Distributed Implementations of Vickrey-Clarke-Groves Mechanisms

Distributed Implementations of Vickrey-Clarke-Groves Mechanisms

David C. Parkes
Division of Engineering and Applied Sciences,
Harvard University,
33 Oxford Street, Cambridge MA 02138
parkes@eecs.harvard.edu

Jeffrey Shneidman
Division of Engineering and Applied Sciences,
Harvard University,
33 Oxford Street, Cambridge MA 02138
jeffsh@eecs.harvard.edu

Abstract

Mechanism design (MD) provides a useful method to implement outcomes with desirable properties in systems with self-interested computational agents. One drawback, however, is that computation is implicitly centralized in MD theory, with a central planner taking all decisions. We consider distributed implementations, in which the outcome is determined by the self-interested agents themselves. Clearly this introduces new opportunities for manipulation. We propose a number of principles to guide the distribution of computation, focusing in particular on Vickrey-Clarke-Groves mechanisms for implementing outcomes that maximize total value across agents. Our solutions bring the complete implementation into an ex post *Nash equilibrium.*

1. Introduction

Mechanism design [18] is concerned with the design of procedures to implement an outcome with desirable properties in systems with self-interested agents that have private information about their preferences and capabilities. Mechanism design has largely focused on a special class of mechanisms in which the computation required to determine the outcome is completely centralized.

These are the *direct-revelation* mechanisms, in which each agent reports its private information to a center that computes the outcome and reports the solution back to the agents. We introduce the fundamentally new problem of *distributed implementation*, in which the goal is to use the same self-interested agents to determine the outcome.

It has now been over 10 years since the first infusion of ideas from mechanism design into distributed AI [12, 27]. Mechanism design has been adopted in many settings, for instance for determining a shared plan of action [16], for the allocation of shared resources [34, 26], or for structuring negotiation between agents [28]. Our hope is that the Distributed Implementation problem will facilitate the integration of methods for cooperative problem solving in Distributed AI with the methods to handle self-interest in computational mechanism design. Indeed, Lesser [19] recently described this unification, of methods in cooperative methods with self-interested methods, as one of the major challenges for multiagent systems research.

The distributed implementation of mechanisms introduces new opportunities for agent manipulation. For instance, consider distributing the winner-determination of a second-price auction across bidding agents. Clearly each agent would like to understate the maximal value of bids from other agents to increase its chance of winning and to decrease its payment.

A distributed implementation provides each agent with an *algorithm* (or a specification of an algorithm). A successful (or *faithful*) distributed implementation must provide the right incentives, so that an agent will *choose* to follow the intended algorithm. We seek implementation in an *ex post* Nash equilibrium, such that no agent can usefully deviate from its algorithm even if it knows the private values of other agents.

All our observations in this paper are quite simple, but we think quite powerful. We provide three general *principles* for distributed implementation: *partition-based*, in which computation is carefully distributed across agents; *information-revelation based*, in which agents only perform restricted computation, as necessary to reveal information about their local private information; and *redundancy-based*, in which multiple agents are asked to perform the same piece of computation, with deviations punished. We will often draw on examples and motivation from the Vickrey-Clarke-Groves mechanism, but the ideas are more general. We include stylized examples to illustrate how to combine existing algorithmic paradigms from cooperative problem solving with the principles for faithful distributed implementation.

1.1. Related Work

Feigenbaum and colleagues [14, 13] initiated the study of *distributed algorithmic mechanism design* (DAMD), with a focus on studying particular communication topolo-

gies and providing distributed algorithms with good computational complexity and good communication properties. However, DAMD has deemphasized incentive issues, and does not consider whether an agent will *choose* to follow a particular algorithm.

Shneidman and Parkes [30] provided the seeds for this work, with an early definition of the concept of "algorithm compatibility." More recently, Shneidman and Parkes [29] have completed a careful case study of distributed implementation for interdomain routing, bringing an earlier algorithm due to Feigenbaum et al. [13] into equilibrium.

Monderer and Tennenholtz [23] have studied a simple single-item auction problem in which agents must forward messages from other agents to a center, using information hiding and redundancy to bring faithful forwarding into an equilibrium. We focus in this paper on a model in which agents can communicate with the center directly— on a trusted channel —thus removing this concern.

Smorodinsky and Tennenholtz [33] consider free-riding in multi-party computation by agents with costly computation, and provide incentives to elicit computational effort from agents. However their work does not take an *implementation* perspective, and there is no private information. Perhaps the closest work in the literature to ours is Brewer's "computation-procuring" auction [5], in which incentives are used to distribute winner-determination across participants in an ascending-price combinatorial auctions. Agents that can find and submit an improved solution are paid some share of the revenue improvement. Although Brewer does not provide formal equilibrium analysis, an experimental study suggests this "computation procuring" auction was effective in eliciting effort from human bidders. Similar ideas can also be traced to the use of the bid queue to store partial solutions in the AUSM mechanism [2], and (in a cooperative setting) to work on *computational ecosystems* [8].

