x_i - 1\right)$$

and the dual function

$$F(p) = \max_{\mathbf{x}} L(\mathbf{x}, p).$$

By the duality theory of convex programming [30] we know that $F(p)$ is a convex function and if $p^* = \operatorname{argmin}_p F(p)$ then

$$\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} L(\mathbf{x}, p^*).$$

Define $\mathbf{x}(p) = \operatorname{argmax}_{\mathbf{x}} L(\mathbf{x}, p)$ so $\mathbf{x}^* = \mathbf{x}(p^*)$. The Lagrangian is separable in $\mathbf{x}$ and can be written as $L = \sum_i L_i$ where

$$L_i(x_i, p) = U_i(x_i) - p(x_i - 1/n)$$

so $x_i(p) = \operatorname{argmax}_{x_i} L(x_i, p)$.

Note that the Lipschitz constant of $F(p)$ is

$$\Lambda(F) = \max_p \left|\frac{dF(p)}{dp}\right| = \left|\frac{\partial L(\mathbf{x}, p)}{\partial p}\right| = \left|\sum_i x_i - 1\right| \le n$$

by the envelope theorem [40].

Now we prove the theorem through two lemmas.

Lemma 1 *Stage* 1 *computes a* $\hat{p}$ *such that*

$$|F(\hat{p}) - F(p^*)| \le \hat{\varepsilon}.$$

Proof: Notice that the processors simply perform a binary search for the x_i that satisfies $U_i'(x_i) = p$, which is the condition for optimality. Thus the x_i that the processor computes, which we denote $\hat{x}_i(p)$, is within ε' of the optimal $x_i(p)$ which implies that $U_i(\hat{x}_i(p))$ is within ε' of $U_i(x_i^*(p))$.

The center is performing binary search on $F(p)$ using solutions that are guaranteed to be accurate to ε' in each coordinate and thus $n\varepsilon'$-accurate in total using approximate gradients. Note that as we only need the sign of the derivative this inaccuracy is not important except when the computed derivative is close to zero. Thus, if the loop terminates normally then we know that this has not occurred and the solution must be accurate to $\hat{\varepsilon}$. However, if at some iteration the center finds a derivative that is close to zero it immediately halts. This is correct since for this to occur the derivative at this point must be less than $\hat{\varepsilon}$. This implies that either p_m is within $\hat{\varepsilon}$ of p^* or that all three, p_m and $p^{\pm}$ are approximately equal. This also implies that $F(p_m)$ is within $\hat{\varepsilon}$ of the optimal solution no matter how far p_m is from p^*. For example assume that $p^* > p_m + \hat{\varepsilon}$ then

$$F(p_m + \hat{\varepsilon}) \le (1 - \hat{\varepsilon})F(p_m) + \hat{\varepsilon}F(p^*)$$

by convexity. Therefore

$$F(p_m + \hat{\varepsilon}) - F(p_m) \leq \hat{\varepsilon}(F(p^*) - F(p_m))$$

and since $|F(p_m + \hat{\varepsilon}) - F(p_m)| \leq \hat{\varepsilon}$ we see that

$$0 \geq F(p^*) - F(p_m) \geq -\hat{\varepsilon}/n \geq -\varepsilon. \quad \diamond$$

Lemma 2 *Stage 2 computes an ε-accurate, feasible solution.*

Proof: In stage two the processors compute an approximation of x_i^+ and x_i^- which are solutions of

$$x_i^{\pm} = \text{argmax}_{x_i}(\pm x_i) \text{ s.t. } L_i(x_i, \hat{p}) > L_i(\hat{x}_i, \hat{p}) - \hat{\varepsilon}$$

that is accurate to $\hat{\varepsilon}$. Now if we can show that $s^+ \geq 1$ and $s^- \leq 1$ then $0 \leq \alpha \leq 1$ and the allocation x_i is a convex combination of $x_i^{\pm}$. Now as $|L(\mathbf{x}^{\pm}, \hat{p}) - F(\hat{p})| \leq 2n\hat{\varepsilon}$ we see that $|L(\mathbf{x}, \hat{p}) - F(\hat{p}))| \leq 2n\hat{\varepsilon}$ as $L(\mathbf{x}, p)$ is concave in x and thus $L(\mathbf{x}, \hat{p}) \geq \min L(\mathbf{x}^{\pm}, \hat{p})$ by concavity.

From this we immediately see that

$$|L(\mathbf{x}, \hat{p}) - F(p^*)| = |L(\mathbf{x}, \hat{p}) - L(\mathbf{x}^*, p^*)| \leq 3n\hat{\varepsilon}$$

and as both $\mathbf{x}$ and $\mathbf{x}^*$ are feasible this implies that

$$|U(\mathbf{x}) - U(\mathbf{x}^*)| \leq 3n\hat{\varepsilon} \leq \varepsilon$$

thus showing that the solution is sufficiently accurate.

Now we must show that $s^+ \geq 1$ and $s^- \leq 1$. If the center in stage 1 never saw a small derivative then $|\hat{p} - p^*| \leq \hat{\varepsilon}$ this implies that

$$|L(\mathbf{x}^*, p^*) - F(\mathbf{x}^*, \hat{p})| \leq n\hat{\varepsilon}$$

as the Lipschitz constant of $L(\mathbf{x}, p)$ is less than n. Now as $L(\hat{\mathbf{x}}, \hat{p}) \leq L(\mathbf{x}(\hat{p}), \hat{p})$ then this implies that x_i^* is an $\hat{\varepsilon}$ accurate solution to the processor's stage 2 problem. Thus $x_i^- \leq x_i^* \leq x_i^+$ implying that $s^+ \geq 1$ and $s^- \leq 1$.

However, if the center terminated stage 1 due to a small derivative then we know that $\left|\frac{\partial F(\hat{p})}{\partial p}\right| \leq \hat{\varepsilon}$, but we also know that

$$\frac{\partial F(\hat{p})}{\partial p} = s(\hat{p}) - 1.$$

Thus we see that $|s(\hat{p})| \leq \hat{\varepsilon}$ where $s(\hat{p}) = \sum_i x_i(\hat{p})$. Now if we consider $x' = x(\hat{p})/s(\hat{p})$ we see that $\sum_i x_i' = 1$ and $|x_i' - x_i(\hat{p})| \leq 2\hat{\varepsilon}$. Thus $|L(\mathbf{x}', \hat{p}) - L(\hat{\mathbf{x}}, \hat{p})| \leq 2n\hat{\varepsilon}$ for $\hat{\varepsilon} \leq 1/2$, so $x_i^- \leq x_i' \leq x_i^+$, completing the proof of the lemma. $\diamond$

Combining the two lemmas completes the proof of the theorem. $\square$

4.2 Two or more resources

The construction of a price mechanism for two or more resources follows the basic ideas of the mechanism for a single resource case. Stage 1 is the same as previously but with two differences. In stage 1 the center computes the derivative using finite

differences of two nearby prices. In higher dimensions this idea must be used to compute an approximation of the gradient. This is much more difficult and we must settle for a randomized method of computing an approximation that has only a small chance of failure [26]. The second difference is that in the price mechanism for a single resource we used a finite accuracy variant of the bisection method for several purposes since all searches were one dimensional. In this case we must do similar searches in two or more dimensions. Therefore we must replace the simple bisection method with an ellipsoidal algorithm for convex programming. This is the same algorithm used for the ellipsoid mechanism, but in this case it operates on the price space, which is much smaller (dimension r) than the primal space (dimension nr) on which the ellipsoid mechanism operates.

