Journal of Artificial Intelligence Research 32 (2008) Submitted 11/07; published 07/08

M-DPOP: Faithful Distributed Implementation of Efficient Social Choice Problems

Adrian Petcu ADRIAN.PETCU@EPFL.CH
Boi Faltings BOI.FALTINGS@EPFL.CH
Artificial Intelligence Lab, Ecole Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland

David C. Parkes PARKES@EECS.HARVARD.EDU
School of Engineering and Applied Sciences, Harvard University
33 Oxford Street, Cambridge, MA 02138 USA

Abstract

In the efficient social choice problem, the goal is to assign values, subject to side constraints, to a set of variables to maximize the total utility across a population of agents, where each agent has private information about its utility function. In this paper we model the social choice problem as a distributed constraint optimization problem (DCOP), in which each agent can communicate with other agents that share an interest in one or more variables. Whereas existing DCOP algorithms can be easily manipulated by an agent, either by misreporting private information or deviating from the algorithm, we introduce *M-DPOP*, the first DCOP algorithm that provides a *faithful distributed implementation* for efficient social choice. This provides a concrete example of how the methods of mechanism design can be unified with those of distributed optimization. Faithfulness ensures that no agent can benefit by unilaterally deviating from any aspect of the protocol, neither information-revelation, computation, nor communication, and whatever the private information of other agents. We allow for payments by agents to a central bank, which is the *only* central authority that we require. To achieve faithfulness, we carefully integrate the Vickrey-Clarke-Groves (VCG) mechanism with the *DPOP* algorithm, such that *each agent is only asked to perform computation, report information, and send messages that is in its own best interest*. Determining agent i's payment requires solving the social choice problem without agent i. Here, we present a method to *reuse computation* performed in solving the main problem in a way that is robust against manipulation by the excluded agent. Experimental results on structured problems show that as much as 87% of the computation required for solving the marginal problems can be avoided by re-use, providing very good scalability in the number of agents. On unstructured problems, we observe a sensitivity of M-DPOP to the density of the problem, and we show that reusability decreases from almost 100% for very sparse problems to around 20% for highly connected problems. We close with a discussion of the features of DCOP that enable faithful implementations in this problem, the challenge of reusing computation from the main problem to marginal problems in other algorithms such as *ADOPT* and *OptAPO*, and the prospect of methods to avoid the welfare loss that can occur because of the transfer of payments to the bank.

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. 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 massively distributed problems such as meeting scheduling, where the decisions are about when and where to hold each meeting, or allocating airport landing slots to airlines, where the decisions are what airline is allocated what slot, or scheduling contractors in construction projects.

One approach to solve such problems is with a central authority that computes the optimal solution. In combination with an *incentive mechanism* such as the Vickrey-Clarke-Groves (VCG) mechanism (Jackson, 2000), we can also prevent manipulation through the misreporting of preferences. 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 web of dependencies that is hard to optimize in a centralized fashion. Privacy concerns also favor decentralized solutions (Greenstadt, Pearce, & Tambe, 2006).

Algorithms for distributed constraint reasoning, such as ABT and AWC (Yokoo & Hirayama, 2000), AAS (Silaghi, Sam-Haroud, & Faltings, 2000), DPOP (Petcu & Faltings, 2005b) and ADOPT (Modi, Shen, Tambe, & Yokoo, 2005), can deal with large problems as long as the influence of each agent on the solution is limited to a bounded number of variables. However, the current techniques assume *cooperative agents*, and do not provide robustness against misreports of preferences of deviations from the algorithm by self-interested agents. This is a major limitation. In recent years, *faithful distributed implementation* (Parkes & Shneidman, 2004) has been proposed as a framework within which to achieve a synthesis of the methods of (centralized) MD with distributed problem solving. *Faithfulness* ensures that no agent can benefit by unilaterally deviating from any aspect of the protocol, neither information-revelation, computation, nor communication, and whatever the private information of other agents. Until now, distributed implementation has been applied to lowest-cost routing (Shneidman & Parkes, 2004; Feigenbaum, Papadimitriou, Sami, & Shenker, 2002), and policy-based routing (Feigenbaum, Ramachandran, & Schapira, 2006), on the Internet, but not to efficient social choice, a problem with broad applicability.

In this paper, we make the following contributions:

- We show how to model the problem of efficient social choice as a DCOP, and adapt the DPOP algorithm to exploit the local structure of the distributed model and achieve the same scalability as would be possible in solving the problem on a centralized problem graph.

- We provide an algorithm whose first stage is to *faithfully* generate the DCOP representation from the underlying social choice problem. Once the DCOP representation is generated, the next stages of our *M-DPOP* algorithm are also faithful, and form an *ex post Nash* equilibrium of the induced non-cooperative game.

- In establishing that DCOP models of social choice problems can be solved faithfully, we observe that the *communication and information structure* in the problem are such that no agent can prevent the rest of the system, in aggregate, from correctly determining the marginal impact that allowing for the agent's (reported) preferences has on the total utility achieved by the other agents. This provides the generality of our techniques to other DCOP algorithms.

- Part of achieving faithfulness requires solving the DCOP with each agent's (reported) preferences ignored in turn, and doing so without this agent able to interfere with this computational process. We provide an algorithm with this robustness property, that is nevertheless able to reuse, where possible, intermediate results of computation from solving the main problem with all agents.

- In experimental analysis, on structured meeting scheduling problems that are a common benchmark in the literature, we demonstrate that as much as 87% of the computation required for solving the marginal problems can be avoided through reuse. Results are also provided for unstructured resource allocation problems [1], and show M-DPOP to be sensitive to problem density: for loose problems, up to around 80% of the computation can be reused, and this decreases for highly connected problems.

The M-DPOP algorithm defines a *strategy* for each agent in the extensive-form game induced by the DCOP for efficient social choice. In particular, the M-DPOP algorithm defines the messages that an agent should send, and the computation that an agent should perform, in response to messages received from other agents. In proving that M-DPOP forms a game-theoretic equilibrium, we show that no agent can benefit by unilaterally deviating, whatever the utility functions of other agents and whatever the constraints. Although not as robust as a *dominant strategy equilibrium*, because this (*ex post*) equilibrium requires every other agent to follow the algorithm, Parkes and Shneidman (2004) have earlier commented that this appears to be the necessary "cost of decentralization."

The total payment made by each agent to the bank is always non-negative and M-DPOP never runs at a deficit (i.e. the bank always receives a non-negative net payment from the agents). In some settings, this transfer of utility to the bank is undesirable and would be best avoided. We provide some statistics for the problem domains studied that show that this loss can represent as much as 35% of the total utility achieved from the solution in some problems studied. While the payments cannot be naively redistributed back to agents without breaking faithfulness, extant work on redistribution mechanisms for VCG payments suggests that this can be mitigated (Guo & Conitzer, 2007; Faltings, 2004; Cavallo, 2006; Moulin, 2007; Bailey, 1997). We defer this extension to M-DPOP, the details of which are surprisingly involved and interesting in their own right, to future work.

The reuse of computation, in solving the marginal problems with each agent removed in turn, is especially important in settings of *distributed* optimization because motivating scenarios are those for which the problem size is massive, perhaps spanning multiple organizations and encompassing thousands of decisions. For example, consider project scheduling, inter-firm logistics, intra-firm meeting scheduling, etc. With appropriate problem structure, DCOP algorithms in these problems can scale linearly in the size of the problem. For instance, DPOP is able to solve such problems through a single back-and-forth traversal over the problem graph. But *without re-use the additional cost of solving each marginal problem would make the computational cost quadratic rather than linear* in the number of agents, which could be untenable in such massive-scale applications.

The rest of this paper is organized as follows: after preliminaries (Section 2), in Section 3 we describe the DPOP (Petcu & Faltings, 2005b) algorithm for distributed constraint optimization, which is the focus of our study. Section 4 introduces our model of self-interested agents and defines the (centralized) VCG mechanism. Section 4.4 provides a simple method, *Simple M-DPOP* to make DPOP faithful and serves to illustrate the excellent fit between the information and communication structure of DCOPs and faithful VCG mechanisms. In Section 5 we describe our main algorithm, M-DPOP, in which computation is re-used in solving the marginal problems with each agent removed

1. We consider distributed combinatorial auctions, with instances randomly generated using a distribution in the CATS problem suite (Leyton-Brown & Shoham, 2006).

in turn. We present experimental results in Section 6. In Section 7 we discuss adapting other DCOP algorithms for social choice (ADOPT and OptAPO, see Section 7.2), and about the waste due to payments in Section 7.3. We conclude in Section 8.

1.1 Related Work

This work draws on two research areas: distributed algorithms for constraint satisfaction and optimization, and mechanism design for coordinated decision making in multi-agent systems with self-interested agents. We briefly overview the most relevant results in these areas.

1.1.1 Constraint Satisfaction and Optimization

Constraint satisfaction and optimization are powerful paradigms that can model a wide range of tasks like scheduling, planning, optimal process control, etc. Traditionally, such problems were gathered into a single place, and a centralized algorithm was applied to find a solution. However, social choice problems are naturally distributed, and often preclude the use of a centralized entity to gather information and compute solutions.

The Distributed Constraint Satisfaction (DisCSP) (Yokoo, Durfee, Ishida, & Kuwabara, 1992; Sycara, Roth, Sadeh-Koniecpol, & Fox, 1991; Collin, Dechter, & Katz, 1991, 1999; Solotorevsky, Gudes, & Meisels, 1996) and the Distributed Constraint Optimization (DCOP) (Modi et al., 2005; Zhang & Wittenburg, 2003; Petcu & Faltings, 2005b; Gershman, Meisels, & Zivan, 2006) formalisms were introduced to enable distributed solutions. The agents involved in such problems must communicate with each other to find a solution to the overall problem (unknown to any one of them). Briefly, these problems consist of individual subproblems (each agent holds its own subproblem), which are connected with (some of) its peers' subproblems via *constraints* that limit what each individual agent can do. The goal is to find feasible solutions to the overall problem (in the case of DisCSP), or optimal ones in the case of DCOP.

Many distributed algorithms for DCOP have been introduced, none of which deals with self-interested agents. The most well known ones are ADOPT, DPOP and OptAPO:

- ADOPT (Modi et al., 2005) is a backtracking based, bound propagation algorithm. ADOPT is completely decentralized and message passing is asynchronous. While ADOPT has the advantage of requiring linear memory, and linear-size messages, its applicability for large problems [2] is questionable due to the fact that it produces a number of messages which is exponential in the depth of the DFS tree chosen.

- OptAPO (Mailler & Lesser, 2005) is a centralized-distributed hybrid that uses *mediator nodes* to centralize subproblems and solve them in dynamic and asynchronous mediation sessions. The authors show that its message complexity is significantly smaller than ADOPT's. However, it is designed for cooperative settings, and in settings with self-interested agents like the social choice problem, it is unclear whether agents would agree revealing their constraints and utility functions to (possibly many) other agents, such that they can solve the partially centralized subproblems.

- *DPOP* (Petcu & Faltings, 2005b) is a complete algorithm based on dynamic programming which generates only a linear number of messages. In DPOP, the size of the messages depends

2. The largest ADOPT experiments that we are aware of comprise problems with around 20 agents and 40 variables.

on the structure of the problem: the largest message is exponential in the *induced width* of the problem (see Section 3.1.4) As with ADOPT, DPOP maintains the full distribution of the problem. These features suggest that DPOP is a good foundation for an efficient distributed implementation of a VCG-based mechanism for social choice problems.

A further discussion about the features of these algorithms and their applicability to social choice problems is provided in Section 7. In this paper, we will focus on DPOP and provide appropriate modifications and payments so that it can be effective for environments with self-interested agents. In Section 7.2 we will also provide a brief discussion about the opportunites and challenges in applying our methodology to ADOPT and OptAPO.

1.1.2 MECHANISM DESIGN AND DISTRIBUTED IMPLEMENTATION

There is a long tradition of using *centralized* incentive mechanisms within Distributed AI, going back at least to the work of Ephrati and Rosenschein (1991) who considered the use of the VCG mechanism to compute joint plans; see also the work of Sandholm (1996) and Parkes et al. (2001) for more recent discussions. Also noteworthy is the work of Rosenschein and Zlotkin (1994, 1996) on *rules of encounter*, which provided non-VCG based approaches for task allocation in systems with two agents.

On the other hand, there are very few known methods for *distributed problem solving* in the presence of self-interested agents. For example, while TRACONET (Sandholm, 1993) improved upon the CONTRACTNET system (Davis & Smith, 1983) of negotiation-based, distributed task reallocation, by providing better economic realism, TRACONET was nevertheless studied for simple, myopically-rational agent behaviors and its performance with game-theoretic agents was never analyzed; this remains true for more recent works (Endriss, Maudet, Sadri, & Toni, 2006; Dunne, Wooldridge, & Laurence, 2005; Dunne, 2005). Similarly, Wellman's work on *market-oriented programming* (Wellman, 1993, 1996) considers the role of virtual markets in the support of optimal resource allocation, but is developed for a model of "price-taking" agents (i.e. agents that treat current prices as though they are final), rather than game-theoretic agents.

The first step in providing a more satisfactory synthesis of distributed algorithms with MD was provided by the agenda of *distributed algorithmic mechanism design* (DAMD), due to the work of Feigenbaum and colleagues (Feigenbaum et al., 2002; Feigenbaum & Shenker, 2002). These authors (FPSS) provided an efficient algorithm for lowest-cost interdomain routing on the Internet, terminating with optimal routes and the payments of the VCG mechanism. The up-shot was that agents– in this case *autonomous systems* running network domains –could not benefit by misreporting information about their own transit costs. But missing from this analysis was any consideration about the robustness of the *algorithm itself* to manipulation. *Distributed implementation* (Parkes & Shneidman, 2004) introduces this additional requirement. An algorithm is *faithful* if an agent cannot benefit by deviating from any of its required actions, including information-revelation, computation and message passing. A number of principles for achieving faithfulness in an *ex post* Nash equilibrium are provided by Parkes and Shneidman (2004). By careful incentive design and a small amount of cryptography they are able to remove the remaining opportunities for manipulation from the lowest-cost routing algorithm of FPSS. Building on this, Feigenbaum et al. (2006) recently provide a faithful method for *policy-based* interdomain routing, better capturing the typical business agreements between Internet domains.