Shoham and Tennenholtz [31, 32] have considered computation in a system of self-interested agents with private inputs. The agents are either reluctant to provide information, or want to know the result of computation but prefer to keep this from their peers. However, their goals are quite different. All computation is centralized, and the focus is on computation but not implementation (i.e., not on taking decisions in a world). The notion of an *Efficient Learning Equilibrium* [4] shares our idea of bringing *algorithms* into an equilibrium.

Combining redundancy with a commitment to implement a "bad" outcome if agents don't send the same message is well known in the literature on implementation in *complete information* settings— where every agent, but not the center, knows all private information —albeit for revealing (common) type information and not for eliciting effort (see Jackson [17] for a survey). However, agents still reveal full information to the center, and the center still determines the outcome of the social-choice rule (e.g. [24]). Multi-stage game forms are used to allow equilibrium refinements that knock-out undesirable equilibria, so that the outcome is implemented in *all* equilibria,[1] but not to facilitate distributed computation. Recent extensions have considered implementation with *incomplete* information, but still with centralized computation, and while adopting difficult solution concepts, for example *perfect Bayesian implementation* [6] and *sequential equilibrium* [1]).

2. Preliminaries

We first introduce notions from traditional (centralized) mechanism design. A more leisurely introduction to mechanism design is provided by Jackson [18] and Parkes [25, chapter 2]. Dash et al. [10] provide a recent multi-agent perspective on important challenges in the field of computational mechanism design.

2.1. Mechanism Design

The standard setting for mechanism design considers a world with possible outcomes $\mathcal{O}$, and agents $i \in \mathcal{I}$ (with N agents altogether). Agent i has private type $\theta_i \in \Theta_i$, which defines the agent's utility $u_i(o;\theta_i)$ for outcome $o \in \mathcal{O}$.

A standard (direct-revelation) mechanism $M = (f, \Theta)$ defines a procedure in which agents report types $\hat{\theta} \in \Theta = (\Theta_1 \times \ldots \times \Theta_N)$ and the mechanism rules select outcome $f(\hat{\theta})$. We write $\hat{\theta}$ to emphasize that agents can misreport their true types (which are not observable to the center).

A mechanism defines a non-cooperative game of incomplete information because agents do not know the types of other agents. Agent i's utility for report $\hat{\theta}_i$ given reports $\hat{\theta}_{-i} = (\hat{\theta}_1, \ldots, \hat{\theta}_{-i}, \hat{\theta}_{+i}, \ldots, \hat{\theta}_N)$ is $u_i(f(\hat{\theta}_i, \hat{\theta}_{-i}); \theta_i)$. An important concept in MD is that of *incentive-compatibility* (IC), which says that agents will choose to reveal their types truthfully in equilibrium. A mechanism that achieves truth-revelation in a *dominant-strategy* equilibrium (every agent's strategy is best-response whatever the strategies of other agents), is termed *strategyproof*, defined as:

$$u_i(f(\theta_i, \theta_{-i}); \theta_i) \geq u_i(f(\hat{\theta}_i, \theta_{-i}); \theta_i),\ \forall \theta_i, \forall \hat{\theta}_i \neq \theta_i, \forall \theta_{-i}$$

Strategyproof is particularly useful because agents do not need to model the other agents to play their best-response. Finally, an IC mechanism is said to *implement* outcome $f(\theta)$ in equilibrium; and $f(\theta)$ is the *social-choice function* implemented within the mechanism.

2.2. Vickrey-Clarke-Groves Mechanisms

In particular, consider a world in which the outcome $o = (k, p)$ defines both a choice $k \in \mathcal{K}$, for some dis-

1 We are less concerned with multiple equilibrium because the center in our model can also choose to incur some computational cost and check whether agents deviate. Also, we assume that the intended algorithm (implemented in software) helps to correlate agents on a desired equilibrium, providing a focal point (see also [4].

crete choice set $\mathcal{K}$, and a payment $p = (p_1, \ldots, p_N)$ by each agent to the center. For example, the choice could define a set of actions to be performed by agents as part of a plan, or an allocation of items. The type of an agent now defines its value $v_i(k; \theta_i)$ for a choice k, and its utility is quasi-linear in value and payments, defined as $u_i(o; \theta_i) = v_i(k; \theta_i) - p_i$.

In this setting, the Vickrey-Clarke-Groves (VCG) (see [18]) mechanism is strategyproof, and implements the social-welfare maximizing (or *efficient*) choice. We define *economy* E_N to include all agents, and *marginal economies* $\{E_{N-1}, E_{N-2}, \ldots\}$ as the economies with each agent removed in turn. The VCG defines choice rule $k^*(\hat{\theta}) = \arg\max_{k \in \mathcal{K}} \sum_i v_i(k; \hat{\theta}_i)$, and payment rule:

$$p_{\text{vcg},i}(\hat{\theta}) = v_i(k^*(\hat{\theta}); \hat{\theta}_i) - \{V_N - V_{N-i}\} \qquad (1)$$

where $V_N = \max_{k \in \mathcal{K}} \sum_i v_i(k; \hat{\theta}_i)$ and $V_{N-i} = \max_{k \in \mathcal{K}} \sum_{j \neq i} v_j(k; \hat{\theta}_j)$, i.e. the value of the efficient choice in the marginal economy without agent i.