The construction of stage 2 is much more complicated than in the simple case. In one dimension it is straightforward to construct two solutions who's convex hull contains a feasible solution. In higher dimensions this is much more difficult. For the case of two resources we have constructed a method which solves this problem. We believe that a generalization of the method should work for an arbitrary number of resources but have no proof of this conjecture.

In the following discussion we describe the construction of a price based mechanism for two or more resources which maximizes $U(\mathbf{x})$ over $0 \leq \mathbf{x} \leq \vec{1}$. As the proofs for stage 1 are not any simpler for two resources than for the general case, we will give them for the case of an arbitrary number of resources. However for stage 2 we specialize to the two resources case and only briefly comment on a possible extension to cases with more resources at the end.

In stage 1 the mechanism computes a $\hat{\mathbf{p}}$ that is $\hat{\varepsilon}$ accurate for the minimum of $F(\mathbf{p})$, where

$$F(\mathbf{p}) = \max_{\mathbf{x}} L(\mathbf{x}, \mathbf{p}) = \max_{\mathbf{x}} U(\mathbf{x}) - P\left(\sum_i x_i - \vec{1}\right)$$

using $\hat{\varepsilon}^3$-approximate solutions by the processors to evaluate $F(\mathbf{p})$.

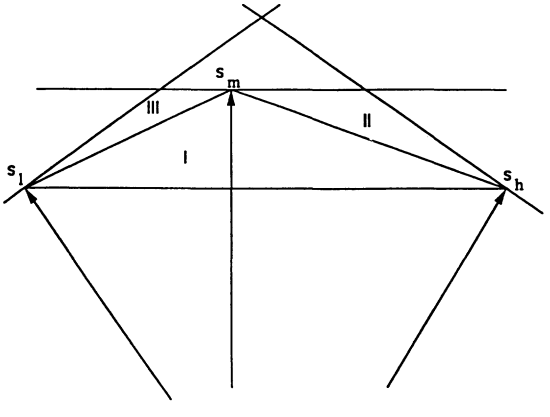

Figure 4. Geometry of stage 2 algorithm

Stage 2 iteratively computes a set of solutions to

$$\max \mathbf{c}^{\dagger}\mathbf{x} \text{ s.t. } |L(\mathbf{x}, \hat{\mathbf{p}}) - L(\hat{\mathbf{x}}, \hat{\mathbf{p}})| \leq \hat{\varepsilon}$$

who's convex hull contains a feasible solution (approximately). This is done by repeatedly constructing simplices which either contain a feasible solution or restrict the solution by pushing a face of the simplex up towards the solution. (See figure 4.) This is continued until either a feasible solution is found (by being contained in the convex hull of a simplex) or a face of the simplex is close enough to a feasible solution that we can just 'round' the solution to feasibility with only a small loss in accuracy.

This idea seems to generalize to the problem with an arbitrary number of resources. Again in this case we can use simplices to trap a feasible solution against the face of a simplex. Unfortunately, this does not guarantee that a straightforward rounding will work. However, we believe that by using several such simplices simultaneously we can successfully 'trap' the feasible solution, to allow for a rounding step to succeed.

As the mechanism for two or more resources is significantly more involved than that for one resource, we will describe the programs more descriptively and less formally than in the previous sections.

Note that this mechanism has a parameter δ which must be chosen in advance. This is the probability that the mechanism will fail, due to 'unlucky' approximations for the gradient of $F(\mathbf{p})$. The running time only depends weakly, $\mathcal{O}(\log 1/\delta)$, on the choice of δ.

The price mechanism for two resources

Center's program—$\mathbf{m}_c^{\varepsilon,n,r}$(*price*)
Stage 1:

1. Construct the ellipsoid $\mathbf{x} = \vec{1}/2$ where $\vec{1} = (1, 1, \ldots, 1)$ and $\mathbf{B} = Diag\left(\frac{2}{\sqrt{r}}, \frac{2}{\sqrt{r}}, \ldots, \frac{2}{\sqrt{r}}\right)$.
2. Let $\hat{\varepsilon} = \varepsilon/4$.
3. Repeat $N_c = \lceil 4r^2 \log(1/\hat{\varepsilon}) \rceil$ times.
   (a) Construct an approximate gradient at $\mathbf{p}$ by choosing $\tilde{\mathbf{p}}$ at random from the box $K = \{\mathbf{p} \mid \|\mathbf{p} - \tilde{\mathbf{p}}\|_1 \leq T\}$ where $T = \dfrac{\varepsilon}{12n^3r}$. Now let

$$c = \sum_{i=1}^{r} \frac{U(\hat{\mathbf{x}} + \tau\mathbf{e}_i) - U(\hat{\mathbf{x}})}{\tau} \mathbf{e}_i$$

   where $\mathbf{e}_i$ is the unit vector in the i'th direction, and $\tau = \dfrac{\delta\varepsilon^3}{576n^7r^4 \log(1/\varepsilon)}$.
   (b) compute a new $\mathbf{p}$ and $\mathbf{B}$ using a finite precision variant of the ellipsoid method.
4. Continue.

Stage 2: (For $r = 2$.) (Initialize.)

