MDPOP: Faithful Distributed Implementation of Efficient Social Choice Problems

Adrian Petcu *
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
adrian.petcu@epfl.ch

Boi Faltings
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
boi.faltings@epfl.ch

David C. Parkes †
DEAS, Harvard University
Cambridge, MA 02138 USA
parkes@eecs.harvard.edu

ABSTRACT

We model social choice problems in which self interested agents with private utility functions have to agree on values for a set of variables subject to side constraints. The goal is to implement the efficient solution, maximizing the total utility across all agents. Existing techniques for this problem fall into two groups. Distributed constraint optimization algorithms can find the solution without any central authority but are vulnerable to manipulation. Incentive compatible mechanisms can ensure that agents report truthful information about their utilities and prevent manipulation of the outcome but require centralized computation.

Following the agenda of *distributed implementation* [16], we integrate these methods and introduce *MDPOP*, the first distributed optimization protocol that *faithfully* implements the VCG mechanism for this problem of efficient social choice. No agent can benefit by unilaterally deviating from any aspect of the protocol, neither information-revelation, computation, nor communication. The only central authority required is a bank that can extract payments from agents. In addition, we exploit structure in the problem and develop a faithful method to redistribute some of the VCG payments back to agents. Agents need only communicate with other agents that have an interest in the same variable, and provided that the distributed optimization itself scales the entire method scales to problems of unbounded size.

1. INTRODUCTION

Distributed optimization problems can model environments where a set of agents must agree on a set of decisions subject to side constraints. We consider settings in which each agent has its own preferences on subsets of these decisions, expressed as relations that define its utility. The agents are self interested, and each one would like to obtain the decision that maximizes its own utility. However, the system as whole agrees (or some social designer determines) that a solution should be selected to maximize the total utility across all agents. Thus, this is a problem of *efficient social choice*. As motivation, we have in mind problems such as meeting scheduling, where the decisions are about when and where to hold each meeting, or scheduling contractors in construction projects.

Agents can of course solve such problems using a central authority that computes the optimal solution. In combination with a mechanism such as the Vickrey-Clarke-Groves (VCG) mechanism, we can also prevent manipulation by agents. However, in many practical settings it is hard to bound the problem so that such a central authority is feasible. Consider meeting scheduling: while each agent only participates in a few meetings, it is in general not possible to find a set of meetings that has no further constraints with any other meetings and thus can be optimized separately. Similarly, contractors in a construction project simultaneously work on other projects, again creating an unbounded web of dependencies that cannot be optimized in a centralized fashion.

Algorithms for distributed constraint reasoning such as ABT and AWC ([21]), AAS [20], DPOP [17] and ADOPT [14] can deal with problems of unbounded size as long as the influence of each agent on the solution is limited to a bounded number of variables. However, the current techniques do not address the problem of making agents truthfully declare their preferences and execute the protocol correctly.

In this paper, we advance the agenda of *distributed implementation* [16], which integrates methods from mechanism design with methods from distributed constraint optimization. In distributing the centralized computation of mechanism design across a system of self-interested agents the key challenge is to ensure that agents cannot gain from deviating from the distributed protocol. In addition to information revelation, agents will now be asked to participate in computation and message passing, both of which can provide new opportunities for manipulation. We describe the first faithful distributed constraint optimization algorithm, implementing the VCG outcome without any trusted third party besides a bank, used to enforce payments. The protocol forms an *ex post* Nash equilibrium [12], so that no agent can benefit by unilaterally deviating, whatever the utility functions of other agents and whatever the constraints. While noting that our protocol *never* runs at a deficit, we also demonstrate how to exploit problem structure in facilitating payment distribution *back* to agents from the bank. To do this we identify components of the problem that define payments that cannot be influenced by some subset of agents, that are then eligible to receive a share of the payments.

After preliminaries, in Section 3 we describe the DPOP [17] algorithm for distributed constraint optimization, which is the focus of our study. Section 4 extends DPOP to compute the VCG outcome and proves that the extended protocol, called MDPOP, is

*The first author is a PhD student at EPFL/Lausanne, supported by the Swiss National Science Foundation grant 200020-103421/1.

†Parkes is supported by National Science Foundation grants IIS-0238147, IIS-0534620 and an Alfred P. Sloan Foundation award.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

faithful. We also provide an accelerated version of MDPOP that simultaneously computes the solution to the marginal and main economies, and again establish faithfulness. Section 5 discusses the issue of budget balance and defines our payment redistribution method.

2. PRELIMINARIES

We assume that the social choice problem consists of a finite but possibly unbounded number of decisions that all have to be made at the same time. Each decision is modeled as a variable that can take values in a well-defined domain. There can be side constraints between the variables, and each agent can also have private *relations* that define its utility for decisions.

Modeled as a distributed constraint optimization problem, DCOP($\mathcal{A}$) on agents $\mathcal{A}$ we have:

DEFINITION 1. *An efficient social choice problem is modeled as a distributed constraint optimization problem (DCOP) as a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R} >$ *such that:*

$\mathcal{A} = \{A_1, ..., A_n\}$ *is a set of* **self-interested** *agents interested in the optimization problem;*

$\mathcal{X} = \{X_1, ..., X_m\}$ *is the set of* **public** *decision variables;* $P(A_i) \subseteq \mathcal{X}$ *is the sub-domain of variables on which agent* A_i ***could*** *have relations;* $X(A_i) \subseteq P(A_i)$ *are the variables in which agent* A_i *is interested and does have relations;*

$\mathcal{D} = \{d_1, ..., d_m\}$ *is the set of finite* **public** *domains of the variables* $\mathcal{X}$*; each domain is known to all interested agents;*

$\mathcal{C} = \{c_1, ..., c_q\}$ *is a set of* **public** *constraints, where a constraint* c_i *is a function* $c_i : d_{i_1} \times .. \times d_{i_k} \rightarrow \{-\infty, 0\}$ *that returns 0 for all allowed combinations of values of the involved variables, and* $-\infty$ *for disallowed ones; these constraints are known and agreed upon by all agents involved in the respective communities;*

$\mathcal{R} = \{R_1, ..., R_n\}$ *is a set of* **private** *relations, where* R_i *is the set of relations specified by agent* A_i *and relation* $r_i^j \in R_i$ *is a function* $d_{j_1} \times .. \times d_{j_k} \rightarrow \mathbb{R}$ *specified by agent* A_i*, which denotes the utility* A_i *receives for all possible values on the involved variables* $\{j_1, \dots, j_k\}$ *(negative values can be thought of as costs).*

The optimal solution is a complete instantiation X^* *of all variables in* $\mathcal{X}$*, s.t.* $X^* = argmax_{X \in \mathcal{D}}(\sum_{R_i \in \mathcal{R}} R_i(X) + \sum_{c_i \in \mathcal{C}} c_i(X))$[1]*, where* $R_i(X) = \sum_{r_i^j \in R_i} r_i^j(X)$ *is* A_i*'s utility for this solution.*

Refer to the agents A_i for which $X_j \in X(A_i)$ for some variable X_j as forming the *community* for variable X_j. We will use DCOP($-A_i$) to denote the constraint optimization problem without agent A_i, and refer to this as the "marginal problem without agent A_i."