3. Distributed Implementations

We now describe a *distributed implementation*, focusing on a setting in which there is still a center, ultimately responsible for selecting and enforcing an outcome. We will seek to off-load as much of the computation as possible onto the agents, but require that this computation is in an equilibrium. We assume that each agent can communicate directly through the center, via a trusted channel.[2]

The basic model of communication assumes *message-passing* between agents, and a *state-based model* for computation, with each agent maintaining an internal state, performing computation to modify that state, and sending messages that depend on state.[3]

A **distributed mechanism** $d_M = (g, \Sigma, s^m)$ defines an *outcome rule* g, a feasible *strategy space* $\Sigma = (\Sigma_1 \times \ldots \times \Sigma_N)$, and an intended (or "suggested") strategy $s^m = (s_1^m, \ldots, s_N^m)$. We also refer to s^m as the *intended implementation*. It is helpful to think of s^m as the algorithm that the designer would like every agent to follow. Given strategy $s \in \Sigma$, it is convenient to write $s(\theta)$ to denote the complete sequence of actions taken by agents when following joint strategy s, given private types θ.

The outcome rule g defines the outcome $g(s(\theta)) \in \mathcal{O}$, selected when agents follow strategy s and have types θ. The center selects outcome $g(s(\theta))$ based on information provided by agents during the course of the algorithm. Taken together, this defines a non-cooperative game.

A strategy $s_i \in \Sigma_i$ is a mapping from *state* and (private) type to an *action*. Actions may be *internal*, in which case they are *computational* actions, or *external*, in which case they are *message-sending* actions. An agent's local state includes its *computational state*, as well as a complete history of all messages ever received or sent by the agent and its model of other agents.

Definition 1 *Distributed mechanism $d_M = (g, \Sigma, s^m)$ is an (ex post)* ***faithful implementation*** *of social-choice function $g(s^m(\theta)) \in \mathcal{O}$ when intended algorithm s^m is an* ex post *Nash equilibrium.*

Formally, strategy profile $s^* = (s_1^*, \ldots, s_N^*)$ is an *ex post* Nash equilibrium when:

$$u_i(g(s_i^*(\theta_i), s_{-i}^*(\theta_{-i})); \theta_i) \geq u_i(g(s_i'(\theta_i), s_{-i}^*(\theta_{-i})); \theta_i)$$

for all agents, for all $s_i' \neq s_i^*$, for every type θ_i, and for all types θ_{-i} of other agents. In words, no agent would like to deviate from s_i^* even with knowledge of the private type information of the other agents. As a solution concept, *ex post* Nash relies on the rationality of other agents, but remains useful because an agent need not model the preferences of other agents.

Given distributed mechanism $d_M = (g, \Sigma, s^m)$, it is useful to categorize the external actions in the intended implementation into *message-passing*, *information-revelation*, and *computational* actions.

Definition 2 *External actions $a_e \in s_i^m(h, \theta_i)$ are* ***message-passing actions*** *when agent i simply forwards a message received from another agent, unchanged, to one (or more) of its neighbors.*

Message-passing actions are included to allow for peer-to-peer communication.

Definition 3 *External actions $a_e \in s_i^m(h, \theta_i)$ are* ***information-revelation*** *actions when any feasible deviation from these actions by agent i is entirely equivalent to following the intended implementation for some other reported type $\hat{\theta}_i$; i.e., $g(s', s_{-i}^m(\theta_{-i})) = g(s_i^m(\hat{\theta}_i), s_{-i}^m(\theta_{-i}))$, for all θ_{-i}, where s' differs from $s_i^m(\theta_i)$ only in these info-revelation actions.*

Informally, information-revelation actions can be executed by a "dumb" agent that only knows type θ_i and can only respond to questions about type, such as "is choice k_1 preferred to choice k_2?", "what is the value for choice k_1?", etc. By definition, the only role that these actions play in the implementation is in revealing private information.[4]

2 Shneidman & Parkes [29] consider a more general model with no center, and with self-enforcement of the final outcome by the agents.

3 The model can be formalized to make the games that we describe precise, for example introducing a start state and end state, and defining state-transition functions. Such a formalism is tangential to the main thrust of this paper, and will be avoided.

4 The definition carefully excludes actions in which useful computation is also "smuggled" within the message, for example "solve problem P_1 if your value is v_1 and solve problem P_2 if your value is v_2." This is precluded because there are presumably *arbitrary* deviations from computing the solution to P_1 or P_2, that are not performed in *any* intended implementation, for any private type.

Definition 4 *External actions $a_e \in s_i^m(h, \theta_i)$ are **computational actions** when they are neither message-passing nor information-revelation actions.*

Although this definition of computational actions is somewhat indirect, the point is that external actions (or messages) that we classify as *computational* are doing more than forwarding a message from another agent or revealing private information. Presumably, computational actions (if they have any use within the implementation) are sending results from local computation.