1. Let $\mathbf{c}_h = (1,0)^\dagger$, $\mathbf{c}_l = (-1,0)^\dagger$.
2. For all i: transmit $\mathbf{c}_h$ and $\mathbf{c}_l$ to the processors.
3. For all i: receive $\mathbf{s}_h$ and $\mathbf{s}_l$ from the processors, where $\mathbf{s} = \sum_{i=1}^{n} \mathbf{x}_i \in \mathfrak{R}^2$.
4. Compute $\mathbf{v}$ such that $\mathbf{v}^\dagger(\mathbf{s}_l - \mathbf{s}_h) = 0$ and $(0,1)\cdot\mathbf{v} = \vec{1}$.
5. Let $\mathbf{v} = \mathbf{v}/\|\mathbf{v}\|_2$.
6. If $\mathbf{v}^\dagger\mathbf{s}_l < \hat{\varepsilon}$ then $\mathbf{c}_m = (0,1)^\dagger$.
7. Else if $\mathbf{v}^\dagger\mathbf{s}_l > \hat{\varepsilon}$ then $\mathbf{c}_m = (0,-1)^\dagger$.
8. Else Goto ROUND.
9. Let $\mathbf{d} = \mathbf{c}_m$. (This will define the outward direction.)
10. Repeat $16\log(1/\hat{\varepsilon})$ times. (Trap the feasible solution.)
    (a) For all i: transmit $\mathbf{c}_m$ to all processors.
    (b) For all i: receive $\mathbf{s}_m$ from all processors.
    (c) If $\vec{1}$ is in the convex hull of $\mathbf{s}_l, \mathbf{s}_m, \mathbf{s}_h$ then goto FOUND.
    (We have found a solution that is feasible.)
    (d) Compute $\mathbf{v}$ such that $\mathbf{v}^\dagger(\mathbf{s}_l - \mathbf{s}_m) = 0$ and $\mathbf{d}^\dagger\mathbf{v} = 1$.
    (e) Let $\mathbf{v} = \mathbf{v}/\|\mathbf{v}\|_2$.
    (f) If $-\hat{\varepsilon} < \mathbf{v}^\dagger(\mathbf{s}_l - \vec{1}) < 0$ then set $\mathbf{s}_h = \mathbf{s}_m$ and goto ROUND.
    (We have found a solution that is nearly feasible.)
    (g) Else if $\mathbf{v}^\dagger(\mathbf{s}_l - \vec{1}) < 0$ then $\mathbf{c}_h = \mathbf{c}_m$ and goto CONT.
    (h) Compute $\mathbf{v}$ such that $\mathbf{v}^\dagger(\mathbf{s}_h - \mathbf{s}_m) = 0$ and $\mathbf{d}^\dagger\mathbf{v} = 1$.
    (i) Let $\mathbf{v} = \mathbf{v}/\|\mathbf{v}\|_2$.
    (j) If $-\hat{\varepsilon} < \mathbf{v}^\dagger(\mathbf{s}_h - \vec{1}) < 0$ then set $\mathbf{s}_l = \mathbf{s}_m$ and goto ROUND.
    (k) Else let $\mathbf{c}_l = \mathbf{c}_m$.
    (l) CONT: Let $\mathbf{c}_m = (\mathbf{c}_l + \mathbf{c}_h)/\|\mathbf{c}_l + \mathbf{c}_h\|_2$.
    (This computes a new $\mathbf{c}$ that is 'midway' between the two old c's.)
11. ROUND: Compute $0 \le \lambda \le 1$ such that $\|\lambda\mathbf{s}_l + (1-\lambda)\mathbf{s}_h\|_2 = \mathbf{v}^\dagger\mathbf{s}_l$.
12. Transmit λ, $\mathbf{v}^\dagger\mathbf{s}_l$, $\mathbf{c}_l$, $\mathbf{c}_h$ to all processors.
13. STOP.
14. FOUND: Compute $\lambda_l, \lambda_m, \lambda_h$ such that $\lambda_l\mathbf{s}_l + \lambda_m\mathbf{s}_m + \lambda_h\mathbf{s}_h = \vec{1}$.
15. Transmit $\lambda_l, \lambda_m, \lambda_h$ and $\mathbf{c}_l, \mathbf{c}_m, \mathbf{c}_h$ to all processors.
16. STOP.

Processor's program—$\mathbf{m}_i^{\varepsilon,n,r}(price)$

Stage 1:

1. Receive $\mathbf{p}$.
2. Construct the ellipsoid $\mathbf{x}_i = \vec{1}/2$ and $\mathbf{B}_i = Diag\left(\frac{2}{\sqrt{r}}, \frac{2}{\sqrt{r}}, \ldots, \frac{2}{\sqrt{r}}\right)$.
3. Repeat $N_c = \lceil 32r^2\log(nr/(\hat{\varepsilon}\delta))\rceil$ times.
   (Compute an ε-accurate consumption bundle for price $\mathbf{p}$.)
   (a) Compute the gradient at $\mathbf{x}_i$.
   (b) Compute a new $\mathbf{x}_i$ and $\mathbf{B}$ using the ellipsoid method for solving $L_i(\mathbf{x}_i, \mathbf{p})$.
   (c) Let $l_{max} = \max[l_{max}, L_i(\mathbf{x}_i, \mathbf{p})]$.
4. Transmit l_{max}.
5. Go to 1.

Stage 2:

1. Receive **c** from center.
2. Construct the ellipsoid $\mathbf{x}_i = \vec{1}/2$ and $\mathbf{B}_i = Diag\left(\frac{2}{\sqrt{r}}, \frac{2}{\sqrt{r}}, \ldots, \frac{2}{\sqrt{r}}\right)$.
3. Repeat $N_c = \lceil 4r^2 \log(n/\hat{\varepsilon}^3) \rceil$ times.
   (Compute the largest $\mathbf{c}^\dagger \mathbf{x}_i$ which is an ε-accurate consumption bundle for price **p**.)
   (a) If $L_i(\mathbf{x}_i, \mathbf{p})$ is sufficiently good then use $\mathbf{v} = \mathbf{c}$.
   (b) Else let $\mathbf{v} = \nabla L_i(\mathbf{x}_i, \mathbf{p})$.
   (c) Compute a new $\mathbf{x}_i$ and **B** using **v**.
4. Transmit $\mathbf{x}_i$.
5. Go to 1.
6. ROUND: (Round $\mathbf{x}_i$ to get a feasible solution.)
7. Receive λ, $\mathbf{v}^\dagger \mathbf{s}_l$, $\mathbf{c}_l$, $\mathbf{c}_h$ from the center.
8. Compute $\mathbf{x}_i = (\lambda \mathbf{x}_i(\mathbf{c}_l) + (1 - \lambda)\mathbf{x}_i(\mathbf{c}_h))\mathbf{v}^\dagger \mathbf{s}_l$, where $\mathbf{x}_i(\mathbf{c})$ is computed as in the above loop.
9. STOP.
10. FOUND: (Form $\mathbf{x}_i$ as a convex combination of $\mathbf{x}_i(c_l)$, $\mathbf{x}_i(c_m)$, and $\mathbf{x}_i(c_h)$.)
11. Receive λ_l, λ_m, λ_h and $\mathbf{c}_l$, $\mathbf{c}_m$, $\mathbf{c}_h$ from the center.
12. Compute $\mathbf{x}_i = \lambda_l \mathbf{x}_i(\mathbf{c}_l) + \lambda_m \mathbf{x}_i(\mathbf{c}_m) + \lambda_h \mathbf{x}_i(\mathbf{c}_h)$.
13. STOP.

Theorem 4 *For any* $\delta \in (0, \varepsilon/n)$ *the two resource price mechanism* $\mathbf{m}^{\varepsilon,n,r}(price)$ *has complexity*

$$T = \mathcal{O}\left(\left(\log \frac{n}{\varepsilon\delta}\right)[X(1 + \log(n/\varepsilon)) + I + \Omega \log(n/\varepsilon)]\right)$$

and will produce an ε*-accurate solution with probability greater than* $1 - \delta$.