An agent can also have *private variables*, and relations/constraints imposed on subsets of private variables and public variables. Decisions about private variables, as well as explicit information about these relations and constraints will remain private to an agent.

In addition to defining values for variables, our faithful protocols will also define payments, to be collected (or made) to agents. The only central authority that we require is a *bank* that can enforce these payments. Agents are modeled with quasilinear utility functions, so that agent i's total utility for decision X and payment p made to a bank is $R_i(X) - p$.

The main assumptions made for this paper are as follows:

- The set of variables $\mathcal{X}$, i.e. the number of decisions, is fixed and independent of the participating agents. Moreover, each agent knows the variables that it is interested in.
- Domains $\mathcal{D}$ are known to all interested agents.
- Each constraint $c_i \in \mathcal{C}$ is known to all agents interested in any variable involved in c_i.
- The agents with possible and actual interest in a variable X_i are known to all agents in the community of X_i.
- An agent can communicate with all agents in all communities in which it is a member.
- Agents are modeled as *rational but helpful*, meaning that although self-interested, they will follow a protocol whenever there is no deviation that will make them *strictly* better off.
- No collusion between agents.
- The problem has a feasible solution.
- Catastrophic failure if all agents in a community do not eventually agree on the same value for the variable.
- Every agent has a trusted communication channel with the bank.

To motivate the assumption that all members of a community are known to each other, consider meeting scheduling in which the decision variables are the times and locations of each meeting. Here, we would require that for each meeting there will be a list of participants that have to agree on the time and place. Realize that the only communication that we assume (other than with the bank) is among agents in the same community.

The assumption of catastrophic failure given disagreement is only used to ensure that once the multi-agent system has come to a decision it will be finally *executed.* It is to prevent "hold-out" by an unhappy agent at this final stage. Given that the other agents set their local values to be the agreed upon solution, no agent can benefit by adopting an alternative view of the decision. To motivate this, realize that a scheduled meeting where some participants assumed a different time than others would not be valid, benefitting no one. [2]

A simple "centralized" model of the DCOP($\mathcal{A}$), which we write COP($\mathcal{A}$), can be represented as a *multigraph* (for example Figure 1(a)), with the decision variables as nodes, and (possibly) multiple relations belonging to different agents that involve the same variables. Our complexity results are stated in terms of the induced width of this graph ([3]).

In order to allow multiple agents to express preferences on the same set of variables and *in a distributed fashion*, we adopt a distributed model where each agent has a local replica of the variables that it is interested in (e.g. Figure 1(b).) For each public variable, $X_j \in X(A_i)$, agent A_i has a local copy of X_j, denoted X_j^i. Agent A_i then models its interests as a local problem $COP(X(A_i), R_i)$, by specifying its relations $r_i^j \in R_i$ on the locally replicated variables $X(A_i)$. All copies of the same variable are synchronized between agents through equality constraints. In solving the problem, agents interact with others only through the equality constraints between local replicas of the public variables. [3]

---

[1]Notice that the second sum is either $-\infty$ if X is an infeasible assignment, or 0 if it is feasible. Thus, optimal solution X^* will always satisfy all hard constraints when that is possible.

[2]On the other hand, this is not an appropriate assumption in market domains where the decision is a trade of goods: it may in fact *not* be catastrophic for a seller to finally renege on an agreed trade. Here, we would need additional techniques such as monitoring to extend out methods. See Shneidman and Parkes [19] for an extended discussion of the problem of final execution of an agreement.

[3]Local, private variables do not show up in inter-agent communication. Agents typically need not solve the internal problem for all

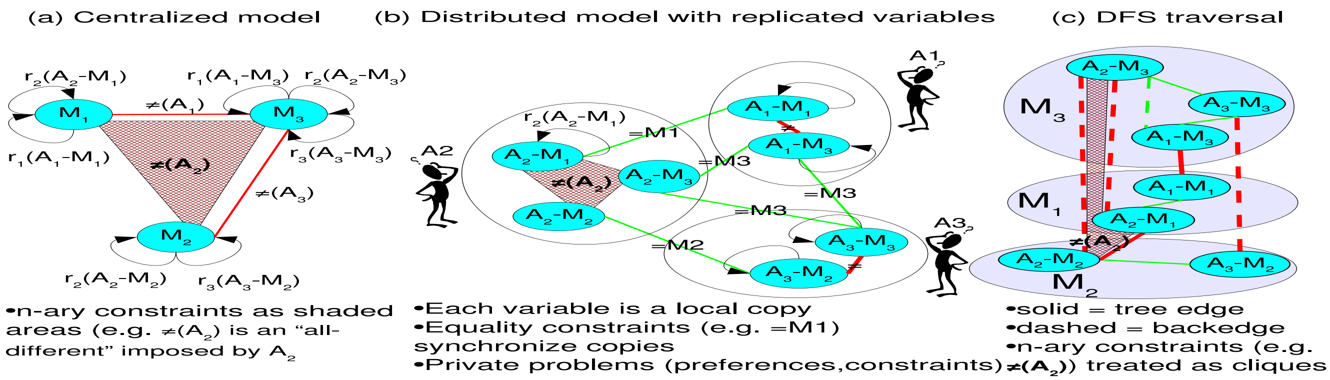

Figure 1: *A meeting scheduling problem, its modeling as a DCOP with replicated variables, and a DFS arrangement*

Example: Meeting Scheduling. This model can be instantiated for distributed meeting scheduling, yielding the PEAV model [13]. Figure 1 shows an example where 3 agents want to find the optimal schedule for 3 meetings. Each agent has as variables the starting times of the meetings it participates in (e.g. $A_2\_M_1$ represents the local copy of the variable representing meeting M_1 for agent A_2). Local *all-different* constraints between an agent's variables ensure that it does not participate in several meetings at the same time. Inter-agent equality constraints between local copies corresponding to the same meeting model the requirement of global agreement. Unary relations on the starting times of the meetings (e.g. $r_2(A_2\_M_1)$) model the preferences of the agents.