It is important to emphasize that we have only characterized the *external* actions. Computational agents are continually performing *internal* actions— computation —to support these external actions, and these actions (or at least a specification) are also defined in an intended strategy. For instance, an agent must perform (internal) computation in responding to an information-revelation action "which bundle of goods maximize your utility given prices p?".

We can now define the important notions of *incentive-compatibility* (IC), *communication compatibility* (CC), and *algorithm compatibility* (AC) in this context.

Definition 5 *Distributed mechanism d_M is CC $\{$ resp. IC, AC$\}$ if an agent cannot receive higher utility by deviating from the intended message-passing $\{$ resp. information-revelation, computational$\}$ actions in an equilibrium.*

CC, IC, and AC are required properties of a faithful distributed implementation. Moreover, a distributed mechanism $d_M = (g, \Sigma, s^m)$ that is IC, CC and AC is a *faithful implementation* of $g(s^m(\theta))$, when IC, CC and AC all hold in a *single* equilibrium.

Remark 1 The only social-choice functions that can be implemented in an *ex post* Nash distributed implementation are those implementable in *strategyproof* direct-revelation mechanisms (follows from the revelation principle [20].)

Remark 2 We assume that agents are self-interested but benevolent, in the sense that an agent will implement the intended strategy as long as it does not *strictly* prefer some other strategy. Thus, a *weak ex post* Nash equilibrium is sufficient for a faithful implementation. Further, a distributed mechanism may have multiple equilibria. We are content to achieve implementation in *one* of these equilibria, which is consistent with the mechanism design literature.

4. A Canonical Distributed Implementation

To illustrate why faithful distributed implementation can be difficult, and also to introduce a general class of distributed VCG mechanisms, consider the following *canonical distributed algorithm* for determining the efficient choice in economies $\{E_N, E_{N-1}, \ldots\}$:

(1) Every agent is asked to report its type $\hat{\theta}_i$ to the center. Upon receipt, the center broadcasts these types to the agents.

(2) Take your favorite distributed algorithm for computing the efficient choice, for instance:

(i) Distributed systematic search, such as Adopt [22], for solving constrained optimization problems.

(ii) Mathematical-programming based decompositions, such as Dantzig-Wolfe and column generation [15].

(iii) Asynchronous Cooperative Problem Solving with Shared Storage, such as *blackboard* models (see [7] for a recent summary) and hint-exchange models [8]).

and use this algorithm to define an intended strategy, s^m, to determine the efficient choice in each of $\{E_N, E_{N-1}, \ldots, \}$. Let $Cand$ denote these candidate choices.

(3) The center adopts choice $k^* = \arg\max_{k \in Cand} \sum_i v_i(k; \hat{\theta}_i)$ for E_N, and choice $k^{-i} = \arg\max_{k \in Cand} \sum_{j \neq i} v_j(k; \hat{\theta}_j)$ for each marginal economy.

Step (3), in which the maximal choice is taken from the set of candidates for each economy $\{E_N, E_{N-1}, \ldots\}$, can require the center to adopt a simple heuristic to modify a choice from one economy so that it is feasible in another. For instance, given an allocation of goods in an auction setting, the center can simply drop any allocation to agent i in k when considering the value of this solution for E_{N-i}.

Suppose the canonical distributed algorithm is used to define a distributed VCG mechanism, with VCG payments computed on the basis of the final choices (denoted $k^*, k^{-1}, k^{-2}, \ldots$). Fix reports $\hat{\theta}_{-i}$ by agents $\neq i$. Now, the utility, $u_i(g(s^m(\theta_i, \hat{\theta}_{-i})); \theta_i)$, to agent i from the intended strategy is:

$$v_i(k^*; \theta_i) + \sum_{j \neq i} v_j(k^*; \hat{\theta}_j) - \sum_{j \neq i} v_j(k^{-i}; \hat{\theta}_j)$$

In a centralized VCG the agent would choose to report its true type in equilibrium, because its report can only influence its utility indirectly through its effect on the choice selected by the mechanism. By the standard Groves argument, reporting a true type is optimal because the mechanism will then choose k^* to exactly maximize $v_i(k; \theta_i) + \sum_{j \neq i} v_j(k; \hat{\theta}_j)$.

In a distributed implementation, agent i can also: a) change the choice of k^* through its computational and message-passing and information-revelation actions within the distributed algorithm s^m; b) change the choice of k^{-i} through its actions within the distributed algorithm s^m. Indeed, strategy s^m is not in equilibrium. To see this, notice that agent i can now also influence the choice of k^{-i}. Agent i will always prefer to *understate* the total value of V_{N-i}, and thus prefer to obstruct any progress towards a good solution to this problem to the best of its ability. At best, the center will then adopt the same k^* as the choice without agent i, so that the agent's payment is zero because it appears that there is no better choice for the other agents even if agent i were not present.