Ours is the first work to achieve faithfulness for general DCOP algorithms, demonstrated here via application to efficient social choice. In other work, Monderer and Tennenholtz (1999) consider a distributed single item allocation problem, but focus on (faithful) *communication* and do not provide distributed computation. Izmalkov et al. (2005) adopt cryptographic primitives such as *ballot boxes* to show how to convert *any* centralized mechanisms into a DI on a *fully connected* communication graph. There interest is in demonstrating the theoretical possibility of "ideal mechanism design" without a trusted center. Our work has a very different focus: we seek computational tractability, do not require fully connected communication graphs, and make no appeal to cryptographic primitives. On the other hand, we are content to retain desired behavior in *some* equilibrium (remaining consistent with the MD literature) while Izmalkov et al. avoid the introduction of any additional equilibria beyond those that exist in a centralized mechanism.

We briefly mention two other related topics. Of note is the well established literature on *iterative VCG mechaisms* (Mishra & Parkes, 2007; Ausubel, Cramton, & Milgrom, 2006; Bikhchandani, de Vries, Schummer, & Vohra, 2002). These provide a *partially* distributed implementation for combinatorial allocation problems, with the center typically issuing "demand queries" of agents via prices, these prices triggering computation on the part of agents in generating a demand set in response. These auctions can often be interpreted as decentralized primal-dual algorithms (Parkes & Ungar, 2000; de Vries & Vohra, 2003). The setting differs from ours in that there remains a center that performs computation, solving a winner determination problem in each round, and each agent communicates directly with the center and not peer-to-peer. Mu'alem (2005) initiates an orthogonal direction within computer science related to the topic of *Nash implementation* (Jackson, 2001) in economics, but her approach relies on information that is part private and part common knowledge, so that no one agent has entirely private information about its preferences.

2. Preliminaries: Modeling Social Choice

We assume that the social choice problem consists of a finite but possibly large 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 discrete and finite domain. Each agent has private information about the variables on which it places *relations*. Each relation associated with an agent defines the utility of that agent for each possible assignment of values to the variables in the domain of the relation. There may also be hard constraints that restrict the space of feasible joint assignments to subsets of variables.

Definition 1 (Social Choice Problem - SCP) *An efficient social choice problem can be modeled as a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R} >$ *such that:*

- $\mathcal{X} = \{X_1, ..., X_m\}$ *is the set of* ***public decision variables*** *(e.g. when and where to hold meetings, to whom should resources be allocated, etc);*
- $\mathcal{D} = \{d_1, ..., d_m\}$ *is the set of finite* ***public domains*** *of the variables* $\mathcal{X}$ *(e.g. list of possible time slots or venues, list of agents eligible to receive a resource, etc);*
- $\mathcal{C} = \{c_1, ..., c_q\}$ *is a set of* ***public constraints*** *that specify the feasible combinations of values of the variables involved. A* **constraint** c_j *is a function* $c_j : d_{j_1} \times .. \times d_{j_k} \rightarrow \{-\infty, 0\}$ *that returns 0 for all allowed combinations of values of the involved variables, and* $-\infty$ *for disallowed ones. We denote by* $scope(c_j)$ *the set of variables associated with constraint* c_j*;*

- $\mathcal{A} = \{A_1, ..., A_n\}$ *is a set of* ***self-interested agents*** *involved in the optimization problem;* $X(A_i) \subseteq \mathcal{X}$ *is a (***privately known***)* [3] *set of variables in which agent* A_i *is "interested" and on which it has relations.*

- $\mathcal{R} = \{R_1, ..., R_n\}$ *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* $r_i^j : d_{j_1} \times .. \times d_{j_k} \rightarrow \mathbb{R}$ *specified by agent* A_i*, which denotes the utility* A_i *receives for all possible values on the involved variables* $\{j_1, \ldots, j_k\}$ *(negative values mean costs). We denote by* $scope(r_i^j)$ *the domain of variables that* r_i^j *is defined on.*

The private relations of each agent may, themselves, be induced by the solution to local optimization problems on additional, private decision variables and with additional, private constraints. These are kept local to an agent and not part of the SCP definition.

The optimal solution to the SCP is a complete instantiation X^* of all variables in $\mathcal{X}$, s.t.

$$X^* \in \arg\max_{X \in \mathcal{D}} \sum_{i \in \{1,..,n\}} R_i(X) + \sum_{c_j \in \mathcal{C}} c_j(X), \tag{1}$$

where $R_i(X) = \sum_{r_i^j \in R_i} r_i^j(X)$ is agent A_i's *total utility* for assignment X. This is the natural problem of social choice: the goal is to find a solution that maximizes the total utility of all agents, while respecting hard constraints; notice that the second sum is $-\infty$ if X is infeasible and precludes this outcome. We assume throughout that there is a feasible solution. In introducing the VCG mechanism we will require the solution to the SCP with the influence of each agent's relations removed in turn. For this, let $SCP(\mathcal{A})$ denote the main problem in Eq. (1) and $SCP(-A_i)$ denote the *marginal problem without agent* A_i, i.e. $\max_{X \in \mathcal{D}} \sum_{j \neq i} R_j(X) + \sum_{c_j \in \mathcal{C}} c_j(X)$. Note that all decision variables remain. The only difference between $SCP(\mathcal{A})$ and $SCP(-A_i)$ is that the preferences of agent A_i are ignored in solving $SCP(-A_i)$.

For variable X_j, refer to the agents A_i for which $X_j \in X(A_i)$ as forming the **community** for X_j. We choose to emphasize the following assumptions:

- Each agent knows the variables in which it is interested, together with the domain of any such variable and the hard constraints that involve the variable.

- Each decision variable is supported by a *community mechanism* that allows all interested agents to report their interest and learn about each other. For example, such a mechanism can be implemented using a bulletin board.

- For each constraint $c_j \in \mathcal{C}$, every agent A_k in a community $X_l \in scope(c_j)$, i.e. with $X_l \in X(A_k)$, can read the membership lists of all other communities $X_m \in scope(c_j)$ for $X_m \neq X_l$. In other words, every agent involved in a hard constraint knows about all other agents involved in that hard constraint.

- Each agent can communicate directly with all agents in all communities in which it is a member, and with all other agents involved in the same shared hard constraints. No other communication between agents is required.

3. Note that the private knowledge of variables of interest is not a requirement; the algorithms we present work with both public and private knowledge of variables of interest. What is required is that agents interested in the same variable know about each other - see assumptions below.

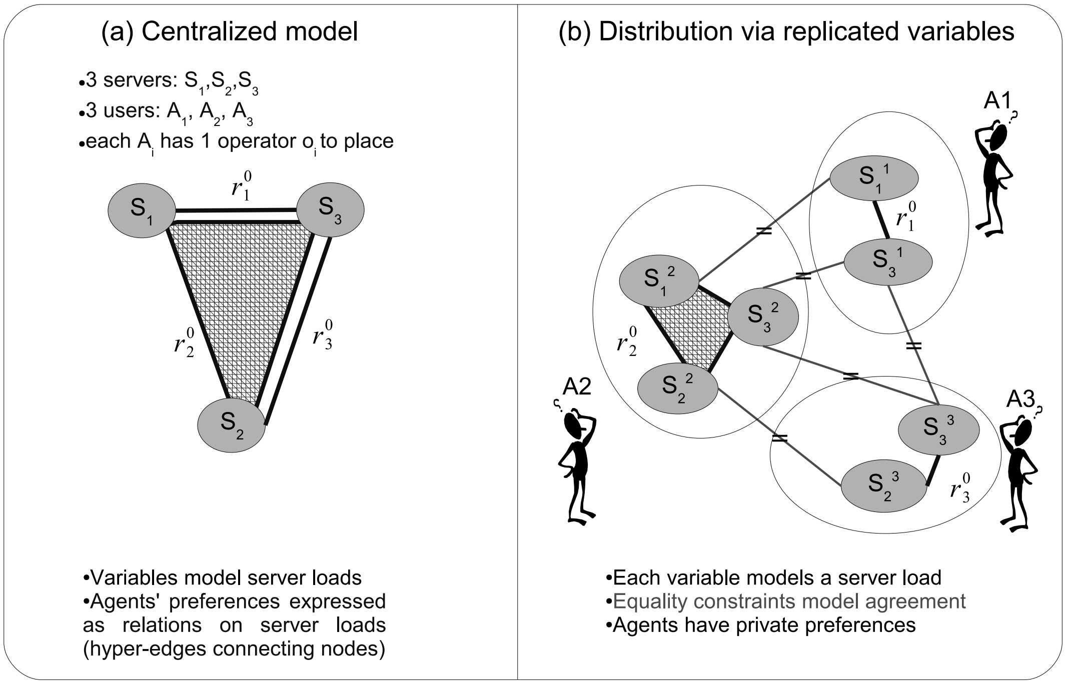


Figure 1: An operator placement problem: **(a)** A centralized model (each variable is a server load –possible values are feasible combinations of services to be run by each server –, and the edges correspond to relations and represent agent preferences). **(b)** A decentralized (DCOP) model with replicated variables. Each agent has a local replica of variables of interest and inter-agent edges denote equality constraints that ensure agreement. The preferences modeled by relations are now hyper-edges local to the respective agents.

In Section 4 we will establish that the step of identifying the SCP, via the community mechanism, is itself *faithful* so that self-interested agents will choose to volunteer the communities of which they are a member (and only those communities.)

2.1 Modeling Social Choice as Constraint Optimization

We first introduce a centralized, constraint optimization problem (COP) model of the efficient social choice problem. This model is represented as a *centralized problem graph*. Given this, we then model this as a distributed constraint optimization problem (DCOP), along with an associated *distributed problem graph*. The distributed problem graph makes explicit the control structure of the distributed algorithm that is ultimately used by the multi-agent system to solve the problem. Both sections are illustrated by reference to an overlay network optimization problem (Huebsch, Hellerstein, Lanham, et al., 2003; Faltings, Parkes, Petcu, & Shneidman, 2006; Pietzuch, Ledlie, Shneidman, Roussopoulos, Welsh, & Seltzer, 2006):

OVERLAY NETWORK OPTIMIZATION Consider the problem of optimal placement of data aggregation and processing operators on an *overlay network* such as a large-scale sensor network (Huebsch et al., 2003; Pietzuch et al., 2006). In this application, there are multiple users and multiple servers. Each user is associated with a query and has a client machine located at a particular node on an overlay network. A query has an associated set of *data producers*, known to the user and located at nodes on the network. Each query also requires a set of data aggregation and processing operators, which should be placed on server nodes between the nodes with data producers and the user's node. Each user assigns a utility to different assignments of operators to servers to represent her preferences for different kinds of data aggregation. Examples of in-network operators for data aggregation include database style "join" operators; e.g., a user may desire "volcano data X" and "earthquake data Y" joined and sent to them. To address this, a specific operator that we call "VolcanoXEarthquakeY_Join" is created and put into the network. Naturally, each user prefers to have their operators placed on the "best" servers in the network, without regard to the costs incurred, overloading servers, denying service to other users, etc. The problem is to find the optimal allocation of operators to servers, subject to capacity and compatibility constraints.

Faltings et al. (2006) model this problem as one of efficient social choice. A distributed algorithm, to be executed by user clients situated on network nodes, is used to determine the assignment of data aggregation and processing operators to server nodes.

2.1.1 A CENTRALIZED COP MODEL AS A MULTIGRAPH

Viewed as a centralized problem, the SCP can be defined as a constraint optimization problem on a *multigraph*, i.e. a graph in which several distinct edges can connect the same set of nodes. We denote this $COP(\mathcal{A})$, and provide an illustration in Figure 1(a). The decision variables are the nodes, and relations defined over subsets of the variables form edges of the multigraph; *hyper*edges that connect more than two vertices at once in the case of a relation involving more than two variables. There can be multiple edges that involve the same set of variables, with each edge corresponding to the relations of a distinct agent on the same set of variables. The hard constraints are also be represented as edges on the graph.

Example 1 (Centralized Model for Overlay Optimization) *The example in Figure 1(a) contains 3 users A_i and 3 servers S_j. For simplicity reasons, assume that each user A_i has one single operator o_i that they want to have executed on some server. According to prerequisites and compatibility issues, assume that S_1 can execute both o_1 and o_2, but not o_3. Similarly, assume that S_2 can execute both o_2 and o_3, but not o_1, and S_3 can execute any combination of at most two out of the three operators. Agents have preferences about where their operators are executed (e.g. because of proximity to data sources, computational capabilities of the servers, cost of electricity, etc). For example, A_1 extracts utility 10 when o_1 is executed by S_1, and utility 5 when o_1 is executed by S_3.*

To model the problem as an optimization problem, we use the following:

1. variables*: for each server S_i, we create a variable S_i that denotes the set of operators that S_i will execute.*

2. values*: each variable S_i can take values from the set of all possible combinations of operators the server can execute. For example, $S_1 = \{null, o_1, o_2, o_1 + o_2\}$, where $null$ means the server executes no operator, o_i that it executes operator o_i, and $o_1 + o_2$ that it executes both o_1 and o_2.*

3. constraints*: restrict the possible combinations of assignments. Example: no two servers should execute the same operator.*

4. relations*: allow agents to express preferences about combinations of assignments. A_1 models its preference for the placement of o_1 by using the relation r_1^0, defined over the variables S_1*

and S_3. This relation associates an utility value to each combination of assignments to S_1 and S_3 (in total $4 \times 8 = 32$ combinations) as follows:

- *0 to all combinations where o_1 is executed neither on S_1, or on S_3 (e.g. $\langle S_1 = o_2, S_3 = o_3 \rangle$)*
- *10 to all combinations where o_1 is executed only on S_1 (e.g. $\langle S_1 = o_1, S_3 = o_2 + o_3 \rangle$)*
- *5 to all combinations where o_1 is executed only on S_3 (e.g. $\langle S_1 = o_2, S_3 = o_1 \rangle$)*

We depict variables as nodes in a graph, and constraints and relations as (hyper)edges (see Figure 1(a)). The problem can get arbitrarily complex, with multiple operators per agent, groups of servers being able to execute only certain groups of compatible operators, etc.