2.1 The Centralized VCG Mechanism

The Vickrey-Clarke-Groves (VCG) mechanism (see Jackson [11]) provides a centralized and incentive-compatible (IC) solution to efficient social-choice problems. Indeed, the Groves family of mechanisms (of which VCG is an instance) are the only efficient, IC social choice mechanisms [9]. There is a long tradition of leveraging the VCG mechanism within DAI, going back to Ephrati and Rosenschein [4] who considered the use of VCG mechanisms to achieve consensus.

To use the centralized VCG, each agent would report to a center its relations (and also the domains of variables, and constraints if they were not already known by the center). The center would assign values to variables and determine payments to be made by each agent. The VCG mechanism enjoys a strong form of IC: it is *truthful*, meaning that each agent can always maximize its own utility by reporting true information about its relations, *whatever* the reports of other agents. Truthful reporting is a *dominant-strategy equilibrium* (DSE), which is useful because it frees an agent from modeling the behavior of other agents in computing its equilibrium strategy. Each agent makes a payment equal to the marginal impact of its presence on the rest of the system. In determining this, the center in the VCG mechanism would also solve the marginal problems DCOP($-A_i$) without each agent A_i. Let X^*_{-i} denote the solution to DCOP($-A_i$) and X^* denote the solution to DCOP($\mathcal{A}$). The payment by agent i to the center is:

$$Tax(A_i) = \sum_{j \neq i} \left(R_j(X^*_{-i}) - R_j(X^*) \right), \tag{1}$$

where $R_j \in \mathcal{R}$ are the relations specified by agent A_j. Thus, agent A_i makes a payment equal to the total marginal negative effect of its presence on the utility of other agents.

combinations of values of the public variables [21].

Realize that in all problem instances we have $Tax(A_i) \geq 0$, for all A_i, with $\sum_{j \neq i} R_j(X^*_{-i}) \geq \sum_{j \neq i} R_j(X^*)$ because agent A_i's presence can only have the effect of changing the values of variables away from the best possible settings just for agents $\neq A_i$. Thus, we always have *weak* budget-balance, with the center running a surplus in all instances of our social choice problem.

The payment by agent A_i can be disaggregated, with $Tax_j(A_i) = R_j(X^*_{-i}) - R_j(X^*)$ denoting the payment made by agent A_i based on its marginal effect on agent A_j. Indeed, it can be further disaggregated to the individual relations of agent A_j. This simple observation will be very powerful in our setting.[4] It will permit a distributed computation of tax payments, where agent A_j computes the payment that should be made by other agents to the bank for their marginal effect on itself; i.e. agent A_j computes $Tax_j(A_i)$ for all $i \neq j$, which will be possible because A_j will know the values on variables of interest in X^* and X^*_{-i}.

2.2 Distributed Implementation

Parkes and Shneidman [16, 19] introduce the notion of *distributed implementation* for social choice problems. A distributed implementation (DI), $d_M = < g, \Sigma, s^m >$, defines an outcome rule $g : \Sigma^n \rightarrow \mathcal{D} \times \mathbb{R}^n$, where $g_1 \in \mathcal{D}$ defines values on variables and $g_2 \in \mathbb{R}^n$ defines the payment by each agent, a feasible strategy space Σ, and a *suggested* (multi-agent) protocol s^m. A protocol d_M is ex post *faithful* if suggested protocol s^m is an *ex post* Nash equilibrium (NE), meaning that no agent can benefit by deviating from the protocol in equilibrium (i.e. given that other agents follow the protocol) and whatever the particular instance of DCOP.

In a DI, the suggested protocol, s^m, combines the information revelation actions of mechanism design with the computational and communication actions of distributed algorithms. Thus, in following s^m an agent is both revealing information about its private relations and assisting in solving the DCOP and computing payments. The outcome rule g defines the assignment of values and payments for all possible termination states, including those that could arise from unilateral deviations. The feasible strategy space Σ, restricts the space of actions available to an agent in all possible states of the protocol, i.e. the messages that an agent can send that are interpretable by the other agents given that they are following the suggested protocol.

Parkes and Shneidman [16] introduced the **partition principle** for the distributed implementation of VCG mechanisms. Briefly, this principle states that a distributed mechanism is an *ex post*

[4] See Feigenbaum et al. [6] for a corresponding disaggregated VCG payment in the domain of shortest-path Internet routing.

faithful distributed implementation of efficient social choice if: (1) optimal solutions are always obtained for $DCOP(\mathcal{A})$ and $DCOP(-A_i)$ given s^m; (2) agent A_i cannot influence either the solution to $DCOP(-A_i)$ or its tax; (3) the optimal solution of $DCOP(\mathcal{A})$ is correctly executed and the corresponding taxes collected.

Loosely, the partition principle holds because no agent can affect its payment for any outcome. Thus, it is in the best interest of every agent to follow the suggested protocol so that the efficient outcome (i.e. the outcome selected in the VCG mechanism) is selected. The only effect of a deviation by an agent is to change either the outcome or some other agent's payment. Truthfulness of VCG then gives faithfulness. The suggested strategy forms an *ex post* NE but not a DSE because it relies on other agents following the strategy; if another agent deviates, e.g. from its role in the computation to solve $DCOP(\mathcal{A})$, then the correct outcome of the VCG mechanism will not be selected.

In related work, Feigenbaum and colleagues [6, 7] introduced the notion of *distributed algorithmic mechanism design*, but emphasized complexity questions rather than the faithfulness that is central to DI; see [19] for a faithful extension. Monderer and Tennenholtz [15] consider a distributed single item auction problem, but focus on communication of messages by self-interested agents rather than distributed computation. Finally, Izmalkov et al. [10] leverage cryptographic methods to convert centralized mechanisms into DIs on fully connected graphs.