Proof: The proof is given in the appendix.

For multiple resources the main ideas for the two resources case easily generalize. However, we have not completed the construction and proof for stage 2 and leave this for future work.

Conjecture 1 *For any* $\delta \in (0, \varepsilon/n)$ *the r resource price mechanism* $\mathbf{m}^{\varepsilon,n,r}(price)$ *has complexity*

$$T = \mathcal{O}\left(r^3\left(\log \frac{nr}{\varepsilon\delta}\right)\left[X\left(1 + \frac{\log 1/\varepsilon}{n} + \frac{r^2}{n} + r^4 \log(n/\varepsilon)\right) + Ir^2 + \Omega r^2 \log(n/\varepsilon)\right]\right)$$

and will produce an ε*-accurate solution with probability greater than* $1 - \delta$.

Proof (partial): The complexity is computed assuming that stage 1 dominates stage 2. The correctness of stage 1 is shown in the proof for the two resources case. The construction of stage 2 is as yet incomplete, However, we believe that a method similar to that for two resources should work. ◇

Thus we have constructed a polynomial-efficient mechanism for two resources, which should be extendible to more than two resources.

5 Resource allocation with externalities

In many instances the utility a processor derives from a certain bundle of resources is dependent on the resources used by the other processors. Two classic examples of this are pollution and congestion. These are both negative externalities, as the increased total use of resources reduces the processors derived utility.

Positive externalities are also possible. For example the usefulness of a communication network may depend on the number of people using it, assuming that it is operating below capacity. In this case if many people use the network then more people are reachable on the network thus increasing its value. (See e.g. [27].) However, in this paper we will only consider negative externalities, as they are the most common in resource allocation and distributed computing.

To account for externalities, we modify our formulation slightly, and assume negative externalities of a simple and common form. We allow the utility functions to depend on $\lambda = (\lambda^1, \ldots, \lambda^r)$ where $\lambda^j = \sum_{i=1}^n x_i^j$. So

$$U(\mathbf{x}, \lambda) = \sum_{i=1}^{n} U_i(\mathbf{x}_i, \lambda).$$

We require that $U(x, \lambda)$ is concave and monotonically decreasing in λ. Since λ is a negative externality, we can be less restrictive about the definition of λ, by viewing it as just another variable subject to $\lambda^j \geq \Sigma_{i=1}^n x_i^j$, as the optimal solution will always occur when the constraint is binding.

We can modify both the ellipsoid and descent mechanisms quite simply, to solve the resource allocation with externalities. We note that the transformation $\hat{\lambda} = 1 - \lambda$ allows us to rewrite

$$F' = \left\{ \mathbf{x} | 0 \leq \mathbf{x}, 0 \leq \hat{\lambda}, \sum_{i=1}^{n} x_i^j + \hat{\lambda}^j \leq 1 \right\}$$

and note that F' is equivalent to F with the addition of r new variables. Thus this new problem is almost identical structurally to the problem without externalities. A slight modification of these mechanisms allows them to solve the new problem with negligible loss of efficiency.

In fact, any mechanism (including the price mechanism) that solves the resource allocation problem can be used to solve the problem with externalities. Let $T(\varepsilon, r, n)$ be the time required by an arbitrary polynomial-efficient mechanism that solves the resource allocation problem without externalities. Note that this time is unchanged if we solve the generalized problem, but fix the externality effect to a certain level, say $\lambda_0 \leq 1$. Letting

$$V(\lambda) = \max_{\mathbf{x} \in F} U(\mathbf{x}, \lambda) | \lambda = \lambda_0$$

we can apply a modified version of the ellipsoid method to maximize $V(\lambda)$. (Note that $V(\cdot)$ is convex.) This allows us to show:

Theorem 5 *Let* $\mathbf{m}^{\varepsilon,n,r}$ *be an polynomial-efficient mechanism for allocating resources without externalities. Then there exists a related mechanism* $\mathbf{m}_{ext}^{\varepsilon,n,r}$ *which solves the problem with externalities in*

$$T_{externalities}(\varepsilon, r, n) = \mathcal{O}(r^3 \log(1/\varepsilon)(T(\varepsilon, r, n) + r))$$

where $T(\varepsilon, r, n)$ *is the running time of* $\mathbf{m}^{\varepsilon,n,r}$.

Proof: There exists a modified version of the ellipsoid mechanism which can maximize $V(\lambda)$ over $\{\lambda | 0 \leq \lambda^j \leq 1\}$ using only $\mathcal{O}(r^2 \log 1/\varepsilon)$ iterations and $2r + 1$ $\mathcal{O}(\varepsilon^4)$-accurate function values of $V(\lambda)$ with only $\mathcal{O}(r^2)$ computations per iteration. This is shown in [26]. Allowing the center to use the algorithm shows that

$$T_{externalities}(\mathbf{m}_{ext}^{\varepsilon,n,r}) = r^3 \log(1/\varepsilon)[T(\varepsilon^3, r, n) + r]$$

and since the mechanism is assumed to be polynomial-efficient it must have a running time which is polynomial in $\log 1/\varepsilon$. Therefore the ε^4 in the complexity can be replaced by ε as this can only change the running time by a multiplicative constant, since $\log \varepsilon^{-3} = 3 \log \varepsilon^{-1}$. □

Thus externalities do not increase the mechanism's dependence on n significantly or decrease its convergence rate.

6 Conclusions

We have described a general framework for constructing and comparing mechanisms for distributed resource allocation under bounded rationality, and developed a rigorous computational model of resource allocation with realistic processors. This is in contrast to the large body of economic work where processors are given unbounded computational ability. Also our measure of complexity is much more relevant than the standard definitions of complexity, such as the size of message space. Using this model we have still been able to design polynomial-efficient[8] mechanisms for resource allocation and compute their efficiency.

Another major contribution of our work is explicitly considering mechanisms with a large number of processors. Many of the mechanisms in the literature either implicitly or explicitly apply to a very small group of processors, often two or three. In contrast, we are interested in mechanisms that apply to large or very large groups of processors. For resource allocation this is an important distinction, as allocating resources to a small group of processors is relatively easy, and the differences between different mechanisms is probably unimportant.

Thus our major contribution to economic theory comes from raising these two issues, computation and number of processors, and by developing a methods to design mechanisms in this framework.

[8] In fact, these mechanisms are nearly optimal, in the sense that it is impossible for any mechanism to have significantly lower complexity. This is discussed in detail in [6].

A Appendix: Proof of Theorem 4

In this section we give the proof of Theorem 4. For stage 1 of the mechanism we give the proof for an arbitrary number of resources as this is notationally cleaner and requires no extra effort. For stage 2 we explicitly consider the case of two resources. In this part of the proof the argument requires this restriction. However, we believe that this should generalize in a natural manner.