5. An Easy Special Case: Groves Mechanisms

If our goal was simply to implement the social-welfare-maximizing outcome, and if running a budget-deficit was acceptable and the center can make a net payment to agents, then we can use the canonical approach for a faithful distributed implementation. We can use the Groves mechanism, in which payments are:

$$p_{\text{groves,i}}(\hat{\theta}) = -\sum_{j \neq i} v_j(k^*(\hat{\theta}); \hat{\theta}_j) \quad (2)$$

These are the payments from the center that align the incentives of each agent with that of maximizing total value. The VCG payments (Equation 1) are a specialization of Groves payments, introducing the additional payment term V_{N-i} from agent i to the center.

Theorem 1 *Distributed mechanism d_M for the Groves mechanism, in which a canonical distributed algorithm is used to determine the efficient choice in E_N, is an (ex post) faithful implementation of the efficient choice and Groves payments.*

Proof: The utility to agent i, given reports $\hat{\theta}_{-i}$ from other agents is $v_i(k^*; \theta_i) + \sum_{j \neq i} v_j(k^*; \hat{\theta}_j)$, where k^* solves $\max_k \sum_i v_i(k, \hat{\theta}_i)$. Agent i can influence the choice of k^* through both revelation and through its computational and message-passing actions. But, the Groves payments align agent i's incentives with the efficient choice k^* in E_N, and the agent will follow the intended strategy when this is also pursued by other agents. ■

Groves mechanisms can also be easily extended to provide a faithful distributed implementation of any *affine maximizer*, with the choice selected to solve $\max_k \sum_i w_i v_i(k; \theta_i) + b_k$ where $w_i, b_k \geq 0$ are set by the designer.

6. The Partition Principle: VCG Mechanisms

Now consider the problem of implementing the VCG outcome as a distributed mechanism. Unlike Groves, the VCG mechanism does not run at a deficit in many MAS problems (for example when used for a Combinatorial Auction [34]).

Theorem 2 (Partition Principle) *Distributed mechanism d_M for the VCG mechanism, in which a canonical distributed algorithm is adopted to solve $\{E_N, E_{N-1}, \ldots\}$, and in which computation is partitioned so that $s^m(\theta)$ will allow the center to solve E_{N-i} whatever the actions of agent i, is an (ex post) faithful distributed implementation of the efficient choice and VCG payments.*

Proof: The utility to agent i is $v_i(k^*; \theta_i) - (\sum_{j \neq i} v_j(k^{-i}; \hat{\theta}_j) - \sum_{j \neq i} v_j(k^*; \hat{\theta}_j))$. Agent i cannot influence the choice of k^{-i}, and once this is fixed the agent should follow s^m to maximize its total utility from the standard Groves argument. ■

Although we describe the partition principle in the context of the canonical distributed algorithm with each agent reporting its type as a first step, the result trivially extends to distributed mechanisms in which the center elicits dynamic value information, as long as it finally learns the value of the choices and shares necessary information with agents to perform the computation.

Note that it is important that no agent can tamper with the reports from other agents. (An agent is paid an amount equal to their *reported* value, so it would always want to *overstate* the value of other agents for the selected choice.) This is achieved in our model, because agents can report their type directly to the center. However, the partition principle still allows agents to send messages peer-to-peer during the *implementation* of a distributed algorithm. It is only the initial information-revelation that must be direct to the center along a trusted channel.

Example 1 [Distributed Systematic Search] Choose your favorite algorithm for distributed systematic search (such as Modi et al.'s DCOP algorithm [22]). First, form a search tree including all agents, and have the agents solve E_N. Then, form a search tree involving all agents except agent 1 and have them solve for E_{N-1}. Do the same for agent 2, and so on until all marginal economies are solved, with the center receiving a choice from the root of the tree in each case. Finally, implement the choice reported for E_N, and VCG payments on the basis of solutions to marginal economies.

Example 2 [Cooperative Problem Solving] Agents report types to the center, that broadcasts this information and also maintains a *blackboard* (see [9, 7]), on which it maintains the current best solution to $\{E_N, E_{N-1}, \ldots\}$. In the intended algorithm, agents follow a "best-effort" strategy, searching for, and suggesting, improvements to any problem. A best-effort strategy is defined as an algorithm that will eventually find an improvement when one exists. Here, we suppose the center *audits* new posts, and only accepts solutions onto the blackboard that improve the current solution. (This prevents agent i from scuppering progress towards solving E_{N-i}.) The mechanism terminates when every agent reports that E_N is solved correctly and every agent except agent i reports that E_{N-i} is solved correctly. Finally, the center implements the VCG outcome on the basis of the final solutions.

Many variations of this general blackboard-style approach are possible. For example, agents can be provided with shared scratch space to post (but not overwrite) *partial solutions* (similar to the hint-sharing methods proposed in the cooperative problem solving methods of Clearwater et al. [8]). A blackboard approach can also be used in an *in-*

cremental revelation mode, in which agents reveal new information about their own value in posting new solutions.

Another way to think about how to write a distributed algorithm for VCG that satisfies the partition principle is to consider algorithms with the following characteristics:

(1) agents only communicate with the center by suggesting candidate choices

(2) any candidate from agent i that for which agent i has no (reported) value is ignored by the center

(3) partitioning is static, in that the computation agent i is asked to perform does not depend on results from computation from any other agent.