3. DISTRIBUTED OPTIMIZATION VIA DPOP

In this section, we instantiate the DPOP algorithm [17] for efficient social choice. DPOP is an instance of Dechter's general bucket elimination scheme [3], adapted for the distributed case. This instantiation runs in three phases, which are very similar to the ones from the standard DPOP protocol. Phase one (section 3.1) constructs $DFS(\mathcal{A})$, which defines the control ordering of the inference algorithm. Phase two is a bottom-up utility propagation, and phase three is a top-down value assignment propagation (see section 3.2). There are some slight differences in phases one and two because we seek to exploit the structure of this DCOP model with replicated variables, for computational efficiency reasons.

Notice that DPOP can be applied to disconnected problems as well: the DFS arrangement is then a DFS forest, and agents in each connected component simply execute DPOP on a tree of that forest. The solution to the (disconnected) problem is then simply the union of optimal solutions for each independent, connected subproblem.

Section 4 will modify DPOP to make it faithful.

3.1 Phase One: DFS Tree Generation

See Algorithm 1. This phase has as a goal to generate a depth-first traversal (DFS) of the problem graph in a distributed manner. A DFS arrangement of a graph G is a rooted tree with the same nodes and edges as G and the property that adjacent nodes from the original graph fall in the same branch of the tree (thus, there are no edges between different branches of the tree). Common definitions of parent, child, pseudoparent, pseudoparent apply. For example, in Figure 1(c), $A_2\_M_1$ and $A_2\_M_2$ are parent/child to each other, and $A_2\_M_3$ and $A_2\_M_2$ are pseudoparent/pseudochild. *Tree-edges* connect parents/children (e.g. $A_2\_M_1 - A_2\_M_2$), and *back-edges* connect pseudoparents/pseudochildren (e.g. $A_2\_M_3 - A_2\_M_2$).

First, each agent A_i formulates internally its interests on the variables $X(A_i)$ as $COP(X(A_i), R_i)$, with a replicated variable X_j^i for each $X_j \in X(A_i)$. All agents subscribe to the communities

Algorithm 1: *Phase One of DPOP.*

DPOP($\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}$):

Each agent A_i models its interests as $COP_i(X(A_i), R_i)$: a set of relations R_i imposed on a set $X(A_i)$ of variables X_j^i that each replicate each public variable $X_j \in X(A_i)$

Each agent A_i subscribes to the communities of $X_i \in X(A_i)$

DFS Generation:

The agents $\mathcal{A}$ choose one of the variables, X_0, as the root.

Agents in X_0's community elect a "leader", A_r

A_r initiates the token passing to construct the DFS

At completion, each A_i knows $P(X_j^i)$, $PP(X_j^i)$, $C(X_j^i)$, $PC(X_j^i)$, for all local copies X_j^i.

they are interested in, and learn which other agents belong to these communities. [5] In doing this the *problem graph* is constructed. Next, one of the variables, X_0 is chosen as the DFS root. [6] The agents involved in the community for X_0 then randomly choose one of them, A_r as the *leader*. The local copy X_r^0 of variable X_0 forms the root of the DFS. In the case that the problem is initially disconnected then a modification is required to choose multiple root communities, one for each connected component.

Second, the agents participate in a distributed depth-first traversal of the problem graph to construct the DFS for problem DCOP($\mathcal{A}$), which we denote $DFS(\mathcal{A})$. (Multiple DFS trees are generated for disconnected problems.) For convenience, we describe the DFS process as a token-passing algorithm in which all members within a community can observe the release or pick up of the token by the other agents. But, this can also be implemented via (private) message passing.

Let us refer to the example from Figure 1, and assume that M_3 was chosen as the start community, and A_2 was chosen within the community as the start node. A_2 creates an empty token, adds $A_2\_M_3$'s ID to the token, and then releases it back to the community. Another agent from M_3's community (e.g. A_3) picks up the token, adds its copy of M_3 to the token ($A_3\_M_3$'s ID) and releases it again. A_1 picks it up and automatically adds equality constraints between its variable $A_1\_M_3$ and all its corresponding replica variables that precede it in the context of the token ($A_2\_M_3$ and $A_3\_M_3$) (i.e. one tree edge and one back edge.) Notice that the result is that all replicas of a variable are arranged in a chain, and have equality constraints (back-edges) with all the predecessors that are replicas of the same variable.

Agent A_1 also adds its copy of M_3 to the token ($A_1\_M_3$) and as the last agent in community M_3 to receive the token looks to see if it is a member of another community that has yet to receive the token (choosing one at random if such a community exists). Here, agent A_1 is linked to community M_1 and adds its copy of M_1 to the token (i.e. $A_1\_M_1$), and then releases the token in M_1's community, where A_2 picks it up. When a dead end is reached, the last agent backtracks by sending the token back to its parent. In our example, this happens when A_3 receives the token from A_2 in the M_2 community. Then, A_3 sends back the token to A_2, etc. Eventually the token returns on the same path all the way to the root, and then the process is complete. [7]

[5] A community can be e.g. a bulletin board, a mailing list, etc

[6] This can be done e.g. randomly, using any distributed algorithm for random number generation, or by simply picking the variable with the highest ID, etc.

[7] K-ary constraints (involving k variables) are treated like a cliques during the DFS construction. Concretely, in Figure 1, there is a ternary *all-diff* constraint $\neq A_2(M_1, M_2, M_3)$. A_2 then considers

In constructing $DFS(\mathcal{A})$, the DFS traversal is made according to the structure defined by the relations of the agents. Most hard constraints appear thus as backedges in such a DFS tree. By convention, any hard constraint $c_i \in \mathcal{C}$ is assigned to the highest agent in the community of the variable involved in c_i that is lowest in the DFS ordering. For example, in Figure 1(c), assume that there is a constraint between M_2 and M_3 that specifies that M_2 should occur after M_3. With our convention, this constraint becomes a backedge between the 2 communities, and is assigned to A_2 for handling, because $A_2\_M_2$ is the highest variable in M_2's community, which is lower than M_3's community in the DFS. A_2 then handles this constraint in parallel with its own relation $A_2\_M_2$-$A_2\_M_3$.

Realize that the choice of DFS does not change the solution, so the choice of root node, leaders, etc does not affect the incentive properties.