A.1 Stage 1

First we describe the basic concepts required by the proof and set some of the notation. Duality theory of convex programming [20] allows us to rewrite the problem in the following manner. Define $\mathbf{s}(\mathbf{x}) = \sum_i \mathbf{x}_i - 1$. Then let

$$L(\mathbf{x}, \mathbf{p}) = U(\mathbf{x}) - \mathbf{p}^\dagger \mathbf{s}(\mathbf{x})$$

then define

$$F(\mathbf{p}) = \max_{\mathbf{x}} L(\mathbf{x}, \mathbf{p})$$

which is a convex function of $\mathbf{p}$ [20] and define $\mathbf{x}(\mathbf{p})$ to be the allocation that achieves the maximum.

Now the basic theory of convex duality says that the minimum over $\mathbf{p}$ of $L(\mathbf{p})$ is equal to the maximum of the allocation problem. Also, if $\mathbf{p}^*$ is the price that achieves the maximum then $\mathbf{x}(\mathbf{p}^*) = \mathbf{x}^*$ is the optimal allocation.

Using this we discover a natural distributed method for the allocation problem. Define

$$L_i(\mathbf{x}_i, \mathbf{p}) = U_i(\mathbf{x}) - \mathbf{p}^\dagger(\mathbf{x}_i - 1/n)$$

so that $L(\mathbf{x}, \mathbf{p}) = \sum_{i=1}^n L_i(\mathbf{x}_i, \mathbf{p})$. Define $F_i(\mathbf{p})$ similarly. Then given a $\mathbf{p}$ we can allow each processor to solve its own

$$\max_{\mathbf{x}_i} L_i(\mathbf{x}_i, \mathbf{p}).$$

Then a central processor can solve the low dimensional problem of finding the minimum of $F(\mathbf{p})$.

The first major difficulty which we must overcome is that of approximating gradients with only information about the functional value of $F(\mathbf{p})$. Deterministically this is very difficult as the function can have many 'kinks' where the value of the gradient can change abruptly. However, if we choose the places to evaluate the function at random, then we can get a useful approximation of the gradient with only a small number of query points.

Lemma 3 *For any $\delta \in (0, \hat{\varepsilon})$ the procedure for approximating gradients is $\hat{\varepsilon}$-accurate with probability greater than $1 - \alpha$ where $\alpha = \delta/(4r^2 \log 1/\varepsilon)$.*

Proof: See [26]. ◇

Lemma 4 *For any $\delta \in (0, \hat{\varepsilon})$ the ellipsoid method in stage one will fail with probability less than δ.*

Proof: The probability of error at each iteration is less than α. As there are $r^2 \log(1/\hat{\varepsilon})$ iterations, the probability of having a failure at any stage during the entire process is less than $r^2 \log(1/\hat{\varepsilon})\alpha = \delta$. ◇

Using the gradient computed by the method in the previous section, we can modify the ellipsoid algorithm slightly to still produce an $\hat{\varepsilon}$-accurate answer. An elementary description of this can be found in [28].

Thus if we compute the values of $F(\mathbf{p})$ to accuracy $\hat{\varepsilon}^3$ then the ellipsoid method described above computes an $\hat{\varepsilon}$-accurate solution $\hat{\mathbf{p}}, \hat{\mathbf{x}}$ such that $|F(\hat{\mathbf{p}}) - F(\mathbf{p}^*)| \leq \hat{\varepsilon}$ and $|L(\mathbf{x}(\hat{\mathbf{p}}), \hat{\mathbf{p}}) - L(\hat{\mathbf{x}}, \hat{\mathbf{p}})| \leq \hat{\varepsilon}$. Thus stage one can compute an $\hat{\varepsilon}$-approximation $\hat{\mathbf{p}}$ to the minimum of $F(\mathbf{p})$ and also an $\hat{\varepsilon}$-approximation to the actual value of $F(\hat{\mathbf{p}})$.

Thus we have shown:

Lemma 5 *For any δ, stage 1 produces a $\hat{\mathbf{p}}$ and $\hat{\mathbf{x}}$ such that*

$$|F(\mathbf{p}) - F(\mathbf{p}^*)| \leq \hat{\varepsilon}$$

and

$$|L(\hat{\mathbf{x}}, \hat{\mathbf{p}}) - F(\hat{\mathbf{p}})| \leq n\hat{\varepsilon}^4$$

and therefore

$$|F(\mathbf{p}^*) - L(\hat{\mathbf{x}}, \hat{\mathbf{p}})| \leq 2\hat{\varepsilon}$$

for $\hat{\varepsilon} \leq 1/2$, with probability greater than $1 - \delta$.

Proof: The first statement follows from the argument above on the accuracy of the ellipsoid method with the randomized algorithm for computing gradients. The second from the accuracy of the processor's own computation of $\mathbf{x}(\hat{\mathbf{p}})$, and the third from combining the first two. ◇

A.2 Stage 2

In this section we construct a feasible solution as the convex combination of several $\hat{\varepsilon}^2$-accurate solutions of

$$\max_{\mathbf{x}} \mathbf{c}^\dagger \mathbf{x} \text{ s.t. } \mathbf{x} \in G$$

where $G = G_1 \times \cdots \times G_n$ and

$$G_i = \{\mathbf{x} | L(\mathbf{x}_i, \hat{\mathbf{p}}) - L(\hat{\mathbf{x}}_i, \hat{\mathbf{p}}) \leq 4\hat{\varepsilon}\}.$$

This is computed using the ellipsoid method by the individual processors as the problem is separable into n independent pieces.

Theorem 6 *The computation of*

$$\max_{\mathbf{x}} \mathbf{c}^\dagger \mathbf{x}_i \text{ s.t. } \mathbf{x}_i \in G_i$$

to accuracy $\hat{\varepsilon}^2$ can be accomplished in $r^2 \log(r/2\hat{\varepsilon}^3)$ iterations of the ellipsoid method.

Proof: Note that G_i must contain the hypersphere of radius $\hat{\varepsilon}$ as the Lipschitz constant of $L_i(\mathbf{x}_i, \hat{\mathbf{p}})$ is less than 1. This hypersphere must contain a cube of side

$2\hat{\varepsilon}/\sqrt{r}$ and thus has volume greater than $(2\hat{\varepsilon}/\sqrt{r})^r$. The ellipsoid method requires $r^2 \log(1/(V^{1/r}\varepsilon)$ steps to compute an ε-accurate solution in a feasible convex region of volume V. $\diamond$

Now we show that there exists a feasible solution in G. This will follow from a sequence of lemmas.