2.1.2 A DECENTRALIZED COP (DCOP) MODEL USING REPLICATED VARIABLES

It is useful to define an alternate graphical representation of the SCP, with the centralized problem graph replaced with a *distributed* problem graph. This distributed problem graph has a direct correspondence with the DPOP algorithm for solving DCOPs. We denote by $DCOP(\mathcal{A})$ the problem with all agents included, which corresponds to the main social choice problem, $SCP(\mathcal{A})$. Similarly, $DCOP(-A_i)$ is the problem with agent A_i removed, which corresponds to $SCP(-A_i)$. In our distributed model, each agent has a *local replica* of the variables in which it is interested.[4] For each public variable, $X_v \in X(A_i)$, in which agent A_i is interested, the agent has a *local replica*, denoted X_v^i. Agent A_i then models its local problem $COP(X(A_i), R_i)$, by specifying its relations $r_i^j \in R_i$ on the locally replicated variables.

Refer to Figure 1(b) for the translation of the centralized problem from Figure 1(a) into a DCOP model. Each agent has as local variables the loads of the servers that are of interest to itself, i.e. servers that can execute one of its operators (e.g. S_1^2 represents A_2's local replica of the variable representing server S_1). Local edges correspond to local *all-different* constraints between an agent's variables and ensure that it does not execute its operator on several servers at the same time. Equality constraints between local replicas of the same value ensure global agreement about what operators will run on which servers.

Agents specify their relations via local edges on local replicas. For example, agent A_1 with its relation on the load of servers S_1 and S_3 can now express a preference for the placement of its operator o_1 with relation r_1^0, which can assign e.g. utility 5 to S_3 executing o_1, and utility 10 to S_1 executing o_1.

We can begin to understand the potential for manipulation by self-interested agents through this example. Notice that although the globally optimal solution may require assigning o_1 to S_3, this is less preferable to A_1, providing utility 5 instead of 10. Therefore, in the absence of an incentive mechanism, A_1 could benefit from a simple manipulation: declare utility $+\infty$ for $\langle S_1 = o_1 \rangle$, thus changing the final assignment to a suboptimal one that is nevertheless better for itself.

4. An alternate model designates an "owner" agent for each decision variable. Each owner agent would then centralize and aggregate the preferences of other agents interested in its variable. Subsequently, the owner agents would use a distributed optimization algorithm to find the optimal solution. This model limits the reusability of computation from the main problem in solving the marginal problems in which each agent is removed in turn because when excluding the owner agent of a variable, one needs to assign ownership to another agent and restart the computational process in regards to this variable and other connected variables. This reuse of computation is important in making M-DPOP scalable. Our approach is disaggregated and facilitates greater reuse.

The *neighborhood* of each local copy X_v^i of a variable is composed of three kinds of variables:

$$Neighbors(X_v^i) = Siblings(X_v^i) \cup Local\_neighbors(X_v^i) \cup Hard\_neighbors(X_v^i). \tag{2}$$

The siblings are local copies of X_v that belong to other agents $A_j \neq A_i$ also interested in X_v:

$$Siblings(X_v^i) = \{X_v^j \mid A_j \neq A_i \text{ and } X_v \in X(A_j)\} \tag{3}$$

All siblings of X_v^i are connected pairwise with an *equality constraint*. This ensures that all agents eventually have a consistent value for each variable. The second set of variables are the local neighbors of X_v^i from the local optimization problem of A_i. These are the local copies of the other variables that agent A_i is interested in, which are connected to X_v^i via relations in A_i's local problem:

$$Local\_neighbors(X_v^i) = \{X_u^i \mid X_u \in X(A_i), \text{ and } \exists r_i^j \in R_i \text{ s.t. } X_u^i \in scope(r_i)\} \tag{4}$$

We must also consider the set of *hard constraints* that contain in their scope the variable X_v and some other public variables: $Hard(X_v) = \{\forall c_s \in C | X_v \in scope(c_s)\}$. These constraints connect X_v with all the other variables X_u that appear in their scope, which may be of interest to some other agents as well. Consequently, X_v^i should be connected with all local copies X_t^j of the other variables X_t that appear in these hard constraints:

$$Hard\_neighbors(X_v^i) = \{X_t^j | \exists c_s \in Hard(X_v) \text{ s.t. } X_t \in scope(c_s), \text{ and } X_t \in X(A_j)\} \tag{5}$$

In general, each agent can also have *private variables*, and relations or constraints that involve private variables, and link them to the public decision variables. For example, consider a meeting scheduling application for employees of a company. Apart from the work-related meetings they schedule together, each one of the employees also has personal items on her agenda, like appointments to the doctor, etc. Decisions about the values for private variables and information about these local relations and constraints remain private. These provide no additional complications and will not be discussed further in the paper.

2.2 Example Social Choice Problems

Before continuing to present our main results we describe three additional problems of social choice that serve to motivate our work. In fact, the problem of efficient social choice is fundamental to microeconomics and political science (Mas-Colell, Whinston, & Green, 1995). Each problem that we present is both large scale and distributed, and involves actors in the system that are businesses and cannot be expected to cooperate, either in revealing their preferences or in following the rules of a distributed algorithm.

AIRPORT SLOT ALLOCATION. As airports become more congested, governments are turning to market-based approaches to allocate landing and takeoff slots. For instance, the U.S. Federal Aviation Administration recently commisioned a study on the use of an auction to allocate slots at New York's congested LaGuardia airport (Ball, Donohue, & Hoffman, 2006). This problem is large scale when it expands to include airports throughout the U.S., and eventually the World, exhibits self-interest (airlines are profit-maximizing agents with private information about their utilities for different slot allocations), and is one in which privacy is a major concern because of the competitiveness of the airline industry. A typical policy goal is to maximize the total utility of the allocation, i.e. one of efficient social choice. This problem motivates our study of *combinatorial auctions* in Section 6. A combinatorial auction (CA) is one in which a set of heterogeneous, indivisible goods are to be allocated to agents, each of which has values expressed on sets of goods; e.g., "I only want the 9am slot if I also get the 10am slot" or "I am indifferent between the 9am and the 9:05am slot." The airport slot allocation problem motivated the first paper on CAs (Rassenti, Smith, & Bulfin, 1982), in which it was recognized that airlines would likely need to express utilities on sets of slots that correspond to the right to fly a *schedule* in and out of an airport.

OPEN-ACCESS WIRELESS NETWORKS. Most wireless spectrum today is owned and operated as closed networks, for example by cellular companies such as T-Mobile and AT&T. However there is plenty of debate about creating open-access wireless networks in which bandwidth must be available for use on any phone and any software.[5] Some have recently proposed using an auction protocol to allow service providers to bid in a dynamic auction for the right to use spectrum for a given period of time to deliver services.[6] Taken to its logical conclusion, and in an idea anticipated by Rosenschein and Zlotkin (1994) for wired telephony, this suggests a secondary market for wireless spectrum and corresponds to a problem of efficient social choice: allocate spectrum to maximize the total utility of consumers. This problem is large scale, exhibits self-interest, and is inherently decentralized.

THE MEETING SCHEDULING PROBLEM. Consider a large organization with dozens of departments, spread across dozens of sites, and employing tens of thousands of people. Employees from different sites and departments want to setup thousands of meetings each week. Due to privacy concerns among different departments, centralized problem solving is not desirable. Furthermore, although the organization as a whole desires to minimize the cost of the whole process, each department and employee is self interested in that it wishes to maximize its own utility. An artificial currency is created for this purpose and a weekly assignment is made to each employee. Employees express their preferences for meeting schedules in units of this currency.

Refer to Figure 2 for an example of such a problem, where 3 agents want to setup 3 meetings. Figure 2(b) shows that each agent has as local variables the time slots corresponding to the meetings it participates in (e.g. M_1^2 represents A_2's local replica of the variable representing meeting M_1). Local edges correspond to local *all-different* constraints between an agent's variables and ensure that it does not participate in several meetings at the same time. Equality constraints between local replicas of the same value ensure global agreement. Agents specify their relations via local edges on local replicas. For example, agent A_1 with its relation on the time of meeting M_1 can now express a preference for a meeting later in the day with relation r_1^0, which can assign low utilities to morning time slots and high utilities to afternoon time slots. Similarly, if A_2 prefers holding meeting M_2 *after* meeting M_1, then it can use the local relation r_2^0 to assign high utilities to all satisfactory combinations of timeslots and low utility otherwise. For example, $\langle M_1 = 9AM, M_2 = 11AM\rangle$ gets utility 10, and $\langle M_1 = 9AM, M_2 = 8AM\rangle$ gets utility 2.

5. In a breakthrough ruling, the U.S. Federal Communications Commission (FCC) will require open access for around one-third of the spectrum to be auctioned in early '08. But it stopped short of mandating that this spectrum be made available in a wholesale market to would be service providers. See http://www.fcc.gov/073107/700mhz_news_release_073107.pdf
6. Google proposed such an auction in a filing made to the FCC on May 21st, 2007. See http://gullfoss2.fcc.gov/prod/ecfs/retrieve.cgi?native_or_pdf=pdf&id_document=6519412647.

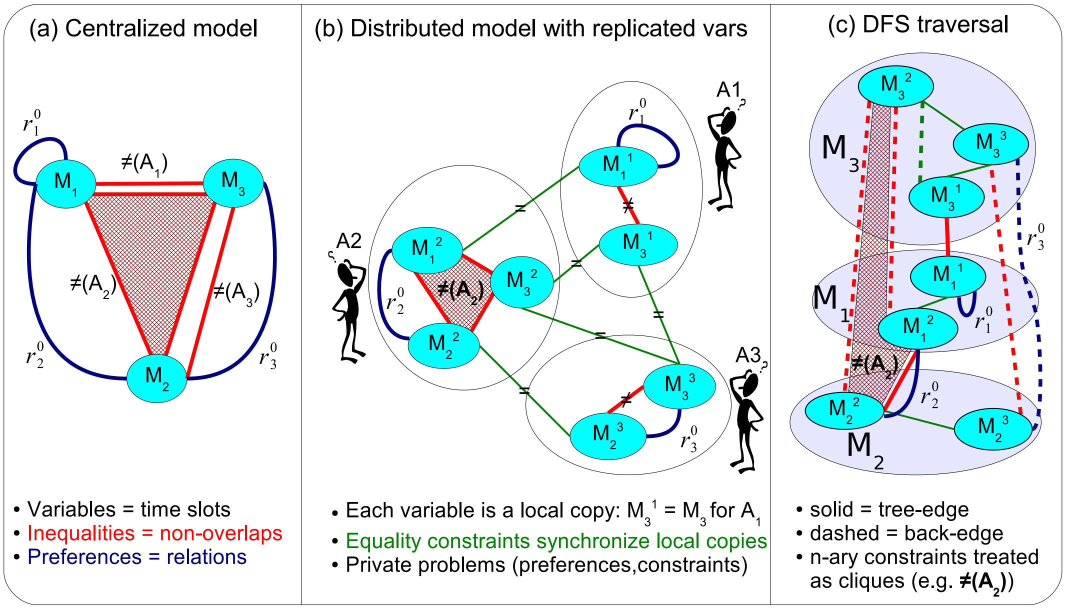


Figure 2: A meeting scheduling problem. **(a)** A centralized model (each vertex is a meeting variable, red edges correspond to hard constraints on non-overlap for meetings that share a participant (that for agent A_2 is a hyperedge because it particpates in every meeting), blue edges correspond to relations and represent agent preferences). **(b)** A decentralized (DCOP) model with replicated variables. Each agent has a local replica of variables of interest and inter-agent edges denote equality constraints that ensure agreement. The hard constraint for non-overlap between meetings M_1, M_2 and M_3 is now a local hyperedge to agent A_2. **(c)** A DFS arrangement of the decentralized problem graph. Used by the DPOP algorithm to control the order of problem solving.

In the experimental results presented in Section 6 we adopt meeting scheduling as prototypical of *structured* social choice problems with the problem instances associated with an organizational hierarchy. Meeting scheduling was introduced in Section 2.1. For a second set of experiments we consider combinatorial auctions (CAs), in which agents bid for *bundles* of goods, and there we consider a set of problem instances that are *unstructured* and provide a comparison point to that of meeting scheduling. CAs provide a nice abstraction of the kinds of allocation problems that exist in the airport and wireless network domains.

3. Cooperative Case: Efficient Social Choice via DPOP

In this section, we review DPOP (Petcu & Faltings, 2005b), which is a general purpose distributed optimization algorithm. DPOP (Distributed Pseudotree Optimization Protocol) is based on dynamic programming and adapts Dechter's (Dechter, 2003) general bucket elimination scheme to the distributed case. Its main advantage is that it only generates a linear number of messages. This is in contrast to other optimization algorithms like ADOPT (Modi et al., 2005) and ensures minimal network overhead produced by message exchange. On the other hand, a concern in DPOP can be the size of individual messages since this grows exponentially with a parameter of the constraint graph called the *induced width* (see Section 3.1.4). Nevertheless, for problems that exhibit local structure, DPOP typically scales to much larger problems, and is orders of magnitude more efficient, than other techniques (Petcu & Faltings, 2005b, 2007). To simplify the exposition, we first illustrate DPOP in a general DCOP context, and then show how to instantiate DPOP for social choice problems. In particular, we explain how to leverage the structure provided by local replicas. We consider only cooperative agents throughout this section.

3.1 The DPOP Algorithm for DCOPs

This section presents the DPOP algorithm for generic DCOPs. To simplify the exposition, we assume – in this section only – that each agent A_i represents a single variable X_i, and that the constraint graph is given.

DPOP is composed of three phases:

- Phase one constructs a "DFS arrangement", $DFS(\mathcal{A})$, which defines the control flow of message passing and computation in DPOP.
- Phase two is a bottom-up utility propagation along the tree constructed in phase 1. In this phase utilities for different values of variables are aggregated to reflect optimal decisions that will be made in subtrees rooted at each node in the tree.
- Phase three is a top-down value assignment propagation along the tree constructed in phase 1. In this phase decisions are made based on the aggregate utility information from phase 2.

In describing these phases we refer to Figure 3 for a running example. We also introduce an explicit numerical example to illustrate phases two and three in more detail.