3.2 Phases Two and Three: Inference

Phase 2 is a bottom-to-top pass that propagates aggregated information about the relations towards the root. The *UTIL* propagation starts bottom-up from the leaves and propagates upwards only through tree edges, from children to parents. A *UTIL* message sent by X_i to its parent X_j informs X_j how much utility $u^*_{X_i}(v^k_j)$ each one of its values v^k_j gives to the whole subtree rooted at X_i in the optimal solution.

To compute the *UTIL* message for its parent, X_j has to join the messages it received from all its children, and the relations it has with its parent and pseudoparents. [8] Afterwards, it projects itself out of the join and sends the result to its parent. The result of the projection is in fact the set of optimal utilities that can be obtained by the subtree rooted at this node, plus the relations it has with its parent/pseudoparents, for each combination of values of the parent/pseudoparents (see [17] for details and examples). This projection provides for an efficient algorithm.

A useful optimization for social choice problems can be introduced to handle replica variables. In the example of Figure 1, $A_2\_M_2$, $A_3\_M_2$ and $A_2\_M_1$ all have back-edges: $A_2\_M_2 - A_2\_M_3$, $A_3\_M_2 - A_3\_M_3$ and $A_2\_M_1 - A_2\_M_3$ respectively. These represent the inequality constraints for agents. Normal DPOP would condition $UTIL$ messages on both $A_2\_M_3$ and $A_3\_M_3$ separately. For social choice these will adopt the same values due to the equality constraints, and thus the conditioning can be collapsed into a single dimension, the value of M_3. This is possible because all 3 agents involved, i.e. A_1, A_2 and A_3 know that $A_1\_M_3$, $A_2\_M_3$ and $A_3\_M_3$ represent the same variable.

Phase 3 is a top-to-bottom pass that makes decisions about the value of variables, with decisions made recursively from the root down to the leaves. This "$VALUE$ propagation" phase is initiated by the agent A_r representing the root variable X_0, once it has received *UTIL* messages from all of its children. Based on these *UTIL* messages, the root assigns itself the value v^* that maximizes the sum of its own utility and that communicated by all its subtrees. It then sends a *VALUE*($X^0_r \leftarrow v^*$) message to every child. The process continues recursively to the leaves, with agents X_i assigning values to local copies of variables.

3.3 Complexity Analysis of DPOP

It has been proved in Petcu and Faltings [17] that *DPOP* produces a linear number of messages for general distributed optimization problems. Its complexity lies in the size of the *UTIL* messages (the *VALUE* messages have linear size). This is also true for its instantiation to social choice problems.

Let us denote by w the width of the problem graph for the centralized model of DCOP($\mathcal{A}$) (e.g. Figure 1(a)). The induced width of a graph is a topological parameter that captures the density and clustering of the graph [3]. It is roughly defined as the *maximal number of overlapping tree paths between any pair of different vertices*. In the example from Figure 1, $w = 2$. Let $D = \max_m |d_m|$ denote the maximal domain of any variable.

THEOREM 1. *The number of messages passed is* $2 \times m$, $(n-1)$ *and* $(n-1)$ *for phases one, two and three respectively, where* n *and* m *are the number of nodes and edges in the distributed model.*

The maximal amount of computation on any node in DPOP is $O(D^{w+1})$*, and the largest UTIL message has* $O(D^{w+1})$ *entries, where* w *is the width of the centralized problem graph.*

Sketch of Proof. Follows from the analysis of DPOP in Petcu and Faltings [17], and the fact that equivalent variables use up only one dimension in the *UTIL* messages (see Section 3.2), and that a dimension is not projected out immediately as it reaches the first target variable, but only when it reaches its top most copy. □

The complexity of DPOP for social choice problems is exponential in the tree width of the centralized graphical model, but not the decentralized graphical model which includes the replicated variables. This is due to the special handling of replica variables described in Section 3.2.

4. MDPOP: A FAITHFUL PROTOCOL FOR DISTRIBUTED OPTIMIZATION

In this section we extend the DPOP algorithm to define *MDPOP*, and prove that MDPOP is a faithful implementation of distributed constraint optimization, terminating with the outcome of the VCG mechanism. We first provide a simple extension, that we call *simple*-MDPOP, before describing our preferred extension, that we call MDPOP.

Algorithm 2: *Simple-MDPOP.*

Run DPOP for $DCOP(\mathcal{A})$ on $DFS(\mathcal{A})$; find X^*
forall $A_i \in \mathcal{A}$ **do**
    Run DPOP for $DCOP(-A_i)$ on $DFS(-A_i)$; find X^*_{-i}
    $\forall A_j \neq A_i$, compute $Tax_j(A_i) = R_j(X^*_{-i}) - R_j(X^*)$
    $\forall A_j \neq A_i$, report $Tax_j(A_i)$ to the bank
    Bank deducts $\sum_{j \neq i} Tax_j(A_i)$ from A_i's account
    A_i implements X^* as solution to its local $COP(A_i)$

Algorithm 2 describes simple-MDPOP. The algorithm is presented for a setting in which the main problem and the subproblems are connected but extends immediately to disconnected problems, as discussed in the previous section and without new incentive considerations. Notice that the protocol sets up, and then solves, $n + 1$ DPOP protocols, one for the main problem and one for the n marginal problems, with each agent A_i removed in turn. Once these $n + 1$ stages are complete every agent A_j has sufficient local knowledge of the solutions $\{X^*, X^*_{-1}, \ldots, X^*_{-n}\}$ to compute the tax payment that every other agent A_i, for $i \neq j$, should make to the bank because of its marginal effect on agent j. Each agent will finally respect decision X^*, to avoid catastrophic failure.

THEOREM 2. *The simple-MDPOP algorithm is a faithful distributed implementation of the optimal solution to a DCOP, and terminates with the outcome of the VCG mechanism.*

the variables in the scope of this constraint to be a fully connected component, which produces the result from Figure 1(c).

[8] A k-ary relation is introduced in this join only once, by the lowest node in the DFS tree, which is part of its scope. E.g. in Figure 1(c), the constraint $\neq A_2(M_1, M_2, M_3)$ is introduced by $A_2\_M_2$.