We refer to such paradigms as *static-partitioning* because of property (3). Property (2) is critical because it ensures that an *optimal* statically-partitioned algorithm can never rely on agent i to provide computation that helps to solve E_{N-i}. With this, it is clear that these static-partitioning methods must satisfy the partitioning-principle, and provide faithful implementations of the VCG outcome.[5]

As an example, let E_{N+1} denote the efficient choice problem, $\max_{k \in \mathcal{K}} \sum_i v_i(k; \theta_i)$, subject to the additional constraint that $v_1(k; \theta_1) > 0$ (loosely, we say the solution must "contain" agent 1). Similarly, let E_{N+1-2} denote the problem $\max_{k \in \mathcal{K}} \sum_{j \neq 2} v_j(k; \theta_j)$ s.t. $v_1(k; \theta_1) > 0$.

Example 3 [static partitioning] Partition the computation across agents according to the following schedule: $\{E_{N+1}, E_{N+1-2}, E_{N+1-3}, \ldots\}$ to agent 1, $\{E_{N+2}, E_{N+2-1}, E_{N+2-3}, E_{N+2-4}, \ldots\}$ to agent 2, etc. Each agent can adopt any sound and complete algorithm to solve its assigned problems. Finally, the center compiles the solutions, e.g. $V_N = \max\{k^{+1}, k^{+2}, \ldots, k^{+m}\}$, where m indexes the Nth agent and k^{+i} denotes the reported solution to E_{N+i}.

7. Information-Revelation Principle

The *information-revelation* principle is a very general observation, in no way limited to distributed implementations of efficient outcomes. Rather, it applies to the distributed implementation of *any* strategyproof social choice function.

We need an additional property, called *information-revelation consistency*, which can be achieved either through checking, or through rules that constrain the feasible strategy-space.

Definition 6 *Information revelation actions a^1 and a^2, by agent i in states h^1 and h^2 are* ***consistent*** *when there is a single type $\hat{\theta}_i$ for which the intended strategy $s^m(h^1, \hat{\theta}_i) = a^1$ and $s^m(h^2, \hat{\theta}_i) = a^2$.*

As an example, consider an ascending-price auction in which "straightforward bidding" is the intended strategy, with an agent bidding for the item while the price is no greater than its value and it is not winning. Consistency requires that no agent can retract an earlier bid and that all bids must be at the current ask price (no jump bids). No agent would want to take either action if following a straightforward bidding strategy.

We say that a distributed mechanism *supports consistency checking* when every pair of information-revelation actions must be consistent (either through constraints, or through checking and then implementing a significantly bad outcome in case of a violation, such as excluding an agent from the system).

Theorem 3 (Information-Revelation Principle) *Distributed mechanism $d_M = (g, \Sigma, s^m)$ with consistency-checking is an (ex post) faithful implementation when the only actions are information-revelation actions and when $f(\theta) = g(s^m(\theta))$ is strategyproof.*

Proof: Since all actions are information-revelation actions, the space of possible outcomes is $g(s_i^m(\hat{\theta}_i), s_{-i}^m(\hat{\theta}_{-i}))$, but $g(s_i^m(\hat{\theta}_i), s_{-i}^m(\hat{\theta}_{-i})) = f(\hat{\theta}_i, \hat{\theta}_{-i})$, and $u_i(f(\theta_i, \hat{\theta}_{-i}); \theta_i) \geq u_i(f(\hat{\theta}_i, \hat{\theta}_{-i}); \theta_i)$ for all $\hat{\theta}_{-i}$, all θ_i, and all $\hat{\theta}_i \neq \theta_i$ by the strategyproofness of $f(\theta)$. ■

We can consider the application of this information-revelation principle to a distributed VCG mechanism.

Corollary 1 *Distributed mechanism $d_M = (g, \Sigma, s^m)$ is an (ex post) faithful implementation of the VCG outcome when all actions are information-revelation actions and the implementation $g(s^m(\theta))$ correctly computes the efficient choice and VCG payments for all types.*

The distributed mechanisms constructed around the information-revelation principle do not fall under the canonical distributed algorithms in §4 because the center *need not know* the exact value of the solutions to $\{E_N, E_{N-i}, \ldots\}$. For example, in a single-item Vickrey auction the center only needs to know that $v_1 \geq p$, $v_2 = p$ and $v_j \leq p$ for all $j \notin \{1, 2\}$ to implement the Vickrey payment p.

Example 4 [Ascending Auctions] The ascending-price combinatorial auctions (CA) described in Mishra & Parkes [21] are *ex post* Nash distributed implementations of the VCG mechanism. The ascending-price auctions (implicitly) maintain prices $p_i(S)$, on every bundle of goods S, and the intended straightforward bidding strategy has each agent responding with its demand set $D_i(p) = \{S : v_i(S) - p_i(S) \geq v_i(S') - p_i(S'), \ \forall S' \neq S\}$, for

5 We need a static partitioning to prevent results from agent i being used to help in the computation by another agent in solving E_{N-i}. Similarly, the center must only pick across candidates, with no additional combination operators.

all prices p. revealed-preference information from each agent is *consistent* across rounds.