3.1.1 DPOP Phase One: DFS Tree Generation

This first phase performs a depth-first search (DFS) traversal of the problem graph, thereby constructing a DFS *arrangement* of the problem graph. The DFS arrangement is subsequently used to provide control flow in DPOP and guide the variable elimination order. When the underlying problem graph is a tree then the DFS arrangement will also be a tree. In general, the DFS arrangement is a graph that we define as the union of a set of *tree edges* and additional *back edges*, which connect some of the nodes with their ancestors.[7]

Definition 2 (DFS arrangement) *A DFS arrangement of a graph G defines a rooted tree on a subset of the edges (the* ***tree edges****) with the remaining edges included as* ***back edges****. The tree edges are defined so that adjacent nodes in G fall in the same branch of the tree.*

Figure 3 shows an example DFS arrangement. The tree edges are shown as solid lines (e.g. $1-3$) and the back edges are shown as dashed lines (e.g. $12-2$, $4-0$). Two nodes X_i and X_v are said to be "in the same branch" of the DFS arrangement if there is a path from the higher node to the lower node along tree edges; e.g., nodes X_0 and X_{11} in Figure 3. DFS arrangements have already been investigated as a means to boost search in constraint optimization problems (Freuder & Quinn, 1985; Modi et al., 2005; Dechter & Mateescu, 2006). Their advantage is that they allow algorithms to exploit the relative independence of nodes lying in different branches of the DFS arrangement

7. For simplicity, we assume in what follows that the original problem is connected. However there is no difficulty in applying DPOP to disconnected problems. The DFS arrangement becomes a *DFS forest*, and agents in each connected component can simply execute DPOP in parallel in a separate control thread. The solution to the overall problem is just the union of optimal solutions for each independent subproblem.

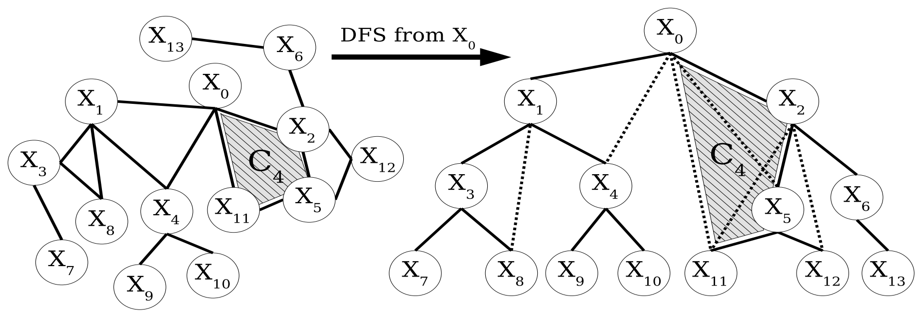


Figure 3: A DFS arrangement for a problem graph. Tree edges are shown in solid and back edges are dashed. The DFS arrangement is constructed by initializing token-passing from X_0. Any k-ary constraints, such as C_4, are treated as if they are cliques.

(i.e. nodes that are not direct descendants or ancestors of one-another), in that it is possible to perform search in parallel on independent branches and then combine the results.

We introduce some definitions related to DFS arrangements:

Definition 3 (DFS concepts) *Given a node X_i in the DFS arrangement, we define:*

• **parent** P_i **/ children** C_i*: X_i's ancestor/descendants connected to X_i via tree-edges (e.g. $P_4 = X_1$, $C_4 = \{X_9, X_{10}\}$).*

• **pseudo-parents** PP_i*: X_i's ancestors connected to X_i via back-edges ($PP_5 = \{X_0\}$).*

• **pseudo-children** PC_i*: X_i's descendants connected to X_i via back-edges (e.g. $PC_1 = \{X_8\}$).*

• **separator** Sep_i *of X_i: ancestors of X_i which are directly connected with X_i or with descendants of X_i (e.g. $Sep_3 = \{X_1\}$ and $Sep_{11} = \{X_0, X_2, X_5\}$).*

• **tree neighbors** TN_i *of X_i are the nodes linked to X_i via tree edges, that is $TN_i = P_i \cup C_i$ (e.g. $TN_4 = \{X_1, X_9, X_{10}\}$).*

Removing the nodes in Sep_i completely disconnects the subtree rooted at X_i from the rest of the problem. In case the problem is a tree, then $Sep_i = \{P_i\}, \forall X_i \in \mathcal{X}$. In the general case, Sep_i contains P_i, all PP_i and all the pseudoparents of descendants of X_i where these pseudoparents are also ancestors of X_i. For example, in Figure 3, the separator of node X_4 contains its parent X_1, and its pseudoparent X_0. It is both necessary and sufficient for the values on variables $\{X_0, X_1\}$ to be set before the problem rooted at node X_4 is independent from the rest of the problem. Separators play an important role in DPOP because contingent solutions must be maintained when propagating utility information up the DFS arrangement for different possible assignments to separator variables.

Constructing the DFS Tree Generating DFS trees in a distributed manner is a task that has received a lot of attention, and there are many algorithms available: for example Collin and Dolev (1994), Barbosa (1996), Cidon (1988), Cheung (1983) to name just a few. For the purposes of executing DPOP, we can assume for example the algorithm of Cheung (1983), which we briefly outline below. When we instantiate DPOP for SCPs, we will present our own adaptation of this DFS generation algorithm to exploit the particulars of SCP.

The simple DFS construction algorithm starts with all agents labeling internally their neighbors as *not-visited*. One of the agents in the graph is designated as the *root*, using for example a leader election algorithm such as that of Abu-Amara (1988),[8] or by simply picking the agent with the lowest ID. The root then initiates the propagation of a *token*, which is a unique message that will be circulated to all the agents in the graph, thus "visiting" them. Initially, the token contains just the ID of the root. The root sends it to one of its neighbors, and waits for its return before sending it to each one of its (still) unvisited neighbors. When an agent X_i first receives the token, it marks the sender as its *parent*. All neighbors of X_i contained in the token are marked as X_i's pseudoparents (PP_i).

After this, X_i adds its own ID to the token, and sends the token *in turn* to each one of its *not-visited* neighbors X_j, which become its *children*. Every time an agent receives the token from one of its neighbors, it marks the sender as *visited*. The token can return either from X_j (the child to whom X_i has sent it in the first place), or from another neighbor, X_k. In the latter case, it means that there is a cycle in the subtree, and X_k is marked as a *pseudochild*.

When a dead end is reached, the last agent backtracks by sending the token back to its parent. When all its neighbors are marked *visited*, X_i has finished exploring all its subtree. X_i then removes its own ID from the token, and sends the token back to its parent; the process is finished for X_i. When the root has marked all its neighbors *visited*, the entire DFS construction process is over.

Handling Non-binary Constraints. No special treatment is required to construct neighbors to a variable that correspond to k-ary constraints, for $k > 2$. For example, in Figure 3 (left), there is a 4-ary constraint C_4 involving $\{X_0, X_2, X_5, X_{11}\}$. By Eq. 2, this implies that $\{X_0, X_2, X_5, X_{11}\}$ are neighbors, and in the DFS construction process and they will appear along the same branch in the tree. This produces the result in Figure 3 (right).

3.1.2 DPOP PHASE TWO: *UTIL* PROPAGATION (INFERENCE)

Phase two is a bottom-to-top pass on the DFS arrangement in which utility information is aggregated and propagated from the leaves towards the root from each node to its parent and through tree edges but not back edges. At a high level, the leaves start by computing and sending *UTIL* messages to their parents, where a *UTIL* message informs the parent about its local utility for solutions to the rest of the problem, minimally specified in terms of its local utility for different value assignments to separator variables. Subsequently each node propagates a *UTIL* message that represents the contingent utility of the subtree rooted at its node for assignments of values to separator variables. In more detail, all nodes perform the following steps:

1. Wait for *UTIL* messages from *all* their children, and store them.

2. Perform an *aggregation*: join messages from children, and also the relations they have with their parents and pseudoparents.

3. Perform an *optimization*: project themselves out of the resulting join by picking their optimal values for each combination of values of the other variables in the join.

4. Send the result to parent as a new *UTIL* message.

8. In cases where the problem is initially disconnected, then it is required to choose multiple roots, one for each connected component. A standard leader election algorithm, when executed by all agents in the problem, will elect exactly as many leaders as there are connected components.

A *UTIL* message sent by a node X_i to its parent P_i is a multidimensional matrix which informs P_i how much utility, $u_i^*(Sep_i)$ the subtree rooted at X_i receives for different assignments of values to variables that define the separator Sep_i for the subtree. One of these variables, by definition, is the variable managed by parent P_i. This *UTIL* message already represents the result of optimization, where variables local to the subtree have been optimized for different assignments of separator variables. To compute a *UTIL* message a node uses two operations: aggregation and optimization. Aggregations apply the JOIN operator and optimizations apply the PROJECT operator as described by Petcu and Faltings (2005b), and briefly summarized here.

Let $UTIL_{i \to j}$ and $UTIL_{k \to j}$ denote *UTIL* messages sent from nodes X_i and X_k to their parent node X_j. We denote by $dim(UTIL_{k \to j})$ the set of *dimensions* of such a matrix, i.e. the set of variables in the separator of sending node X_k. Assuming X_j is the node receiving these messages, we define:

Definition 4 (JOIN operator) *The $\oplus$ operator (join): $UTIL_{i \to j} \oplus UTIL_{k \to j}$ is the join of two UTIL matrices. This is also a matrix with $dim(UTIL_{i \to j}) \cup dim(UTIL_{k \to j})$ as dimensions. The value of each cell in the join is the sum of the corresponding cells in the two source matrices.*

Definition 5 (PROJECT operator) *The $\perp$ operator (projection): if $X_j \in dim(UTIL_{i \to j})$, $UTIL_{i \to j} \perp_{X_j}$ is the projection through optimization of the $UTIL_{i \to j}$ matrix along the X_j axis: for each instantiation of the variables in $\{dim(UTIL_{i \to j}) \setminus X_j\}$, all the corresponding values from $UTIL_{i \to j}$ (one for each value of X_j) are tried, and the maximal one is chosen. The result is a matrix with one less dimension (X_j).*

Notice that the subtree rooted at X_i is influenced by the rest of the problem only through X_i's separator variables. Therefore, a *UTIL* message contains the optimal utility obtained in the subtree for each instantiation of variables Sep_i and the separator size plays a crucial role in bounding the message size.

Example 2 (UTIL propagation) *Figure 4 shows a simple example of a UTIL propagation. The problem has a tree structure (Figure 4(a)), with 3 relations r_3^1, r_2^1, and r_1^0 detailed in Figure 4(b). The relations are between variables $(X_3, X_1), (X_2, X_1)$ and (X_1, X_0) respectively. These are all individual variables and there are no local replicas. In the UTIL phase X_2 and X_3 project themselves out of r_2^1 and r_3^1, respectively. The results are the highlighed cells in r_2^1 and r_3^1 in Figure 4(b). For instance, the optimal value for X_2 given that $X_1 := a$ is to assign $X_2 := c$ and this has utility 5. These projections define the UTIL messages they send to X_1. X_1 receives the messages from X_2 and X_3, and joins them together with its relation with X_0 (adds the utilities from the messages into the corresponding cells of r_1^0). It then projects itself out of this join. For instance, the optimal value for X_1 given $X_0 := b$ is $X_1 := a$ because $2 + 5 + 6 \geq \max\{3 + 4 + 4, 3 + 6 + 3\}$. The result is depicted in Figure 4(d). This is the UTIL message that X_0 receives from X_1. Each value in the message represents the total utility of the entire problem for each value of X_0. We return to this example below in the context of the third phase of value propagation.*

Non-binary Relations and Constraints. As with binary constraints/relations, a k-ary constraint is introduced in the UTIL propagation only once, by the lowest node in the DFS arrangement that is part of the scope of the constraint. For example, in Figure 3, the constraint C_4 is introduced in the UTIL propagation only once, by X_{11}, while computing its message for its parent, X_5.

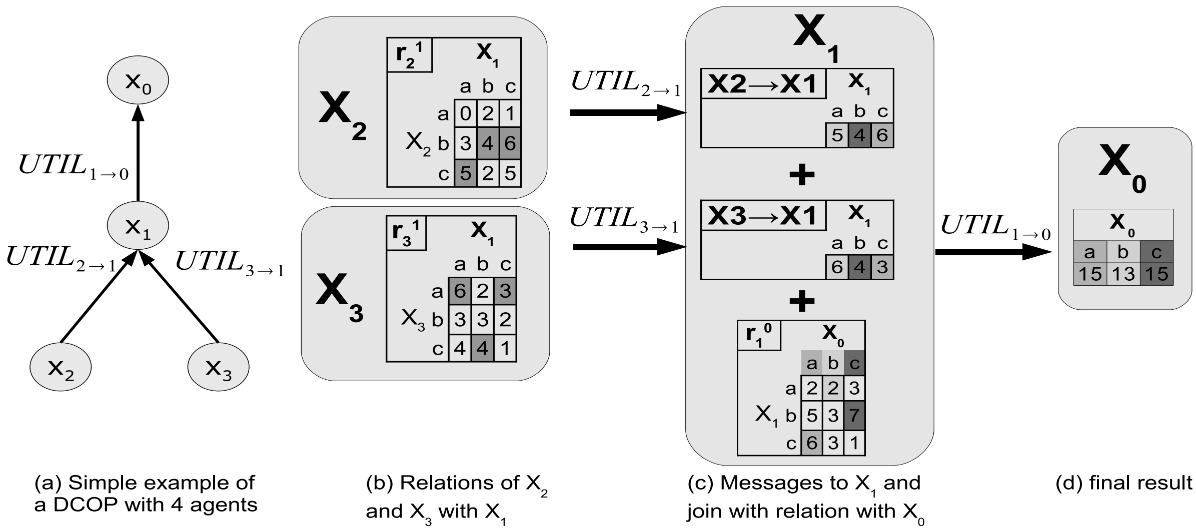


Figure 4: Numerical example of *UTIL* propagation. (a) A simple DCOP problem in which there are three relations r_3^1, r_2^1 and r_1^0 between (X_3, X_1), (X_2, X_1) and (X_1, X_0) respectively. (b) Projections of X_2 and X_3 out of their relations with X_1. The results are sent to X_1 as $UTIL_{2\to1}$, and $UTIL_{3\to1}$ respectively. (c) X_1 joins $UTIL_{2\to1}$ and $UTIL_{3\to1}$ with its own relation with X_0. (d) X_1 projects itself out of the join and sends the result to X_0.