PROOF. Follows from the partition principle [16]. First, DPOP computes optimal solutions to DCOP($\mathcal{A}$) and $DCOP(-A_i)$ for all $A_i \in \mathcal{A}$ when every agent follows the protocol. Second, agent A_i cannot influence the solution to $DCOP(-A_i)$ because it is not involved in that computation. The DFS is constructed and then inference performed by the other agents, who completely ignore A_i's variables and constraints, and any messages that agent A_i might send. Moreover, agent A_i is not required to perform any message passing in solving for $DCOP(-A_i)$. Note that any hard constraints that A_i may have handled in $DCOP(\mathcal{A})$ are reassigned automatically to some other agent in $DCOP(-A_i)$.

Notice that $DCOP(-A_i)$ could become disconnected without the presence of agent A_i. However, as noted in the beginning of Section 3, DPOP would still solve $DCOP(-A_i)$ correctly. Finally, agent A_i cannot prevent the correct calculation and reporting of the tax it should pay because this is done by agents $A_j \neq A_i$. The bank collects payments and all agents finally set local copies of variables as in X^* to prevent catastrophic failure. (Notice that agent A_i will not deviate as long as other agents do not deviate. Moreover, if agent A_i is the only agent that is interested in a variable then its value is already optimal for agent A_i anyway.) □

In particular, notice that we get from the partition principle that no agent has an interest in obstructing the choice of root community or leader agent in Phase one of DPOP, or in the information-revelation, computation and message passing in Phases two and three of DPOP. Also, no agent A_i can usefully influence its payment by misreporting the local utility of another agent A_j, as $UTIL$ messages are exchanged. While this could change the select of X^* or X^*_{-k} for some $k \neq \{i, j\}$, it would *not* change the utility information used in finally determining agent A_i's payments because only the utility information local to A_j and known to A_j is used in computing the component of A_i's payment due to its effect on A_j.

Note on antisocial behavior: While it is true that an agent A_j has no immediate self-interest in reporting the payment another agent should make, it does have a long-term self-interest if it wants other agents to be truthful (e.g. imagine a system where over time agents realize that the correct payments are not being collected from others). Reporting exaggerated taxes hurts other agents, but does not increase one's own utility, so this is excluded by our assumption that the agents are self-interested but helpful.

4.1 Full-MDPOP

In *simple-MDPOP*, the computation to solve the main problem is completely isolated from the computation to solve each of the marginal problems. The full *MDPOP* algorithm leverages the computation already performed in solving the main problem in solving the marginal problems whenever this is possible and without breaking incentive properties.

This enables the algorithm to scale well to problems where each agent's influence is limited to a small part of the entire problem. [9]

The first stage of MDPOP solves the main problem just as in Simple-MDPOP, running DPOP($\mathcal{A}$). Once this is complete, each marginal problem is solved in parallel. To solve $DCOP(-A_i)$, a DFS-tree is constructed as a modification to DFS($\mathcal{A}$), retaining as much of the structure as possible. This maximizes the reuse of $UTIL$ messages. The new tree, $DFS(-A_i)$ must be constructed in a way that is non-manipulable, i.e. without allowing agent i to interfere with its construction, and also to ensure correctness. This requires that communities of variables that remain connected in $DCOP(-A_i)$ remain connected in the $DFS(-A_i)$ tree when edges that link to nodes owned by A_i are disabled in solving $DCOP(-A_i)$. For instance, in Figure 2, $A_1\_M_1$–$A_1\_M_3$ is a tree edge in $DFS(\mathcal{A})$, and its removal disconnects $DCOP(-A_1)$).

[9]For example, in a meeting scheduling problem with thousands of agents, any one agent only participates in a few meetings, in a rather restricted circle of acquaintances.

Algorithm 3: *MDPOP.*

Run DPOP for $DCOP(\mathcal{A})$ on $DFS(\mathcal{A})$; find X^*
forall $A_i \in \mathcal{A}$ **do**
  Create $DFS(-A_i)$ **by adjusting** $DFS(\mathcal{A})$**:**
  exclude all variables X^i_j and relations that belong to A_i;
  the highest descendant of each excluded X^i_j that has a back edge with an ancestor of X^i_j turns it into a tree-edge;
  Run DPOP for $DCOP(-A_i)$ **on** $DFS(-A_i)$:
  children/parents of each excluded X^i_j recompute their *UTIL* messages and restart propagations;
  reuse *UTIL* msgs from $DPOP(\mathcal{A})$ not influenced by A_i;
  Compute and levy taxes as in simple-MDPOP;
  A_i implements X^* as solution to its local $COP(A_i)$;

Phase One of MDPOP for a marginal problem. Consider DPOP($-A_i$). In building $DFS(-A_i)$ from $DFS(\mathcal{A})$, existing links that were back-edges in $DFS(\mathcal{A})$ can be turned into tree-edges in $DFS(-A_i)$ as necessary to keep it connected. This preserves as much as possible of the tree structure. Figure 2 shows an example of a common $DFS(\mathcal{A})$, adjusted for each marginal problem using this idea. For example, $A_2\_M_3 - A_2\_M_1$ is a back-edge in $DFS(\mathcal{A})$, but becomes a tree-edge in $DFS(-A_1)$ to compensate for the loss of edge $A_1\_M_1 - A_1\_M_3$. The algorithm works by considering each of the nodes X_i belonging to A_i in turn. For each X_i that will be excluded from DPOP($-A_i$), all nodes below X_i check the path from the root to themselves, and the list of nodes reachable from their children (both these pieces of information are available after $DFS(\mathcal{A})$ is constructed). The highest node below X_i that has a back edge pointing to a node above X_i converts this edge into a tree edge and converts its pseudo parent into a parent.

Thus, no additional links are created, as we use only existing ones, previously designated as back-edges. Realize that this conversion, in converting back edges to tree edges, cannot increase the induced width of $DFS(-A_i)$ above the one of $DFS(\mathcal{A})$, therefore *UTIL* messages can only decrease in size.

Phase Two of MDPOP for a marginal problem. Each marginal problem is then solved on $DFS(-A_i)$. Notice that the parent and children of excluded nodes will have to recompute their messages from $DPOP(\mathcal{A})$ to account for the new structure and initiate the corresponding propagation for $DPOP(-A_i)$.