Lemma 6 *Given p is an $\hat{\varepsilon}^2$ accurate solution of $F(p)$, there exists a p' such that $\|p - p'\|_2 \leq \hat{\varepsilon}$ and $\|\nabla F(p')\|_2 \leq \hat{\varepsilon}$.*

Proof: Consider the path generated by

$$\frac{d\gamma(t)}{dt} = \frac{\nabla F(\gamma(t))}{\|\nabla F(\gamma(t))\|_2} \quad \text{s.t.} \quad \gamma(0) = \mathbf{p}$$

which is a path from $\mathbf{p}$ to $\mathbf{p}^*$ where t is the arc length of the path from $\gamma(0)$ to $\gamma(t)$. Note that $\gamma(l) = \mathbf{p}^*$ where l is the length of the path γ.

Now

$$F(\hat{\mathbf{p}}) - F(\mathbf{p}^*) = \int_0^l \nabla F(\gamma(t)) \cdot \frac{d\gamma(t)}{dt}\, dt = \int_0^l \|\nabla F(\gamma(t))\|_2\, dt$$

which is just the line integral of ∇F.

By assumption $F(\hat{\mathbf{p}}) - F(\mathbf{p}^*) \leq \varepsilon^2$. Thus the following must hold

$$\int_0^{\hat{\varepsilon}} \|\nabla F(\gamma(x))\|_2\, d\gamma \leq \hat{\varepsilon}^2$$

as the integrand is non-negative.

Now assume that the theorem is false. This implies that for all $\mathbf{x} \leq \hat{\varepsilon}$ we must have $\|\nabla F(\gamma(\mathbf{x}))\|_2 > \hat{\varepsilon}$ as the Euclidean distance $\|\gamma(0) - \gamma(\hat{\varepsilon})\|_2$ is less than the distance along γ. Since

$$\int_0^{\hat{\varepsilon}} \|\nabla F(\gamma(x))\|_2\, d\gamma < \hat{\varepsilon} \min_{0 \leq \mathbf{x} \leq \hat{\varepsilon}} \|\nabla F(\gamma(x))\|_2$$

this implies that

$$\int_0^{\hat{\varepsilon}} \|\nabla F(\gamma(x))\|_2\, d\gamma > \hat{\varepsilon}^2$$

proving the theorem. $\diamond$

However, the gradient of F is related to feasibility.

Lemma 7 *The feasibility vector,* $\mathbf{s}(\mathbf{p}) - \vec{1} = \nabla F(\mathbf{p})$.

Proof: This follows from the well know envelope theorem,

$$\frac{dF(\mathbf{p})}{d\mathbf{p}} = \frac{\partial F(\mathbf{p})}{\partial \mathbf{p}} = s(\mathbf{p}) - \vec{1}. \quad \diamond$$

Using this we show that a feasible solution exists for the above optimization problem.

Lemma 8 *Given* $\mathbf{p}$ *is an* $\hat{\varepsilon}^2$ *accurate solution of* $F(\mathbf{p})$, *there exists an* $\mathbf{x}_i$ *such that* $|L(\hat{\mathbf{x}}_i, \hat{\mathbf{p}}) - L(\mathbf{x}_i, \hat{\mathbf{p}})| \leq 4\hat{\varepsilon}$ *and* $\mathbf{x}_i$ *is feasible.*

Proof: From the previous lemma there exists a $\mathbf{p}'$ such that $\| \mathbf{s}(\mathbf{p}') - \vec{1} \|_2 \leq \varepsilon$ then letting $\mathbf{x}^j = \mathbf{x}^j(\mathbf{p}')/\mathbf{s}^j(\mathbf{p}')$ gives an $\mathbf{x}$ that is feasible and noting that

$$\| \mathbf{x} - \mathbf{x}(\mathbf{p}') \|_2 \leq 2\hat{\varepsilon}$$

implies that

$$|L(\mathbf{x}_i(\mathbf{p}'), \mathbf{p}') - L(\mathbf{x}_i, \hat{\mathbf{p}})| \leq 2\hat{\varepsilon}$$

by the Lipschitz constant of L_i. Now by the Lipschitz constant of F_i for changing $\mathbf{p}$ we know that

$$|F_i(\hat{\mathbf{p}}) - F_i(\mathbf{p}')| \leq \hat{\varepsilon}.$$

Combining these inequalities produces the required result. ◇

Now that we have shown that a G contains at least one feasible solution we will describe how to construct one of these.

A.3 Constructing a feasible solution

The easiest way to understand the stage 2 algorithm is to consider the map

$$\mathbf{s}: S^1 \to \Re^r$$

defined by

$$\mathbf{s}_i(\theta) = \text{argmax}_{\mathbf{x}_i} (\cos\theta, \sin\theta)^\dagger \mathbf{x}_i \text{ s.t. } \mathbf{x} \in G$$

and $\mathbf{s}(\theta) = \sum_i \mathbf{s}_i(\theta)$. Note that in the mechanism we work directly with $\mathbf{c} = (\cos\theta, \sin\theta)$ instead of θ. For the purposes of the proof it is much clearer to work in θ directly.

The mechanism works in the following manner. The feasible point $\vec{1}$ must lie on one side of the line between $\mathbf{s}_l$ and $\mathbf{s}_h$. We choose θ_m such that $\mathbf{s}_m$ is on the same side of the line that $\vec{1}$ is. Now these three points create three possible regions in which $\vec{1}$ can lie. (See figure 4.) If the point lies in the convex hull of $\mathbf{s}_l, \mathbf{s}_h, \mathbf{s}_m$ (region I) then we stop. Also if the point lies close to one of the lines $\overline{\mathbf{s}_l\mathbf{s}_m}$ or $\overline{\mathbf{s}_h\mathbf{s}_m}$ then we also stop. However if the point lies in one of the remaining regions (regions II or III) then we choose the two points that define that region to be our new θ_l and θ_h and continue the process.

In order to prove that the mechanism constructs a feasible solution we note that Stage 2 actually performs a binary search in θ, and the only ways it can halt are:

1. A feasible solution $\mathbf{x}$ is found as the convex combination of 3 points $\mathbf{x}_l, \mathbf{x}_m, x_h$.
2. A solution $\mathbf{x}$ with $\| \mathbf{s}(\mathbf{x}) - \vec{1} \|_2$ is the convex combination of two points $\mathbf{x}_l, \mathbf{x}_h$.
3. $\theta_l - \theta_h \leq 2\pi/\hat{\varepsilon}$.

In the first case the following lemma shows that the convex combination of $\mathbf{x}_l, \mathbf{x}_m, \mathbf{x}_h$ which gives a feasible solution that is $\hat{\varepsilon}$-accurate.