3.1.3 DPOP PHASE THREE: *VALUE* PROPAGATION

Phase three is a top-to-bottom pass that assigns values to variables, with decisions made recursively from the root down to the leaves. This "$VALUE$ propagation" phase is initiated by the root agent X_0 once it has received *UTIL* messages from all of its children. Based on these *UTIL* messages, the root assigns to variable X_0 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 the optimal values to their variables. At the end of this phase, the algorithm finishes, with all variables being assigned their optimal values.

Example 3 (Value propagation) *Return to the example in Figure 4. Once X_0 receives the $UTIL$ message from node X_1 it can simply choose the value for X_0 that produces the largest utility for the whole problem: $X_0 = a$ ($X_0 = a$ and $X_0 = c$ produce the same result in this example, so either one can be chosen). Now in the value-assignment propagation phase X_0 informs X_1 of its choice via a message $VALUE(X_0 \leftarrow a)$. Node X_1 then assigns optimal value $X_1 = c$ and the process continues with a message $VALUE(X_1 \leftarrow c)$ sent to its children, X_2 and X_3. The children assign $X_2 = b$ and $X_3 = a$ and the algorithm terminates with an optimal solution $\langle X_0 = a, X_1 = c, X_2 = b, X_3 = a\rangle$ and total utility of 15.*

3.1.4 COMPLEXITY ANALYSIS OF DPOP

DPOP produces a number of messages that scales linearly in the size of the problem graph, i.e. linearly in the number of nodes and edges in the DCOP model (Petcu & Faltings, 2005b). The complexity of DPOP lies in the size of the *UTIL* messages (note that the tokens passed around in constructing the *DFS*$(\mathcal{A})$ and the *VALUE* messages are of size linear in the problem graph). Petcu and Faltings (2005b) show that the size of the largest *UTIL* message is exponential in a parameter called the *induced width* (Kloks, 1994; Dechter, 2003).

The induced width, denoted w, of a constraint graph given by a chosen DFS arrangement is a structural parameter that equals the size of the largest separator of any node in the DFS arrangement (see Definition 3.):

$$w = \max_{X_i \in \mathcal{X}} |Sep_i|. \tag{6}$$

In the example from Figure 3, the induced width of the graph given this particular DFS ordering is $w = 3$, given by $Sep_{11} = \{X_0, X_2, X_5\}$. Intuitively, the more a problem has a tree-like structure, the lower its induced width. In particular, if the problem graph is a tree then it will have an induced width equal to 1 because the DFS arrangement will always be a tree. Problem graphs that are cliques, on the other hand, have an induced width equal to the number of nodes minus 1, irrespective of the DFS-tree arrangement.

Proposition 1 (DPOP Complexity) *(Petcu & Faltings, 2005b) The number of messages passed in DPOP is $2m$, $(n-1)$ and $(n-1)$ for phases one, two and three respectively, where n and m are the number of nodes and edges in the DCOP model with replicated variables. The maximal number of utility values computed by any node in DPOP is $O(D^{w+1})$, and the largest UTIL message has $O(D^w)$ entries, where w is the induced width of the DFS ordering used.*

In the case of trees, DPOP generates *UTIL* messages of dimension equal to the domain size of the variable defining the parent of each node. In the case of cliques, the maximal message size in DPOP is exponential in $n-1$. Not all DFS arrangements yield the same width, and it is desirable to construct DFS arrangements that provide low induced width. However, finding the tree arrangement with the lowest induced width is an NP-hard optimization problem (Arnborg, 1985). Nevertheless, good heuristics have been identified for finding tree arrangements with low width (Kloks, 1994; Bayardo & Miranker, 1995; Bidyuk & Dechter, 2004; Petcu & Faltings, 2007, 2005b). Although most were designed and explored in a centralized context, some of them (notably *max-degree* and *maximum cardinality set*) are easily amenable to a distributed environment.

3.2 DPOP Applied to Social Choice Problems

In this section, we instantiate DPOP for efficient social choice problems. Specifically, we first show how the optimization problem is constructed by agents from their preferences and potential variables of interest. Subsequently, we show the changes we make to DPOP to adapt it to the SCP domain. The most prominent such adaptation exploits the fact that several variables represent local replicas of the same variable, and can be treated as such both during the UTIL and the VALUE phases. This adaptation improves efficiency significantly, and allows *complexity claims to be stated in terms of the induced width of the centralized COP problem graph rather than the distributed COP problem graph (see Section 3.2.5).*

3.2.1 Initialization: Community Formation

To initialize the algorithm, each agent first forms the communities around its variables of interest, $X(A_i)$, and defines a local optimization problem $COP_i(X(A_i), R_i)$ with a replicated variable X_v^i

for each $X_v \in X(A_i)$. Shorthand $X_v^i \in COP_i$ denotes that agent A_i has a local replica of variable X_v. Each agent owns multiple nodes and we can conceptualize each node as having an associated "virtual agent" operated by the owning agent. Each such virtual agent is responsible for the associated variable.

All agents subscribe to the communities in which they are interested, and learn which other agents belong to these communities. Neighboring relations are established for each local variable according to Eq. 2, as follows: all agents in a community X_v connect their corresponding local copies of X_v with equality constraints. By doing so, the local problems $COP_i(X(A_i), R_i)$ are connected with each other according to the interests of the owning agents. Local relations in each $COP_i(X(A_i), R_i)$ connect the corresponding local variables. Hard constraints connect local copies of the variables they involve. Thus, the overall problem graph is formed.

For example, consider again Figure 2(b). The decision variables are the start times of the three meetings. Each agent models its local optimization problem by creating local copies of the variables in which it is interested and expressing preferences with local relations. Formally, the initialization process is described in Algorithm 1.

Algorithm 1: *DPOP init: community formation and building $DCOP(\mathcal{A})$.*

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

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_v^i that each replicate a public variable $X_v \in X(A_i)$

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

Each agent A_i connects its local copies $X_v^i \in X(A_i)$ with the corresponding local copies of other agents via equality constraints

3.2.2 DFS TRAVERSAL

The method for DFS traversal is described in Algorithm 2. The algorithm starts by choosing one of the variables, X_0, as the root. This can be done randomly, for example using a distributed algorithm for random number generation, with a leader election algorithm like Ostrovski (1994), or by simply picking the variable with the lowest ID. The agents involved in the community for X_0 then randomly choose one of them, A_r as the *leader*. The local copy X_0^r of variable X_0 becomes the root of the DFS. Making the assumption that virtual agents act on behalf of each variable in the problem, the functioning of the token passing mechanism is similar to that described in Section 3.1.1, with additional consideration given to the community structure. Once a root has been chosen, the agents participate in a *distributed depth-first traversal of the problem graph*. 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. The neighbors of each node are sorted (in line 7) to prioritize for copies of variables held by other agents, and then other local variables, and finally other variables linked through hard constraints.

Example 4 *Consider the meeting scheduling example in Figure 2. Assume that M_3 was chosen as the start community and A_2 was chosen within the community as the leader. A_2 creates an empty token $DFS = \emptyset$ and adds M_3^2's ID to the token ($DFS = \{M_3^2\}$). As in Eq. 2, $Neighbors(M_3^2) = \{M_3^3, M_3^1, M_1^2, M_2^2\}$. A_2 sends the token $DFS = \{M_3^2\}$ to the first unvisited neighbor from this*

Algorithm 2: *DPOP Phase One: DFS construction.*

Inputs: each A_i knows its COP_i, and $Neighbors(X_v^i), \forall X_v^i \in COP_i$
Outputs: each A_i knows $P(X_v^i)$, $PP(X_v^i)$, $C(X_v^i)$, $PC(X_v^i)$, $\forall X_v^i \in COP_i$.

Procedure Initialization
The agents 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 from X_0^r to construct the DFS

Procedure Token Passing (performed by each "virtual agent" $X_v^i \in COP_i$)
if X_v^i *is root* **then** $P(X_v^i) = null$; create empty token $DFS := \emptyset$
else DFS**:=Handle_incoming_tokens()**
Let $DFS := DFS \cup \{X_v^i\}$
Sort $Neighbors(X_v^i)$ by $Siblings(X_v^i)$, then $Local\_neighbors(X_v^i)$, then $Hard\_neighbors(X_v^i)$. Set $C(X_v^i) := null$.
forall $X_l \in Neighbors(X_v^i)$ *s.t.* X_l *not visited yet* **do**
    $C(X_v^i) := C(X_v^i) \cup X_l$. Send DFS to X_l wait for DFS token to return.
Send DFS token back to $P(X_v^i)$.

Procedure Handle_incoming_tokens() //run by each "virtual agent" $X_v^i \in COP_i$
Wait for any incoming DFS message; let X_l be the sender
Mark X_l as *visited*.
if *this is the first DFS message (i.e.* X_l *is my parent)* **then**
    $P(X_v^i) := X_l$; $PP(X_v^i) := \{X_k \neq X_l | X_k \in Neighbors(X_v^i) \cap DFS\}$; $PP(X_v^i) := \emptyset$
else
    if $X_l \notin C(X_v^i)$ *(i.e. this is a DFS coming from a pseudochild)* **then**
        $PC(X_v^i) := PC(X_v^i) \cup X_l$

list, i.e. M_3^3, which belongs to A_3. A_3 receives the token and adds its copy of M_3 (now $DFS = \{M_3^2, M_3^3\}$). A_3 then sends the token to M_3^3's first unvisited neighbor, M_3^1 (which belongs to A_1).

Agent A_1 receives the token and adds its own copy of M_3 to it (now $DFS = \{M_3^2, M_3^3, M_3^1\}$). M_3^1's neighbor list is $Neighbors(M_3^1) = \{M_3^2, M_3^3, M_1^1\}$. Since the token that A_1 has received already contains M_3^2 and M_3^3, this means that they were already visited. Thus, the next variable to visit is M_1^1, which happens to be a variable that also belongs to A_1. The token is "passed" to M_1^1 internally (no message exchange required), and M_1^1 is added to the token (now $DFS = \{M_3^2, M_3^3, M_3^1, M_1^1\}$).

The process continues, exploring sibling variables from each community in turn, and then passing on to another community, and so on. Eventually 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. 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 and so on. Eventually, the token returns on the same path all the way to the root and the process completes.

3.2.3 HANDLING THE PUBLIC HARD CONSTRAINTS.

Social choice problems, as defined in Definition 1 can contain side constraints, in the form of publicly known hard constraints, that represent domain knowledge such as "a resource can be allocated only once", "this hotel can accomodate 100 people", "no person can be in more than one meeting at the same time." etc. These constraints are not owned by any agent, but are available to all agents interested in any variable involved in the domain of any such constraint. Handling these constraints is essentially unchanged from handling the non-binary constraints in standard DPOP, as described in Section 3.1.1 for the DFS construction phase, and in Section 3.1.2 for the UTIL phase. Specifically:

DFS Construction: Neighboring relationships as defined in Eq. 2 require for each local variable that other local copies that share a hard constraint are considered as neighbors. Because of the prioritization in line 7 of Algorithm 2 (for DFS construction), the DFS traversal is mostly made according to the structure defined by the relations of the agents and most hard constraints will appear as backedges in the DFS arrangement of the problem graph.

UTIL Propagation: Hard constraints are introduced in the UTIL propagation phase by the lowest agent in the community of the variable from the scope of the hard constraint, i.e. the agent with the variable that is lowest in the DFS ordering. For example, if there was a constraint between M_2 and M_3 in Figure 2 to specify that M_2 should occur after M_3 then this becomes a backedge between the 2 communities and would be assigned to A_3 for handling.

3.2.4 HANDLING REPLICA VARIABLES

Our distributed model of SCP replicates each decision variable for every interested agent and connects all these copies with equality constraints. By handling replica variables carefully we can avoid increasing the induced width k of the DCOP model when compared to the induced width w of the centralized model. With no further adaptation, the *UTIL* messages in DPOP on the distributed problem graph would be conditioned on as many variables as there are local copies of an original variable. However, all the local copies represent the same variable and must be assigned the same value; thus, sending many combinations where different local copies of the same variable take different values is wasteful. Therefore, we handle multiple replicas of the same variable in *UTIL* propagation as though they are the single, original variable, and condition relations on just this one value. This is realized by updating the JOIN operator as follows:

Definition 6 (Updated JOIN operator for SCP) *Defined in two steps:*

Step 1: Consider all UTIL messages received as in input. For each one, consider each variable X_v^i on which the message is conditioned, and that is also a local copy of an original variable X_v. Rename X_v^i from the input UTIL message as X_v, i.e. the corresponding name from the original problem.

Step 2: Apply the normal JOIN operator for DPOP.

Applying the updated JOIN operator makes all local copies of the same variable become indistinguishable from each other, and merges them into a single dimension in the *UTIL* message and avoids this exponential blow-up.

Example 5 *Consider the meeting scheduling example in Figure 2. The centralized model in Figure 2(a) has a DFS arrangement that yields induced width 2 because it is a clique with 3 nodes.*

Nevertheless, the corresponding DCOP model in Figure 2(b) has induced width 3, as can be seen in the DFS arrangement from Figure 2(c), in which $Sep_{M_2^2} = \{M_3^2, M_3^3, M_1^2\}$. Applying DPOP to this DFS arrangement, M_2^2 would condition its UTIL message $UTIL_{M_2^2 \to M_1^2}$ on all variables in its separator: $\{M_3^2, M_3^3, M_1^2\}$. However, both M_3^2 and M_3^3 represent the same variable, M_3. Therefore, M_2^2 can apply the updated JOIN operator, which leverages the equality constraint between the two local replicas and collapse them into a single dimension (called M_3) in its message for M_1^2. The result it that the outgoing message only has 2 dimensions: $\{M_3, M_1^2\}$, and it takes much less space. This is possible because all 3 agents involved, i.e. A_1, A_2 and A_3 know that M_3^1, M_3^2 and M_3^3 represent the same variable.

With this change, the VALUE propagation phase is modified so that only the top most local copy of any variable solve an optimization problem and compute the best value, announcing this result to all the other local copies which then assume the same value.

3.2.5 COMPLEXITY ANALYSIS OF DPOP APPLIED TO SOCIAL CHOICE

By this special handling of replica variables, DPOP applied to SCPs will scale with the induced width of the *centralized* problem graph, and independently of the number of agents involved and in the number of local replica variables.