Decentralized optimization algorithms, such as Dantzig-Wolfe, Bender's, and column generation [15, 3] have received much attention for solving large-scale structured optimization problems. A typical situation supposes that a firm needs to determine an allocation of resources across units, where individual units best understand their needs but the firm must impose global resource constraints. In the Dantzig-Wolfe decomposition, prices are published over a sequence of rounds, with units responding with preferred allocations. This information is aggregated in the center, which eventually announces an optimal global solution. These approaches are a very natural fit with the information-revelation principle:

Example 5 Adopt a decentralized optimization algorithm, such as Dantzig-Wolfe, and use it to compute the solution to $\{E_N, E_{N-1}, \ldots\}$. Ensure *consistency*, such that revealed preferences are consistent across rounds (this also ensures convergence). All responses from agents in Dantzig-Wolfe are information-revelation actions, and as such this provides an (ex post) faithful implementation of the VCG outcome.

8. Redundancy Principle

The *redundancy* principle is another very general observation, in no way limited to distributed implementations of efficient choices. Rather, it applies to the distributed implementation of *any strategyproof* social-choice function in which the computation can be usefully "chunked" into a sequence of steps, with each step given to two or more agents. Consider the following "chunk, duplicate and punish" algorithmic paradigm:

(1) Agents report types $\hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$

(2) Partition the distributed computation into "chunked" steps $s^{m1}, s^{m2}, \ldots, s^{mT}$.

(3) Give each chunked step to 2 or more agents, providing necessary inputs to allow the computation to be performed.

(4) The center steps in and repeats the calculation if the responses differ, punishing one (or both) agents when the response differs from that in the intended algorithm.

Punishment can be by removing the agent from the system for some period of time or some other punitive sanction, such as imposing a fine. Note that the center is assumed to have the computational resources to perform a check when agents respond with two different answers.[6]

6 This prevents an agent from "threatening" another agent, which would happen with a simpler scheme that punished both agents under any disagreement. We can also do this checking even when there is agreement, with some small probability, to handle the remaining issue of multiple equilibria. However, as we already argued, we thing software acts as a useful correlating device from this perspective.

Theorem 4 (Redundancy Principle) *Distributed mechanism $d_M = (g, \Sigma, s^m)$ constructed with a "chunk, duplicate and punish" scheme is an (ex post) faithful implementation when social-choice function $f(\theta) = g(s^m(\theta))$ is strategyproof.*

Proof: Consider agent i, and fix the strategy s^m_{-i} of other agents. First, whatever the information-revelation actions, agent i should choose a strategy that is faithful to the intended computational strategy because *any* deviation will lead to a penalty that by assumption exceeds any potential benefit. Then, we can assume w.o.l.g. that agent i will follows the intended computational strategy, and then appeal to the information-revelation principle and the strategyproofness of $f(\theta) = g(s^m(\theta))$, because the only remaining actions are information-revelation actions. ■

Example 6 [Pair-wise Chunking] Collect reported types $\hat{\theta}$, and then ask any two agents solve E_N, any two agents to solve E_{N-1}, and so on, for every agent. If the choices reported back for any problem differ, then the center can step in and determine the correct answer and punish.

Notice that this simple distributed implementation works *even if agent 1 is asked to solve V_{N-1}*.

Example 7 [Systematic Search] A more intricate example is to consider a distributed version of a systematic search algorithm, in which the center structures a search tree and allocates pairs of agents to conduct the search under nodes. For example state-of-the-art winner-determination algorithms for CAs use "branch-on-bids" coupled with LP-based heuristics to determine optimal allocations [11]. Such a search could be structured to ask agents 2 and 3 to "continue to follow algorithm $\mathcal{A}$ and search under a particular node for 20 steps and then report back the new search tree," and so on ...

9. Discussion

There are many outstanding issues and lots of interesting directions:

Costly Computation. On one hand, we assume that computation is costly (else why else would we want to distribute it across agents?) but on the other hand, we assume that computation is free (else why else would an agent happily perform a computation for the center when it is indifferent about the result of the computation?). This is a tricky place to be! Future work should strive to explicitly consider an agent's computational cost within implementation.

Restricted Communication Networks. The model in this paper assumes that an agent can send a message to the center without interference from another agent. What are the implications of restricted communication networks, for example multi-agent systems in which messages can only be sent peer-to-peer [23, 29]?

Self-Enforcing Outcomes. Can we find ways to relax the assumption that the center can *enforce* an outcome? This has been considered in an interdomain routing setting [29], in which an agent's neighbors know the outcome (and the prescribed actions) and are able to monitor the agent's actions and report deviations to the center.

Specific Instantiations. It will be interesting to build out specific instantiations of the stylized examples provided in this paper, in an effort to begin to understand the computational effectiveness of distributed implementations of incentive-compatible mechanisms.

10. Conclusions

In addressing the implicit centralization of mechanism design theory, we have described three general principles to guide the development of faithful distributed implementations, in which self-interested agents *choose* to perform the computation and help the center to determine an appropriate outcome.