Subsequently, any message can be reused iff it comes from a subtree that does not contain any of A_i's variables, because A_i could not have influenced it. E.g. in $DPOP(-A_1)$, $A_2 - M_1$ is a child of $A_2 - M_1 \in A_1$. It has to recompute a *UTIL* message and send it to $A_2 - M_3$. To do this, it can reuse the message sent by $A_2 - M_2$ in $DPOP(\mathcal{A})$, because the sending subtree does not contain A_1. By doing so, it reuses the effort spent in $DPOP(\mathcal{A})$ to compute the messages $A_3 - M_2 \rightarrow A_2 - M_2$ and $A_2 - M_2 \rightarrow A_2 - M_1$.

THEOREM 3. *The MDPOP algorithm is a faithful distributed implementation of the optimal solution to a DCOP, and terminates with the outcome of the VCG mechanism.*

Sketch of Proof. From the partition principle [16]. First, agent i cannot prevent the construction of a valid DFS for $DCOP(-A_i)$ because in the construction of $DFS(-A_i)$ from the main DFS, all transformations are initiated by neighbors of A_i, and all links with A_i are simply dropped. Second, agent i cannot influence the execution of DPOP on $DCOP(-A_i)$ because all messages that A_i

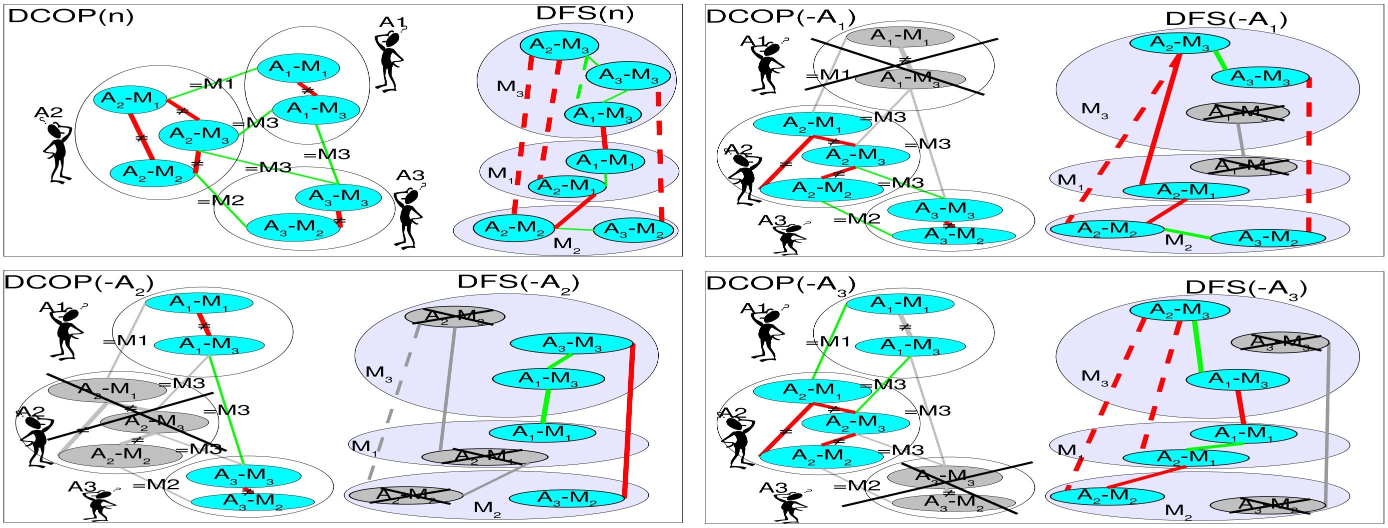

Figure 2: *Each agent A_i is excluded in turn from the optimization $DCOP(-A_i)$. $DFS(-A_i)$ is adapted from $DFS(\mathcal{A})$.*

influenced in the main problem $DCOP(\mathcal{A})$ are recomputed in the new structure. This follows from the fact that every link where agent A_i was responsible for computing a message is eliminated by its neighbors and that all these propagations are restarted. □

5. INCENTIVE COMPATIBLE VCG PAYMENT REDISTRIBUTION

No social choice mechanism can be efficient, incentive-compatible, and individually rational while at the same time guaranteeing exact budget-balance [8]. [10] In our setting, where there are no positive externalities, the VCG mechanism runs at a surplus with the bank receiving a net payment from agents. While these taxes cannot be simply redistributed among the agents, a tax payment *can* be refunded to an agent A_l as long as that agent has no influence on the computation of the payments it receives. This general idea was suggested by Ephrati and Rosenschein ([4]), and recently explored by Faltings [5] and Cavallo [2].

The most straightforward way to implement this idea is to consider any agent that does not influence any of the optimizations that are used in computing a certain part of the VCG tax as eligible to receive this tax. However, this approach would not maintain incentive-compatibility, as an agent with only a small influence on some aspect of the problem could gain an advantage by misstating its preferences to become non-pivotal and thus receive a possibly much larger tax payment.

Faltings [5] suggests to deal with this problem by forcing an agent A_l to be non-pivotal independently of its declarations by simply ignoring it in the optimization. In this way, it is guaranteed that the agent does not have an influence on the tax computation and thus can receive it without creating unwanted incentives. While the mechanism may be forced to choose a suboptimal solution, [5] shows through experiments on randomly generated problems that the expected utility loss from suboptimality is much smaller than what would result from wasting the taxes.

However, a drawback of this approach is that in a large optimization problem, some agent would not be considered at all in the entire problem. It would be more advantageous if various agents could receive *some portion* of the tax in return for *some* reduction of their influence on the solution.

Consider the VCG payment portion:

$$tax_j(A_i) = r_j(X^*_{-i}) - r_j(X^*)$$

that is paid in Algorithm 2 by agent A_i with respect to relation r_j. Let $r_j \in R_k$, i.e. r_j was posted by agent A_k.

We designate an agent A_l, $l \notin \{k, i\}$ to receive this payment as a refund. A straightforward way to choose A_l would be to take an agent that did not influence the values of any of the variables in r_j in the solution. However, this would destroy incentive-compatibility, since an agent may now have an incentive to hide its interest in order to be eligible to receive the refund. Similarly, an agent could have an interest to make other agents look pivotal to increase its own chance of receiving a refund.

To avoid such influence, A_l needs to be chosen independently of the agents' declarations. Our algorithm does this as follows:

1. For each agent A_l, we use the set $P(A_l)$ of the variables on which the agent could possibility express interest and ignore its declarations when they involve other variables.