Consider a DFS arrangement for the centralized model of the SCP that is equivalent to the DFS arrangement for the DCOP model. "Equivalent" here means that the original variables from SCP are visited in the same order in which their corresponding communities are visited during the distributed DFS construction. (Recall that the distributed DFS traversal described in Section 3.1.1 visits all local copies from a community from DCOP before moving on to the next community). Let w denote the induced width of this DFS arrangement of the centralized SCP. Similarly, let k denote the induced width of the DFS arrangement of the distributed model. Let $D = \max_m |d_m|$ denote the maximal domain of any variable. Then, we have the following:

Theorem 1 (DPOP Complexity for SCP) *The number of messages passed in DPOP in solving a SCP is $2m$, $(n-1)$ and $(n-1)$ for phases one, two and three respectively, where n and m are the number of nodes and edges in the DCOP model with replicated variables. The maximal number of utility values computed by any node in DPOP is $O(D^{w+1})$, and the largest UTIL message has $O(D^{w+1})$ entries, where w is the induced width of the* ***centralized*** *problem graph.*

PROOF. The first part of the claim (number of messages) follows trivially from Proposition 1. For the second part (message size and computation): given a DFS arrangement of a DCOP, applying Proposition 1 trivially gives that in the basic DPOP algorithm, the maximal amount of computation on any node is $O(D^{k+1})$, and the largest *UTIL* message has $O(D^k)$ entries, where k is the induced width of the DCOP problem graph. To improve this analysis we need to consider the special handling of the replica variables.

Consider the *UTIL* messages which travel up along the DFS tree, and whose sets of dimensions contain the separators of the sending nodes. Recall that the updated JOIN collapses all local replicas into the original variables. The union of the dimensions of the *UTIL* messages to join in the DPOP on the DCOP model becomes identical to the set of dimensions of the nodes in the DPOP on the centralized model. Thus, each node in the DCOP model performs the same amount of computation as its counterpart on the centralized model. It follows that the *computation* required in DPOP scales as $O(D^{w+1})$ rather than $O(D^{k+1})$ by this special handling.

There remains one additional difference between DPOP on the DFS arrangement for the centralized SCP versus DPOP on the DFS arrangement for the DCOP. A variable X_v that is replicated across multiple agents can only be projected out from the *UTIL* propagation through local optimization by the top-most agent handling a local replica of X_v. This is the first node at which all relevant information is in place to support this optimization step. In particular, whenever a node with the maximal separator set is not also associated with the top-most replica of its variable then it must retain dependence on the value assigned to its variable in the *UTIL* message that it sends to its parent. This increases the worst case *message size* of DPOP to $O(D^{w+1})$, as opposed to $O(D^w)$ for the normal DPOP. Computation remains $O(D^{w+1})$ because the utility has to be determined for each value of X_v anyway, and before projecting X_v out. □

To see the effect on message size described in the proof, in which a local variable cannot be immediately removed during *UTIL* propagation, consider again the problem from Figure 2. Suppose now that agent A_3 is also involved in meeting M_1. This introduces an additional back-edge $M_2^3 - M_1^3$ in the DFS arrangement for the decentralized model shown in Figure 2(c). The DFS arrangement of the COP model that corresponds to the decentralized model is simply a traversal of the COP in the order in which the communities are visited during the distributed DFS construction. This corresponds to a chain: $M_3 - M_1 - M_2$. The introduction of the additional back-edge $M_2^3 - M_1^3$ in the distributed DFS arrangement does not change the DFS of the COP model, and its width remains $w = 2$. However, as M_2^3 is not the top most copy of M_2, agent A_3 cannot project M_2 out of its outgoing *UTIL* message. The result is that it sends a *UTIL* message with $w + 1 = 3$ dimensions, as opposed to just $w = 2$.

4. Handling Self-interest: A Faithful Algorithm for Social Choice

Having adapted DPOP to remain efficient for SCPs, we now turn to the issue of self-interest. Without further modification, an agent can manipulate DPOP by misreporting its private relations and deviating from the algorithm in various ways. In the setting of meeting scheduling, for example, an agent might benefit by *misrepresenting its local preferences* ("I have massively more utility for the meeting occurring at 2pm than at 9am"), *incorrectly propagating utility information of other (competing) agents* ("The other person on my team has very high utility for the meeting at 2pm"), or by *incorrectly propagating value decisions* ("It has already been decided that some other meeting involving the other person on my team will be at 9am so this meeting must be at 2pm.")

By introducing carefully crafted payments, by leveraging the information and communication structure inherent to DCOPs for social choice, and by careful partitioning of computation so that each agent is only asked to reveal information, perform optimization, and send messages that are in its own interest, we are able to achieve *faithfulness*. This will mean that each agent will *choose*, even when self-interested, to follow the modified algorithm. We first define the VCG mechanism for social choice and illustrate its ability to prevent manipulation in centralized problem solving in a simple example. With this in place, we next review the definitions of *faithful distributed implementation* and the results of a useful principle, the *partition principle*. We then describe the *Simple M-DPOP* algorithm – without reuse of computation – and prove its faithfulness.

4.1 Review: Mechanism Design and the VCG Mechanism

Mechanism design (MD) addresses the problem of optimizing some criteria, frequently social welfare, in the presence of self-interested agents that each have private information relevant to the problem at hand. In the standard story, agents report private information to a "center," that solves an optimization problem and enforces the outcome.

The second-price, sealed-bid (Vickrey) auction is a simple example of a mechanism: each agent makes a claim about its value for an item to an auctioneer, who allocates the item to the highest bidder for the second-highest price (Krishna, 2002). The Vickrey auction is useful because it is non-manipulable, in that the weakly dominant strategy of each agent is to report its true value, and efficient, in that the item is allocated to the agent with the highest value.

In our setting of efficient social choice, we will assume the existence of a *currency* so that agents can make payments, and make the standard assumption of *quasilinear* utility functions, so that agent A_i's *net utility* is,

$$u_i(X, p) = R_i(X) - p, \tag{7}$$

for an assignment $X \in \mathcal{D}$ to variables $\mathcal{X}$ and payment $p \in \mathbb{R}$ to the center, i.e., its net utility is that defined by its utility for the assignment, $R_i(X) = \sum_{r_i^j \in R_i} r_i^j(X)$, minus the amount of its payment. One of the most celebrated results of MD is provided by the Vickrey-Clarke-Groves (VCG) mechanism, which generalizes Vickrey's second price auction to the problem of efficient social choice:

Definition 7 (VCG mechanism for Efficient Social Choice) *Given knowledge of public constraints $\mathcal{C}$, and public decision variables $\mathcal{X}$, the Vickrey-Clarke-Groves (VCG) mechanism works as follows:*

- *Each agent, A_i, makes a report $\hat{R}_i$ about its private relations.*
- *The center's decision, X^*, is that which solves $SCP(\mathcal{A})$* given the reports $\hat{R} = (\hat{R}_1, \ldots, \hat{R}_n)$.
- *Each agent A_i, makes payment*

$$Tax(A_i) = \sum_{j \neq i} \left(\hat{R}_j(X^*_{-i}) - \hat{R}_j(X^*) \right), \tag{8}$$

to the center, where X^*_{-i}, for each A_i, is the solution to $SCP(-A_i)$ given reports $\hat{R}_{-i} = (\hat{R}_1, \ldots, \hat{R}_{i-1}, \hat{R}_{i+1}, \ldots, \hat{R}_n)$.

Each agent makes a payment that equals the *negative marginal externality that its presence imposes on the rest of the system*, in terms of the impact of its preferences on the solution to the SCP.

The VCG mechanism has a number of useful properties:

- **Strategyproofness:** Each agent's weakly dominant strategy, i.e. its utility-maximizing strategy whatever the strategies and whatever the private information of other agents, is to truthfully report its preferences to the center. This is the sense in which the VCG mechanism is non-manipulable.

- **Efficiency:** In equilibrium, the mechanism makes a decision that maximizes the total utility to agents over all feasible solutions to the SCP.

- **Participation:** In equilibrium, the utility to agent A_i, $R_i(X^*) - Tax(A_i) = (R_i(X^*) + \sum_{j \neq i} \hat{R}_j(X^*)) - \sum_{j \neq i} \hat{R}_j(X^*_{-i})$, is non-negative, by the principle of optimality, and therefore agents will choose to participate.

- **No-Deficit:** The payment made by agent A_i is non-negative in the SCP, because $\sum_{j \neq i} \hat{R}_j(X^*_{-i}) \geq \sum_{j \neq i} \hat{R}_j(X^*)$ by the principle of optimality, and therefore the entire mechanism runs at a budget surplus.

To begin to understand why the VCG mechanism is strategyproof, notice that the first term in $Tax(A_i)$ is independent of A_i's report. The second term, when taken together with the agent's own *true* utility from the decision, provides A_i with net utility $R_i(X^*) + \sum_{j \neq i} \hat{R}_j(X^*)$. This is the total utility for all agents, and to maximize this the agent should simply report its true preference information, because the center will then explicitly solve this problem in picking X^*.

Example 6 *Return to the example in Figure 4. We can make this into a SCP by associating agents A_1, A_2 and A_3 with relations r_1^0, r_2^1 and r_3^1 on variables $\{X_0, X_1\}$, $\{X_1, X_2\}$, and $\{X_1, X_3\}$ respectively. Breaking ties as before, the solution to $SCP(\mathcal{A})$ is $< X_0 = a, X_1 = c, X_2 = b, X_3 = a >$ with utility $< 6, 6, 3 >$ to agents A_1, A_2 and A_3 respectively. Removing agent A_1, the solution would be $< X_0 =?, X_1 = a, X_2 = c, X_3 = a >$ with utility $< 5, 6 >$ to agents A_2 and A_3. The '?' indicates that agents A_2 and A_3 are indifferent to the value on X_0. Removing agent A_2, the solution would be $< X_0 = c, X_1 = b, X_2 =?, X_3 = c >$, with utility $< 7, 4 >$ to agents A_1 and A_3. Removing agent A_3, the solution would be $< X_0 = a, X_1 = c, X_2 = b, X_3 =? >$, with utility $< 6, 6 >$ to agents A_1 and A_2. The VCG mechanism would assign $< X_0 = a, X_1 = c, X_2 = b, X_3 = a >$, with payments $(5+6)-(6+3) = 2, (7+4)-(6+3) = 2, (6+6)-(6+6) = 0$ collected from agents A_1, A_2 and A_3 respectively. A_3 has no negative impact on agents A_1 and A_2 and does not incur a payment. The other agents make payments: the presence of A_1 helps A_2 but hurts A_3 by more, while the presence of A_2 hurts both A_1 and A_3. The only conflict in this problem is about the value assigned to variable X_1. Agents A_1, A_2 and A_3 each prefer that X_1 be assigned to b, c and a respectively. In the chosen solution, only agent A_2 gets its best outcome. Considering the case of A_3, it can force either a or b to be selected by reporting a suitably high utility for this choice, but for $X_1 = a$ it must pay 4 while for $X_1 = b$ it must pay 1, and in either case it weakly prefers the current outcome in which it makes zero payment.*

Having introduced the VCG mechanism, it is important to realize that the VCG mechanism provides the *only* known, general purpose, method that exists to solve optimization problems in the presence of self-interest and private information. On the positive side, it is straightforward to extend the VCG mechanism (and the techniques of our paper) to maximize a *linear weighted sum* of the utility of each agent, where these weights are fixed and known, for instance by a social planner (Jackson, 2000). Roberts (1979) on the other hand, established that the Groves mechanisms – of which the VCG mechanism is the most important special case – are the only non-trivial strategyproof mechanisms in the domain of social choice unless there is some known structure to agent preferences; e.g., everyone prefers earlier meetings, or more of a resource is always weakly preferred to less. Together with another technical assumption, Robert's theorem has also been extended by Lavi et al. (2003) to domains with this kind of structure, for instance to combinatorial auctions. *We see that there is a very real sense in which it is only possible to address self-interested in DCOPs when maximizing something like the total utility of participants.*

4.2 Faithful Distributed Implementation

Our goal in faithful distributed implementation is to distribute the computation required to solve the SCP and determine payments to the population of agents, but to do this while retaining an analog to strategyproofness. This can be challenging because it opens up additional opportunities for manipulation beyond those in the centralized VCG mechanism.

In presenting our results, we introduce the following additional assumptions over-and-above those made so far:

- Agents are *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 (given the behavior of other agents).
- Each agent is prevented from posing as several independent agents by an external technique (perhaps cryptographic) for providing strong (perhaps pseudonymous) identities.
- *Catastrophic failure* will occur if all agents in the community of a variable do not eventually choose the same value for the variable.
- There is a *trusted bank*, connected with a *trusted communication channel* to each agent, and with the authority to collect payments from each agent.

The property of "rational but helpful" is required in being able to rely upon agents to compute the payments that other agents should make. Strong identities is required to avoid known vulnerabilities of the VCG mechanism as shown by Yokoo, Sakurai and Matsubara (2004), wherein agents can sometimes do better by participating under multiple identities. Catastrophic failure ensures that the decision determined by the protocol is actually executed. It prevents a "hold-out" problem, where an unhappy agent refuses to adopt the consensus decision. An alternative solution would be to have agents report the final decision to a trusted party, responsible for enforcement. By a "trusted communication channel", we mean that each agent can send a message to the bank without interference by any other agent. These messages are only sent by an agent upon termination of M-DPOP, to inform the bank about other agents' payments. The bank is also assumed in other work on distributed MD (Feigenbaum et al., 2002, 2006; Shneidman & Parkes, 2004), and is the only trusted entity that we require. Its purpose is to ensure that payments can be used to align incentives.

To provide a formal definition of a distributed implementation we need the concept of a *local state*. The local state of an agent A_i corresponds to the sequence of messages that the agent has received and sent, together with the initial information available to an agent (including both its own relations, and public information such as constraints). Given this, a distributed implementation, $d_M =< g, \Sigma, \breve{s} >$, is defined in terms of three components (Shneidman & Parkes, 2004; Parkes & Shneidman, 2004):

- *Strategy space*, Σ, which defines the set of feasible strategies $\sigma_i \in \Sigma$ available to agent A_i, where strategy σ_i defines the message(s) that agent A_i will send in every possible local state.
- *Suggested protocol*, $\breve{s} = (\breve{s}_1, \ldots, \breve{s}_n)$, which defines a strategy that is parameterized by the private relations R_i of agent A_i.