Lemma 9 *Let* $\mathbf{u}, \mathbf{v}, \mathbf{w}$ *all satisfy* $|L(\mathbf{u}, \hat{\mathbf{p}}) - F(\hat{\mathbf{p}})| \leq 4\hat{\varepsilon}$ (*resp.* $\mathbf{v}, \mathbf{w}$). *Assume that* $\mathbf{x} = \lambda_u \mathbf{u} + \lambda_v \mathbf{v} + \lambda_w \mathbf{w}$ *with* $0 \leq \lambda_u, \lambda_v, \lambda_w$ *and* $\lambda_u + \lambda_v + \lambda_w = \vec{1}$. *Then* $|L(\mathbf{x}, \hat{\mathbf{p}}) - F(\hat{\mathbf{p}})| \leq 4\hat{\varepsilon}$.

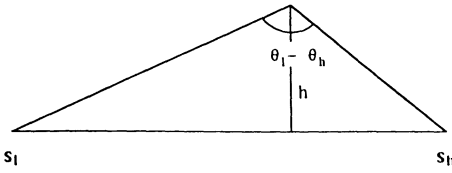

Figure 5. Accuracy of stage 2

Proof: By convexity of L we have that $L(\mathbf{x}, \hat{\mathbf{p}}) \leq \max[L(\mathbf{u}, \hat{\mathbf{p}}), L(\mathbf{v}, \hat{\mathbf{p}}), L(\mathbf{w}, \hat{\mathbf{p}})]$. However by definition $F(\hat{\mathbf{p}}) > L(\mathbf{x}, \hat{\mathbf{p}})$. ◇

From the previous lemma we see that for case 2 we can simply round the solution to feasibility.

Lemma 10 *In case two the allocation defined by* $\mathbf{x}^j = \mathbf{x}^j/\mathbf{s}^j(\mathbf{x})$ *satisfies* $|L(\mathbf{x}, \hat{\mathbf{p}}) - F(\hat{\mathbf{p}})| \leq 2\hat{\varepsilon}$.

Proof: Note that $\mathbf{x}_j$ is feasible. Also $\|\mathbf{x}' - \mathbf{x}\|_2 \leq \varepsilon$ as $\|\mathbf{s}(\mathbf{x}) - \vec{1}\|_2 \leq \hat{\varepsilon}$. Thus by the Lipschitz continuity of L we get the desired result. ◇

Now we will show that case 3 is actually equivalent to case 2 as $|\theta_l - \theta_h| \leq 2\pi\hat{\varepsilon}$ implies that there exists a solution $\mathbf{x}$ which is the convex combination of $\mathbf{x}(\theta_l)$ and $\mathbf{x}(\theta_h)$ and $\|\mathbf{s}(\mathbf{x}) - \vec{1}\|_2 \leq \hat{\varepsilon}$.

Lemma 11 *Assume that the center's stage 2 algorithm received the exact solution* $\mathbf{s}(\theta)$ *at each iteration. Then if* $|\theta_l - \theta_h| \leq \hat{\varepsilon}$ *then* $\vec{1}$ *must be within* $\hat{\varepsilon}$ *of the line* $\overline{\mathbf{s}_l\mathbf{s}_h}$.

Proof: This follows from elementary geometry and the fact that the feasible solution $\vec{1}$ must be contained in the region bounded by $\overline{\mathbf{s}_l\mathbf{s}_h}$ by construction. Also we know that $\mathbf{c}(\theta_l)^\dagger\mathbf{s}_l \geq \mathbf{c}(\theta_l)^\dagger\vec{1}$ and $\mathbf{c}(\theta_m)^\dagger\mathbf{s}_m \geq \mathbf{c}(\theta_m)^\dagger\vec{1}$ as these $(\mathbf{s}_l, \mathbf{s}_h)$ are the respective maxima of their respective $\mathbf{c}$'s. This region forms a triangle with angle $\alpha = \pi - \hat{\varepsilon}$ as $\hat{\varepsilon} = |\theta_h - \theta_l|$. (See figure 5.)

Since we know that $\|\mathbf{s}_l - \mathbf{s}_h\|_2 \leq 2$ the height of the triangle must be less than $2 \sin \hat{\varepsilon}/2$ which is less than $\hat{\varepsilon}$. Thus as the point $\vec{1}$ is contained within this triangle, we see that the distance from $\vec{1}$ to $\overline{\mathbf{s}_l\mathbf{s}_h}$ must be less than $\hat{\varepsilon}$. ◇

However, the center does not receive exact solutions from the processors. The solutions from the processors are only accurate to $\hat{\varepsilon}^2$. In fact the solutions $\hat{\mathbf{s}}(\theta)$ can be arbitrarily far from the true solutions $\mathbf{s}(\theta)$. However, as the next lemma shows, these solutions are actually close to the real solution for a slightly different θ. Thus we can imagine that the values of θ are uncertain. However, this does not effect the binary search if these errors in θ are less than $\hat{\varepsilon}$.

Lemma 12 *Assume that* $|\mathbf{c}(\theta)^\dagger(\mathbf{s} - \mathbf{s}(\theta))| \leq \hat{\varepsilon}^2$. *Then there exists a* θ' *such that* $|\theta - \theta'| \leq 2\pi\hat{\varepsilon}$ *and* $\|\mathbf{s} - \mathbf{s}(\theta')\|_2 \leq \hat{\varepsilon}$.

Proof: As $|\mathbf{c}(\theta)^\dagger(\mathbf{s} - \mathbf{s}(\theta))| \leq \hat{\varepsilon}^2$ we see that there exists a $\hat{t}$ such that $\|\mathbf{s} - \mathbf{s}(\hat{t})\|_2 \leq \hat{\varepsilon}$. Let $\gamma(\mathbf{x})$ be the boundary of the feasible region with $\gamma(0) = \mathbf{s}(\theta)$ and $\gamma(\hat{t}) = \mathbf{s}(\hat{t})$ and assume that γ is parametrized by arc length.

We will now show that there exists a θ' such that $|\theta' - \theta| \leq \hat{\varepsilon}$ while $\|\mathbf{s}(\theta') - \mathbf{s}(\hat{t})\|_2 \leq \hat{\varepsilon}$. Combining these inequalities produces the theorem.

First note that at $\mathbf{s}(\theta)$ the normal to γ must point in the direction defined by θ. Note that

$$\mathbf{c}(\theta)^{\dagger}(\mathbf{s}(\theta)-\mathbf{s}(\beta))=\int_0^{\mathbf{x}(\beta)}\tan(\theta(u)-\theta)du\leq\int_0^{\mathbf{x}(\beta)}2/\pi(\theta(u)-\theta)du$$

where $\mathbf{x}(\beta)$ is the smallest $\mathbf{x}$ such that $\theta(\mathbf{x})\leq\beta$.