2. For each payment portion $tax_j(A_i)$, choose an agent A_l that will be eligible to receive it, using any criterion that is not related to the agents' own declarations. This can be done by random selection among agents that cannot possibly be part of the community of the variables.

3. Using the declarations of the agents, for each payment portion verify that the agent A_l chosen to receive it indeed cannot have any influence on the values of the variables involved. If there is no possible influence, the agent receives the payment as a refund, if not, it has to be wasted.

We now give a brief description of the third step of the algorithm. This step is important because even an agent not in the community of a variable may still be able to influence relations via the propagation of its effect over the problem graph.

We use the *omnidirectional* utility propagation from the DPOP extension presented in [18]. In this version, messages circulate in

[10] For budget balance in general problems, one can settle for *ex ante* individual-rationality [1] and Bayes-Nash incentive-compatibility, but this requires the mechanism and agents to have common knowledge about a distribution on agent types.

all directions along the DFS tree (parent to children, too). A message from a parent to its child summarizes the utility information from the entire problem except the subtree of that child. Joining messages from the parent and the children gives each node the same global view of the system as the root in the simple DPOP has.

We can characterize the influence that an agent A_l has on a variable X_k by a *label* that contains a value for each member of X_k's domain d_k. The value is "1" if A_l can make X_k take the corresponding value by its declarations, and "0" if it cannot. If A_l posts a relation on X_k, it can make any value the most preferred one, so each position has a "1".

As an example, consider a variable X_k that can take three values `a, b, c`. Let A_l have no influence on any sibling or ancestor of X_k in the DFS ordering, but complete control of the value of X_k. Thus, A_l's label for X_k is `(1,1,1)`. Let X_a be the ancestor of X_k in the DFS ordering with possible values `d, e, f`, and assume that some other agent has imposed the following relation between X_k and X_a:

$X_a =$	d	e	f
$X_k = a$	3	2	1
$X_k = b$	2	3	1
$X_k = c$	4	3	2

Furthermore, let the sum of all other messages arriving at X_a, assuming omnidirectional propagation, be the vector $(5, 5, 5)$, giving the utilities of $X_a = (\mathtt{d}, \mathtt{e}, \mathtt{f})$ in the rest of the problem. Note that this vector is not influenced by A_l.

Agent A_l can influence the value of X_a only through the utilities it gives to the three different values of X_k. Letting these be $U_l(X_k)$, and factoring the utilities reported in the rest of the problem, the propagation would choose the maximum in each row of the following table, indicated in bold:

X_a=	d	e	f
$X_k = a$	$\mathbf{8 + U_l(X_k = a)}$	$7 + U_l(X_k = a)$	$6 + U_l(X_k = a)$
$X_k = b$	$7 + U_l(X_k = b)$	$\mathbf{8 + U_l(X_k = b)}$	$6 + U_l(X_k = b)$
$X_k = c$	$\mathbf{9 + U_l(X_k = c)}$	$8 + U_l(X_k = c)$	$7 + U_l(X_k = c)$

where the chosen row depends on the value of X_k. Now note that A_l can never force $X_a = \mathtt{f}$, since this will never give the maximum utility. Thus, A_l's label for X_a is `(1,1,0)`. Had A_l's label for X_k been `(1,0,1)`, its label for X_a would have been `(1,0,0)`, meaning that only $X_a = \mathtt{d}$ is possible and thus A_l has in fact no possibility to influence X_a's value.

Note that the number of "1"s in a label can never increase during such propagation, since for every choice of input value there can be only one optimal output value. This means that propagation will eventually converge to labels with a single "1". By propagating labels in the same way as propagating messages in MDPOP, we can determine the variables that an agent can potentially influence.

The presence of backedges in the DFS tree somewhat complicates the algorithm. The full algorithm will be described in a longer version of this paper.

6. CONCLUSIONS

We presented a multiagent constraint optimization algorithm for use in efficient social choice problems when agents are self interested and have private information about their utility for different outcomes. Our algorithm is faithful, in the sense that no agent can improve its utility either by misreporting its local information or deviating from any aspect of the algorithm. The only centralized control we assume is that of a bank that is able to receive messages about payments and collect payments. In addition to promoting efficient decisions we also seek to return payments back to agents, to further improve the net utility of outcomes. Future work should provide a comprehensive empirical analysis, in order to understand the scheme's scalability and budget balance properties on realistic problem instances.

7. REFERENCES

[1] C. d. Aspremont and L. A. Gerard-Varet. Incentives and incomplete information. *Journal of Public Economics*, 11:25–45, 1979.

[2] R. Cavallo. Optimal decision-making with minimal waste: Strategyproof redistribution of VCG payments. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-06)*, Hakodate, Japan, May 2006.

[3] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[4] E. Ephrati and J. Rosenschein. The Clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-91*, pages 173–178, Anaheim, CA, July 1991.

[5] B. Faltings. A budget-balanced, incentive-compatible scheme for social choice. In *Workshop on Agent-mediated E-commerce (AMEC) VI*. Springer Lecture Notes in Computer Science, 2004.

[6] 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.

[7] 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.

[8] J. Green and J.-J. Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45:427–438, 1977.

[9] T. Groves and M. Loeb. Incentives and public inputs. *Journal of Public Economics*, 4:211–226, 1975.

[10] S. Izmalkov, S. Micali, and M. Lepinski. Rational secure computation and ideal mechanism design. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 585–595, Washington, DC, USA, 2005. IEEE Computer Society.

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

[12] V. Krishna. *Auction Theory*. Academic Press, 2002.

[13] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the realworld: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS-04*, 2004.

[14] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AI Journal*, 161:149–180, 2005.

[15] 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.

[16] D. C. Parkes and J. Shneidman. Distributed implementations of Vickrey-Clarke-Groves mechanisms. In *Proc. 3rd Int. Joint Conf. on Autonomous Agents and Multi Agent Systems*, pages 261–268, 2004.

[17] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, Edinburgh, Scotland, Aug 2005.

[18] A. Petcu and B. Faltings. Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, Pittsburgh, USA, July 2005.

[19] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC'04)*, St. John's, Canada, 2004.

[20] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *AAAI/IAAI*, pages 917–922, Austin, Texas, 2000.

[21] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.