- *Outcome rule*, $g = (g_1, g_2)$, where $g_1 : \Sigma^n \to \mathcal{D}$ defines the assignment of values, $g_1(\sigma) \in \mathcal{D}$, to variables $\mathcal{X}$ given a *joint strategy*, $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma^n$, and $g_2 : \Sigma^n \to \mathbb{R}^n$ defines the payment $g_{2,i}(\sigma) \in \mathbb{R}$ made by each agent A_i given joint strategy $\sigma \in \Sigma^n$.

By defining the message(s) that are sent in every state, a strategy $\sigma_i \in \Sigma$ encompasses all computation performed internally to an agent, all information that an agent reveals about its private inputs (e.g. its relations), and all decisions that an agent makes about how to propagate information received as messages from other agents.[9] The suggested protocol $\breve{s}_i$ corresponds to an algorithm, which takes as input the private information available to an agent and relevant details about the agent's local state, and generates a message or messages to send to neighbors in the network. When applied to distributed input $R = (R_1, \ldots, R_n)$ and the known parts of the input such as hard constraints $\mathcal{C}$, the protocol $\breve{s}$ induces a particular execution trace of the algorithm. This in turn induces the outcome $g(\sigma)$, for $\sigma = \breve{s}(R)$, where $g_1(\sigma)$ is the final assignment of values (information about which is distributed across agents) and $g_2(\sigma)$ is the vector of payments that the bank will collect from agents.[10]

The main question that we ask, given a distributed algorithm and its corresponding suggested protocol, is whether the suggested protocol forms an *ex post* Nash equilibrium of the induced game:

Definition 8 (Ex post Nash equilibrium.) *Given distributed implementation $d_M =< g, \Sigma, \breve{s} >$, the suggested protocol $\breve{s} = (\breve{s}_1, \ldots, \breve{s}_n)$ is an an ex post Nash equilibrium (EPNE) if, for all agents A_i, all relations R_i, all relations of other agents R_{-i}, and all alternate strategies $\sigma'_i \in \Sigma$,*

$$R_i(g_1(\breve{s}_i(R_i), \breve{s}_{-i}(R_{-i}))) - g_2(\breve{s}_i(R_i), \breve{s}_{-i}(R_{-i})) \geq R_i(g_1(\sigma'_i, \breve{s}_{-i}(R_{-i}))) - g_2(\sigma'_i, \breve{s}_{-i}(R_{-i})) \tag{9}$$

In an EPNE, no agent A_i can benefit by deviating from protocol, s_i, whatever the particular instance of DCOP (i.e. for all private relations $R = (R_1, \ldots, R_n)$), *so long as the other agents also choose to follow the protocol*. It is this latter requirement that makes EPNE weaker than dominant-strategy equilibrium, in which s_i would be the best protocol for agent i *even if* the other agents followed an arbitrary protocol.

Definition 9 (Faithfulness) *Distributed implementation, $d_M = < g, \Sigma, \breve{s} >$, is ex post faithful if suggested protocol $\breve{s}$ is an ex post Nash equilibrium.*

That is, when a suggested protocol, $\breve{s}$, is said to be *ex post* faithful (or simply "faithful") then it is in the best interest of every agent A_i to follow all aspects of the algorithm – information revelation, computation and message-passing – whatever the private inputs of the other agents, as long as every other agent follows the algorithm.

9. The idea that each agent only has a limited set of possible messages that can be sent in a local state – as implied by the notion of a (restricted) strategy space Σ – is justified in the following sense. Agents in the model are autonomous and self-interested and, of course, free to send any message in any state. But on the other hand, and if the suggested protocol is followed by every other agent, then only some messages will be semantically meaningful to the recipient agent(s) and trigger a meaningful change in local state in the recipient agent(s); i.e. a change in local state that will changes the future (external) behavior of the recipient agent. In this way, the strategy space characterizes the complete set of "interesting" behaviors available to an agent *given that the other agents follow the suggested protocol.* This is sufficient, from a technical perspecitve, to define an *ex post* Nash equilibrium.
10. The outcome rule must be well-defined for any unilateral deviation from $\breve{s}$, i.e. where any one agent deviates and does not follow the suggested protocol. Either the protocol still reaches a terminal state so that decisions and payments are defined, or the protocol reaches some "bad" state with suitably negative utility to all participants, such as livelock or deadlock. We neglect this latter possibility for the rest of our analysis, but it can be easily treated by introducing special notation for this bad outcome.

4.3 The Partition Principle Applied to Efficient Social Choice

One cannot achieve a faithful DI for efficient SCP by simply running DPOP, $n+1$ times on the same problem graph, once for the main problem and then with each agent's effect nullified in turn by asking it to simply propagate messages. Agent A_i would seek to do the following: (a) interfere with the computational process for $SCP(-A_i)$, to make the solution as close as possible to that to $SCP(\mathcal{A})$, so that its marginal impact appears small; and (b) otherwise decrease its payment, for example by increasing the apparent utility of other agents for the solution to $SCP(\mathcal{A})$, and in turn increases the value of the second term in its VCG payment (Eq. 8).

This opportunity for manipulation was recognized by Parkes and Shneidman (2004) in a more general setting, who proposed the *partition principle* as a method for achieving faithfulness in distributed VCG mechanisms, instantiated here in the context of efficient SCPs:

Definition 10 (partition principle) *A distributed algorithm, corresponding to suggested protocol $\breve{s}$, satisfies the partition principle in application to efficient social choice, if:*

1. **(Correctness)** *An optimal solution is obtained for $SCP(\mathcal{A})$ and $SCP(-A_i)$ when every agent follows $\breve{s}$, and the bank receives messages that instruct it to collect the correct VCG payment from every agent.*

2. **(Robustness)** *Agent A_i cannot influence the solution to $SCP(-A_i)$, or the report(s) that the bank receives about the negative externality that A_i imposes on the rest of the system conditioned on solutions to $SCP(\mathcal{A})$ and $SCP(-A_i)$.*

3. **(Enforcement)** *The decision that corresponds to $SCP(\mathcal{A})$ is enforced, and the bank collects the payments as instructed.*

Theorem 2 *(Parkes & Shneidman, 2004) A distributed algorithm for efficient social choice that satisfies the partition principle is an ex post faithful distributed implementation.*

For some intuition behind this result, note that the opportunity for manipulation by an agent A_i is now restricted to: (a) influencing the solution computed to $SCP(\mathcal{A})$; and (b) influencing the payments made by other agents. Agent A_i cannot prevent the other agents from correctly solving $SCP(-A_i)$ or from correctly reporting the negative externality that A_i imposes on the other agents by its presence. As long as the other agents follow the algorithm, then *ex post* faithfulness follows from the strategyproofness of the VCG mechanism because the additional opportunity for manipulation, over and above that available from misreporting preferences in the centralized context, is to change (either increase or reduce) the amount of some *other* agent's payment. This is opportunity (b). Opportunity (a) is not new. An agent can always influence the solution in the context of a centralized VCG mechanism by misreporting its preferences.

Remark: As has been suggested in previous work, the weakening from dominant-strategy equilibrium in the centralized VCG mechanism, to *ex post* Nash equilibrium in a distributed implementation, can be viewed as the "cost of decentralization". The incentive properties necessarily rely on the payments that are collected which rely in turn on the computation performed by other agents and in turn on the strategy followed by other agents.[11]

11. An exception is provided by Izmalkov et al. (2005), who are able to avoid this through the use of cryptographic primitives, in their case best thought of as physical devices such as ballot boxes.

4.4 Simple M-DPOP

Algorithm 3 describes simple-M-DPOP. In this variation the main problem, $SCP(\mathcal{A})$ is solved, followed by the social choice problem, $SCP(-A_i)$ with each agent removed in turn.[12] Once these $n+1$ problems are solved, every agent A_j knows the *local part* of the solution to X^* and X^*_{-i} for all $A_i \neq A_j$, which is the part of the solution that affects its own utility. This provides enough information to allow the system of agents without some agent A_i, for any A_i, to each send a message to the bank about a *component* of the payment that agent A_i should make.

Algorithm 3: *Simple-M-DPOP.*

Run DPOP for $DCOP(\mathcal{A})$ on $DFS(\mathcal{A})$; find X^*
forall $A_i \in \mathcal{A}$ **do**
    Build $DFS(-A_i)$; run DPOP for $DCOP(-A_i)$ on $DFS(-A_i)$; find X^*_{-i}
    All agents $A_j \neq A_i$ compute $Tax_j(A_i) = R_j(X^*_{-i}) - R_j(X^*)$ and report to bank.
    Bank deducts $\sum_{j \neq i} Tax_j(A_i)$ from A_i's account
Each A_i assigns values in X^* as the solution to its local COP_i

The computation of payments is disaggregated across the agents. The tax payment collected from agent A_i is $Tax(A_i) = \sum_{j \neq i} Tax_j(A_i)$, where

$$Tax_j(A_i) = R_j(X^*_{-i}) - R_j(X^*), \tag{10}$$

is the component of the payment that occurs because of the negative effect that agent A_i has on the utility of agent A_j. This information is communicated to the bank by agent A_j in the equilibrium.

The important observation, in being able to satisfy the partition principle, is that these components of A_i's payment satisfy a **locality property**, so that *each agent A_j can compute this component of A_i's payment with just its private information about its relations and its local information about the parts of solutions X^* and X^*_{-i} that affect its own utility*. All of this information is available upon termination of simple-M-DPOP. Correctly determining this payment, once we condition on solutions X^* and X^*_{-i}, does not rely on any aspect of any other agent's algorithm, including that of A_i.[13]

Figure 5 provides an illustration of Simple M-DPOP on the earlier meeting scheduling example, and shows how the marginal problems (and the DFS arrangements for each such problem) are related to the main problem.

Theorem 3 *The simple-M-DPOP algorithm is a faithful distributed implementation of efficient social choice and terminates with the outcome of the VCG mechanism.*

PROOF. To prove this we establish that simple-M-DPOP satisfies the partition principle and then by appeal to Theorem 2. First, DPOP computes optimal solutions to $SCP(\mathcal{A})$ and $SCP(-A_i)$ for

12. Simple M-DPOP is presented for a setting in which the main problem and the subproblems are connected but extends immediately to disconnected problems. Indeed, it may be that the main problem is connected but one or more subproblems are disconnected. To see that there are no additional incentive concerns notice that it is sufficient to recognize that the correctness and robustness properties of the partition principle would be retained in this case.

13. A similar disaggregation was identified by Feigenbaum et al. (2002) for lowest-cost interdomain routing on the Internet. Shneidman and Parkes (2004) subsequently modified the protocol by those authors so that agents other than A_i had enough information to report the payments to be made by agent A_i.

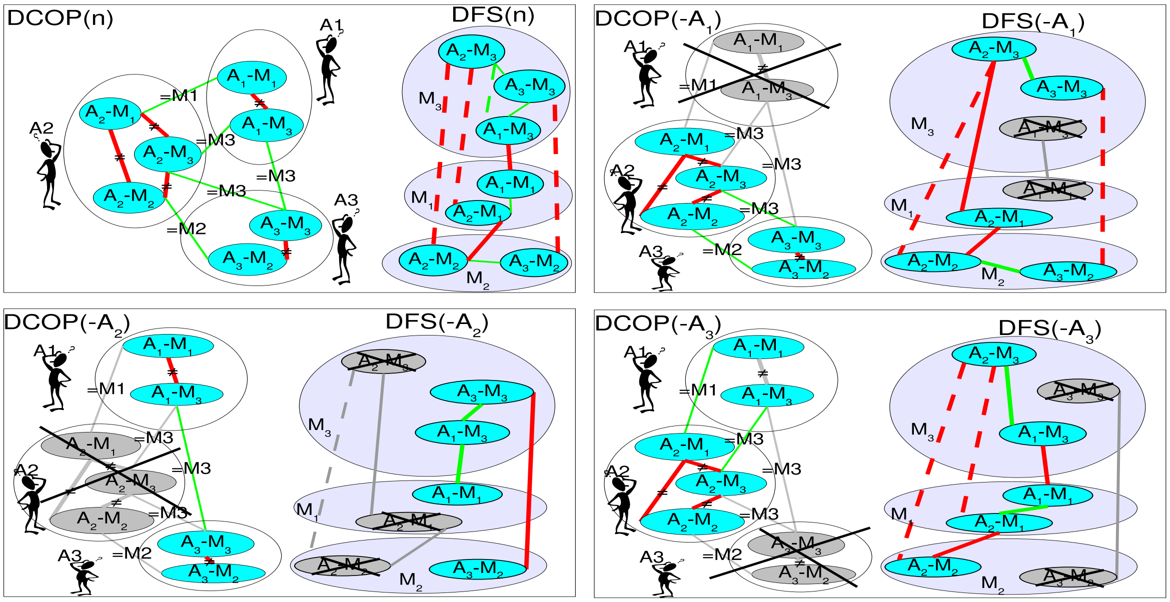


Figure 5: Simple M-DPOP: Each agent A_i is excluded in turn from the optimization $DCOP(-A_i)$. This is illustrated on the meeting scheduling example.

all $A_i \in \mathcal{A}$ when every agent follows the protocol. This is immediate because of the correctness of the DCOP model of SCP and the correctness of DPOP. The correct VCG payments are collected when every agent follows the algorithm by the correctness of the disaggregation of VCG payments in Eq. 10. Second, agent A_i cannot influence the solution to $SCP(-A_i)$ because it is not involved in that computation in any way. The DFS arrangement is constructed, and the problem solved, by the other agents, who completely ignore A_i and any messages that agent A_i might send. (Any hard constraints that A_i may have handled in $SCP(\mathcal{A})$ are reassigned automatically to some other agent in $SCP(-A_i)$ as a consequence of the fact that the DFS arrangement is reconstructed). DPOP still solves $SCP(-A_i)$ correctly in the case that the problem graph corresponding to $SCP(-A_i)$ becomes disconnected (in this case the DFS arrangement is a forest). The robustness of the value of the reports from agents $\neq A_i$ about the negative externality imposed by A_i, conditioned on solutions to $SCP(\mathcal{A})$ and $SCP(-A_i)$, follows from the locality property of payment terms $Tax_j(A_i)$ for all $A_j \neq A_i$. For enforcement, the bank is trusted and empowered to collect payments, and all agents will finally set local copies of variables as in X^* to prevent catastrophic failure. 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. $\square$

The partition principle, and faithfulness, has sweeping implications. Not only will each agent follow the subtantive aspects of simple-M-DPOP, *but each agent will also choose to faithfully participate in the community discovery phase, in any algorithm for choosing a root community, and in selecting a leader agent in Phase one of DPOP.*[14]

14. One can also observe that is not useful for an agent to misreport the local utility of *another agent* A_j while sending $UTIL$ messages around the system. On one hand, such a deviation could of course change the selection of X^* or X^*_{-k} for some $k \neq \{i, j\}$ and thus the payments by other agents or the solution ultimately selected. But, by deviating

Remark on Antisocial Behavior: Note that 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.