We hope this work will start an interesting conversation between researchers familiar with methods in DAI for solving distributed problems with cooperative agents with researchers in DAI familiar with methods for handling agent self-interest through centralized techniques from mechanism design. The goal should be distributed implementations with good computational properties and good incentive properties.

Acknowledgments

Useful comments and suggestions were received during presentations of earlier versions of this work at EC'03, Stanford, MIT, and AAMAS'03. This work is supported in part by NSF grant IIS-0238147.

References

[1] S. Baliga. Implementation in economic environments with incomplete information: The Use of Multi-Stage Games. *Games and Economic Behavior*, 27:173–183, 1999.

[2] J. S. Banks, J. O. Ledyard, and D. Porter. Allocating uncertain and unresponsive resources: An experimental approach. *The Rand Journal of Economics*, 20:1–25, 1989.

[3] S. Bradley, A. Hax, and T. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[4] R. Brafman and M. Tennenholtz. Efficient learning equilibrium. *Artificial Intelligence*, 2004. To appear.

[5] P. J. Brewer. Decentralized computation procurement and computational robustness in a smart market. *Economic Theory*, 13:41–92, 1999.

[6] S. Brusco. Implementing action profiles with sequential mechanisms. *Review of Economic Design*, 3:271–300, 1998.

[7] N. Carver. A revisionist view of blackboard systems. In *Proc. 1997 Midwest Art. Intell. and Cog. Sci. Conf.*, 1997. May.

[8] S. H. Clearwater, B. A. Huberman, and T. Hogg. Cooperative solution of constraint satisfaction problems. *Science*, 254:1181–1183, 1991.

[9] D. D. Corkill, K. Q. Gallagher, and P. M. Johnson. Achieving flexibility, efficiency and generality in blackboard architectures. In *Proc. 6th Nat. Conference on Art. Intelligence (AAAI)*, pages 18–23, 1987.

[10] R. K. Dash, N. R. Jennings, and D. C. Parkes. Computational-mechanism design: A call to arms. *IEEE Intelligent Systems*, pages 40–47, November 2003. Special Issue on Agents and Markets.

[11] S. de Vries and R. V. Vohra. Combinatorial auctions: A survey. *Informs Journal on Computing*, 15(3):284–309, 2003.

[12] E. Ephrati and J. S. Rosenschein. The clarke tax as a consensus mechanism among automated agents. In *Proc. 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 173–178, July 1991.

[13] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing*, pages 173–182, 2002.

[14] J. Feigenbaum and S. Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, 2002.

[15] A. M. Geoffrion. Elements of large-scale mathematical programming. *Management Science*, 16(11):652–691, 1970.

[16] L. Hunsberger and B. J. Grosz. A combinatorial auction for collaborative planning. In *Proc. 4th International Conference on Multi-Agent Systems (ICMAS-00)*, pages 151–158, 2000.

[17] M. O. Jackson. A crash course in implementation theory. Technical Report SSWP 1076, California Institute of Technology, 1999.

[18] M. O. Jackson. Mechanism theory. In *The Encyclopedia of Life Support Systems*. EOLSS Publishers, 2000.

[19] V. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), 1999.

[20] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[21] D. Mishra and D. C. Parkes. Ascending price Vickrey auctions using primal-dual algorithms. Technical report, Harvard University, 2004.

[22] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Solving distributed conraint optimization problems optimally,efficiently, and asynchronously. *AIJ*, 2004. Forthcoming.

[23] D. Monderer and M. Tennenholtz. Distributed games: From mechanisms to protocols. In *Proc. 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 32–37, July 1999.

[24] J. Moore and R. Repullo. Subgame perfect implementation. *Econometrica*, 56(5):1191–1220, Sept. 1988.

[25] D. C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, May 2001.

[26] D. C. Parkes and S. Singh. An MDP-based approach to Online Mechanism Design. In *In Proc. 17th Annual Conf. on Neural Information Processing Systems (NIPS'03)*, 2003.

[27] J. S. Rosenschein and G. Zlotkin. Designing conventions for automated negotiation. *AI Magazine*, 1994. Fall.

[28] T. Sandholm. Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. *Autonomous Agents and Multi-Agent Systems*, 3(1):73–96, 2000.

[29] J. Shneidman and D. C. Parkes. Overcoming rational manipulation in mechanism implementations. Technical report, Harvard University, 2003. Submitted for publication.

[30] J. Shneidman and D. C. Parkes. Using redundancy to improve robustness of distributed mechanism implementations. In *Fourth ACM Conf. on Electronic Commerce (EC'03)*, 2003. (Poster paper).

[31] Y. Shoham and M. Tennenholtz. On rational computability and communication complexity. *Games and Economic Behavior*, 35:197–211, 2001.

[32] Y. Shoham and M. Tennenholtz. Non-cooperative computation: Boolean functions with correctness and exclusivity. *Theoretical Computer Science*, 2004. To appear.

[33] R. Smorodinsky and M. Tennenholtz. Overcoming free riding in multi-party computations – The anonymous case. Technical report, Technion, 2003.

[34] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.