Now assume the lemma is false and all $\theta(\mathbf{x})$ for $|\mathbf{x}-\mathbf{x}(\hat{t})|$ satisfy $|\theta(\mathbf{x})-\theta|>\hat{\varepsilon}$, therefore

$$\mathbf{c}(\theta)^{\dagger}(\mathbf{s}(\theta(\mathbf{x}))-\mathbf{s}(\theta))>\hat{\varepsilon}^2$$

by the previous equation. However this contradicts the fact that $\hat{\mathbf{s}}$ is $\hat{\varepsilon}^2$-accurate proving the lemma. ◇

Thus the algorithm works even with the inexact replies given by the processors.

□

References

1. Arrow, K., Hurwicz, L.: Decentralization and computation in resource allocation. In: Pfouts, R. (ed.) Essays in economics and econometrics. Chapel Hill: University of North Carolina Press, 1960
2. Blum, L.: Lectures on a theory of computation and complexity over the reals (or an arbitrary ring). Technical Report TR-89-065, International Computer Science Institute, Perkeley, CA, December 1989
3. Blum, L., Shub, M., Smale, S.: On a theory of computation over the real numbers: *NP*-completeness, recursive functions, and universal machines. Bull. ACM **21**, 1–46 (1988)
4. Ecker, J. G., Kupfershmid, J.: The ellipsoid algorithm for nonlinear programming. Math. Prog. **27**, 83–106 (1983)
5. Ferguson, D. W., Nikolau, C., Yemini, Y.: Microeconomic algorithms for load balancing in distributed computer systems. Proceedings of the 8th International Conference on Distributed Computing Systems, 1988
6. Friedman, E. J.: The complexity of allocating resources in parallel: upper and lower bounds. In: Pardalos, P. (ed.) Complexity in numerical optimization. Singapore: World Scientific 1993
7. Friedman, E. J.: A strongly polynomial algorithm for convex programs with combinatorial constraints and resource allocation. Technical report UCB/ERL/IGCT M92/6, Interdisciplinary Group on Coordination Theory, Electronics Research Laboratory, University of California, Berkeley, CA 94720, January 1992
8. Hochbaum, D. S.: Lower and upper bounds for the allocation problem and other nonlinear optimization problems. Manuscript, School of Business Administration and Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720. September 1989, Revised December 1990
9. Huberman, B.: The ecology of computation. Amsterdam: North Holland 1988
10. Hurwicz, L.: The design of mechanisms for resource allocation. Am. Econ. Rev. 1–30 (1973)
11. Hurwicz, L.: On informational decentralization and efficiency in resource allocation mechanism. In: Reiter, S. (ed.) Studies in mathematical economics. Washington D.C.: Mathematical Association of America 1986
12. Hurwicz, L.: On informationally decentralized systems. In: McGuire, C. B., Radner, R. (ed.) Decision and organization. Minneapolis: University of Minnesota Press 1986
13. Hurwicz, L.: On the dimensional requirements of informationally decentralized pareto satisfactory adjustment processes. In: Arrow, K. J., Hurwicz, L. (ed.) Studies in resource allocation processes. Cambridge, New York: Cambridge University Press 1977
14. Hurwicz, L., Marschak, T.: Discrete allocation mechanisms: dimensional requirements for resource-allocation mechanisms when the desired outcomes are unbounded. J. Complexity **1**, 264–303 (1985)

15. Ibaraki, T., Katoh, N.: Resource allocation problems: algorithmic approaches. MIT Press 1988
16. Khachian, L. G.: A polynomial algorithm for linear programming. Sov. Math. Doklady **20**, 191–4 (1979)
17. Kurose, J. F., Simha, R.: A microeconomic approach to optimal resource allocation in distributed computer systems. IEEE Trans. Comp. **38**, 705–16 (1989)
18. Leighton, F. T.: Introduction to parallel algorithms and architectures. Morgan-Kaufman 1991
19. Levin, A. Yu.: On an algorithm for minimizing convex functions. Sov. Maths. **1**, 286–90 (1965)
20. Luenberger, D. G.: Linear and nonlinear programming, 2nd edn. Reading, Massachusetts: Addison-Wesley Publishing Company 1984
21. Marschak, T. A.: Organizational design. In: Arrow, K. J., Intrilligator, M. D. (eds.) Handbook of mathematical economics. New York. North-Holland 1986
22. Megiddo, N.: Towards a genuinely polynomial algorithm for linear programming. Siam J. Comp. **12**, 347–59 (1983)
23. Mount, K., Reiter, S.: The existence of a locally stable dynamic process with a statically minimal message space. In: Groves, T., Radner, R., Reiter, S. (eds.) Information, incentives, and economic mechanisms: essays in honor of Leonid Hurwicz. Minneapolis: University of Minnesota Press 1987
24. Mount, K., Reiter, S.: The informational size of message spaces. J. Econ. Theory **8**, 161–192 (1974)
25. Mount, K., Reiter, S.: A model of computing with human agents 1990. Discussion Paper No. 890. Center for Mathematical Studies, J.L. Kellogg Graduate School of Management, Northwestern University, Evanston, Ilinois 60208
26. Nemirovsky, A. S., Yudin, D. B.: Problem complexity and method efficiency in optimization. New York: Wiley 1983
27. Oren, S. S., Smith, S.: Critical mass and tariffs in electronics communication markets. Bell J. Econ. **12**, 467–487 (1981)
28. Papadimitriou, C. H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1982
29. Radner, R.: The organization of decentralized information processing. Econometrica **61**, 1109–1146 (1993)
30. Rockafellar, R. T.: Convex analysis. Princeton, New Jersey: Princeton University Press 1970
31. Russell, S., Wefald, E.: Do the right thing: studies in limited rationality. Boston: MIT Press, 1991
32. Sanders, B.: An incentive compatible flow control algorithm for rate allocation in computer networks. IEEE Trans. Comp. **37**, 1062–72 (1988)
33. Sanders, B.: A private good/public good for optimal flow control of an m/m/1 queue. IEEE Trans. Autom. Contr. **30**, 1143–5 (1985)
34. Sapetnekar, S., Rao, V., Vaidya, P., Kang, S.: An exact solution for the transistor sizing problem for cmos circuits using convex optimization. Mimeo 1992
35. Scarf, H.: Some examples of global instability of the competitive equilibria. Int. Econ. Rev. **1**, 157–72 (1960)
36. Shenker, S.: Efficient network allocations with selfish users. In: King, P. J. B., Mitrani, I., Pooley, R. J. (eds.) Performance '90, pp. 279–285. New York: North-Holland 1990
37. Simon, H. A.: Theories of bounded rationality. In: McGuire, C. B., Radner, R. (eds.) Decision and organization. Minneapolis: University of Minnesota Press 1986
38. Smale, S.: A convergent process of price adjustment and global newton methods. J. Math. Econ. **3**, 107–20 (1976)
39. Vaidya, P. M.: A new algorithm for minimizing convex functions over convex sets. In: 30th Annual Symposium on the Foundations of Computer Science, pp. 332–337. IEEE, October 1989
40. Varian, H. R.: Microeconomic analysis. New York: Norton, W. W. 1978