5. M-DPOP: Reusing Computation While Retaining Faithfulness

In this section, we present our main result, which is the M-DPOP algorithm. In *simple-M-DPOP*, the computation to solve the main problem is completely isolated from the computation to solve each of the marginal problems. In comparison, in *M-DPOP* we re-use computation already performed in solving the main problem in solving the marginal problems. This enables the algorithm to scale well to problems where each agent's influence is limited to a small part of the entire problem because little additional computation is required beyond that of DPOP. These problems in which an agent's influence is limited are precisely those of interest because they are also those for which the induced tree width is small and for which DPOP scales.

The challenge that we face, in facilitating this re-use of computation, is to retain the incentive properties that are provided by the partition principle. *A possible new manipulation is for agent A_i to deviate in the computation in $DCOP(\mathcal{A})$, with the intended effect to change the solution to $DCOP(-A_i)$ via the indirect impact of the computation performed in $DCOP(\mathcal{A})$ when it is reused in solving $DCOP(-A_i)$. To prevent this, we have to determine which UTIL messages in $DCOP(\mathcal{A})$ could not have been influenced by agent A_i.*

Example 7 *Refer to Figure 6. Here agent A_i controls only X_3 and X_{10}. Then it has no way of influencing the messages sent in the subtrees rooted at $\{X_{14}, X_{15}, X_2, X_7, X_5, X_{11}\}$. We want to be able to reuse as many of these UTIL messages as possible. In solving the problem with agent A_i removed we will strive to construct a DFS^{-i} arrangement for problem $DCOP(-A_i)$ that is as similar as possible to the DFS for the main problem. This is done with the goal of maximizing the re-use of computation across problems. See Figure 6(b). Notice that this is now a DFS forest, with three distinct connected components. The UTIL messages that were sent by the shaded nodes can be re-used in solving $DCOP(-A_i)$. These are all the UTIL messages sent by nodes in the subtrees that were not influenced by agent A_i except for $\{X_{14}, X_{15}, X_5\}$ and also X_9, which now has a different local DFS arrangement.*

M-DPOP uses the "safe reusability" idea suggested by this example. See Algorithm 4. In its first stage, M-DPOP solves the main problem just as in Simple-M-DPOP. Once this is complete, each marginal problem $DCOP(-A_i)$ is solved in parallel. To solve $DCOP(-A_i)$, a DFS^{-i} forest (it will be a forest in the case that $DCOP(-A_i)$ becomes disconnected) is constructed as a modification to $DFS(\mathcal{A})$, retaining as much of the structure of $DFS(\mathcal{A})$ as possible. A new $DPOP(-A_i)$ execution is performed on the DFS^{-i} and $UTIL$ messages are determined to be either *reusable* or *not reusable* by the sender of the message based on the differences between DFS^{-i} and $DFS(\mathcal{A})$. We will explain below how DFS^{-i} is constructed.

in this way the agent cannot change the utility information that is finally used in determining its own payments. This is because it is agent A_j *itself* that computes the marginal effect of agent A_i on its local solution, and component $Tax_j(A_i)$ of agent A_i's payment.

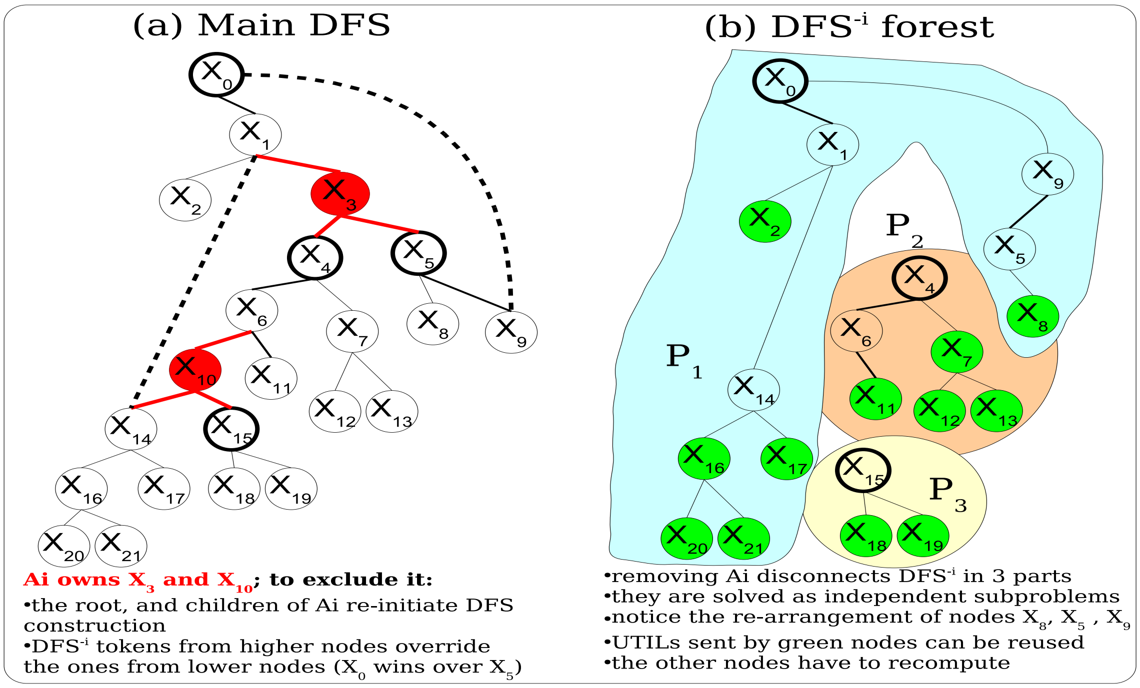


Figure 6: Reconstructing $DFS(-A_i)$ from $DFS(\mathcal{A})$ in M-DPOP. The result is in general a DFS forest. The bold nodes from main DFS initiate DFS^{-i} propagation. The one initiated by X_5 is redundant and eventually stopped by X_9. The ones from X_4 and X_{15} are useful, as their subtrees become really disconnected after removing A_i. X_{14} does not initiate any propagation since it has X_1 as a pseudoparent. X_1 is not controlled by A_i, and will eventually connect to X_{14}. Notice that $X_0 - X_9$ and $X_1 - X_{14}$ are turned into tree edges.

5.1 Phase One of M-DPOP for a Marginal Problem: Constructing DFS^{-i}

Given a graph $DCOP(\mathcal{A})$ and a DFS arrangement $DFS(\mathcal{A})$ of $DCOP(\mathcal{A})$, if one removes a set of nodes $X(A_i) \in DCOP(\mathcal{A})$ (the ones that belong to A_i), then we need an algorithm that constructs a DFS arrangement, DFS^{-i}, for $DCOP(\mathcal{A}) \setminus X(A_i)$. We want to achieve the following properties:

1. DFS^{-i} must represent a correct DFS arrangement for the graph $DCOP(-A_i)$ (a DFS forest in the case $DCOP(-A_i)$ becomes disconnected).

2. DFS^{-i} must be constructed in a way that is non-manipulable by A_i, i.e. without allowing agent A_i to interfere with its construction.

3. DFS^{-i} should be as similar as possible to $DFS(\mathcal{A})$. This allows for reusing *UTIL* messages from $DPOP(\mathcal{A})$, and saves on computation and communication.

The main difficulty stems from the fact that removing the nodes that represent variables of interest to agent A_i from $DFS(\mathcal{A})$ can create disconnected subtrees. We need to reconnect and possibly rearrange the (now disconnected) subtrees of $DFS(\mathcal{A})$ whenever this is possible. Return to the example in Figure 6. Removing agent A_i and nodes X_3 and X_{10} disrupts the tree in two ways: some subtrees become completely disconnected from the rest of the problem (e.g. $X_{15} - X_{18} - X_{19}$); some other ones remain connected only via back-edges, thus forming an invalid DFS arrangement

Algorithm 4: *M-DPOP: faithfully reuses computation from the main problem.*

Run DPOP for $DCOP(\mathcal{A})$ on $DFS(\mathcal{A})$; find X^*
forall $A_i \in \mathcal{A}$ **do**
  in parallel
Create DFS^{-i} with Algorithm 5 by adjusting $DFS(\mathcal{A})$
Run DPOP for $DCOP(-A_i)$ **on** DFS^{-i}:
if *leaves in* DFS^{-i} *observe no changes in their* DFS^{-i} **then**
  they send $null\ UTIL^{-i}$ messages
else they compute their $UTIL^{-i}$ messages anew, as in DPOP
subsequently, all nodes $X_k \in DFS^{-i}$ do:
if X_k *receives only* $null\ UTIL^{-i}$ *msgs* $\wedge$ *(*$P_k = P_k^{-i} \wedge PP_k = PP_k^{-i} \wedge C_k = C_k^{-i}$*)* **then**
  X_k sends a $null\ UTIL^{-i}$ message
else
  node X_k computes its $UTIL^{-i}$ message, **reusing**:
  forall $X_l \in Neighbors(X_k)$ *s.t.* X_l *sent* $UTIL^{-i} = null$ **do**
    X_k **reuses** the $UTIL$ message X_l had sent in $DCOP(\mathcal{A})$
Compute and levy taxes as in simple-M-DPOP;
Each A_i assigns values in X^* as the solution to its local COP_i;

(e.g. $X_5 - X_8 - X_9$). The basic principle we use is to reconnect disconnected parts via back-edges from $DFS(\mathcal{A})$ whenever possible. This is intended to preserve as much of the structure of as possible. For example, in Figure 6, the back edge $X_0 - X_9$ is turned into a tree edge, and X_5 becomes X_9's child. Node X_8 remains X_5's child.

The DFS^{-i} reconstruction algorithm is presented in Algorithm 5. The high-level overview is as follows (in bold we state the purpose of each step):

1. (**Similarity to** $DFS(\mathcal{A})$ **:**) All nodes retain the DFS data structures from constructing $DFS(\mathcal{A})$; i.e., the lists of their children, pseudo parents/children, and their parents from $DFS(\mathcal{A})$. They will use this data as a starting point for building the DFS arrangements, $DFS(-A_i)$, for marginal problems.

2. (**At least one traversal of each connected component on a DFS forest:**) The root of $DFS(\mathcal{A})$ *and the children*[15] *of removed nodes* each initiate a DFS^{-i} token passing as in $DFS(\mathcal{A})$, except for these changes:

   - Each node X_k sends the token only to neighbors not owned by A_i.
   - The order in which X_k sends the token to its neighbors is based on $DFS(\mathcal{A})$: First X_k's children from $DFS(\mathcal{A})$, then its pseudochildren, then its pseudoparents, and then its parent. This order helps preserve structure from $DFS(\mathcal{A})$ into $DFS(-A_i)$.

15. Children which have pseudoparents above the excluded node, for instance X_{14} in Figure 6, do not initiate DFS token passing because it would be redundant: they would eventually receive a DFS token from their pseudoparent.

Algorithm 5: *Reconstruction of DFS^{-i} from $DFS(\mathcal{A})$.*
All data structures for the DFS^{-i} are denoted with superscript $^{-i}$.

Procedure Token_passing for DFS^{-i} (executed by all nodes $X_k \notin X(A_i)$) :

forall $X_l \in Neighbors(X_k)$ *s.t.* X_l *belongs to* A_i **do**
  Remove X_l from $Neighbors(X_k)$ and from C_k, PC_k, PP_k *//i.e. send nothing to* A_i
Sort $Neighbors(X_k)$ in this order: C_k, PC_k, PP_k, P_k *//mimic* $DFS(\mathcal{A})$
if X_k *is root, or* $P_k \in X(A_i)$ *(i.e. executed by the root and children of* A_i*)* **then**
  Initiate DFS^{-i} as in normal DFS (Algorithm 2)
else do Process_incoming_tokens()
Send $DFS^{-i}(X_k)$ back to P_k^{-i} *//* X_k*'s subtree completely explored*

Procedure Process_incoming_tokens()

Wait for any incoming DFS^{-i} token; Let X_l be its sender
if $X_l \in A_i$ **then** ignore message
else
  if *this is first token received* **then**
    $P_k^{-i} = X_l$; $PP_k^{-i} = \{X_j \neq P_k^{-i} | X_j \in Neighbors(X_i) \cap DFS^{-i}\}$
    $root_k^{-i}$= first node in the token DFS^{-i}
  else
    let X_r be the first node in DFS^{-i}
    if $X_r \neq root_k^{-i}$ *//i.e. this is another* DFS^{-i} *traversal* **then**
      if *depth of* X_r *in* $DFS(\mathcal{A})$ < *depth of* $root_k^{-i}$ *in* $DFS(\mathcal{A})$ **then**
        Reset $P_k^{-i}, PP_k^{-i}, C_k^{-i}, PC_k^{-i}$ *//override redundant DFS from lower root*
        $P_k^{-i} = X_l$; $PP_k^{-i} = \{X_j \neq P_k^{-i} | X_j \in Neighbors(X_i) \cap DFS^{-i}\}$
        $root_k^{-i} = X_r$
  Continue as in Algorithm 2

3. (**Unique traversal of each connected component on a DFS forest:**) Each node X_k retains its "root path" in $DFS(\mathcal{A})$ and knows its depth in the DFS arrangement. When a new token DFS^{-i} arrives:

   - If it is the first DFS^{-i} token that arrives, then the sender (let this be X_l) is marked as the parent of X_k in DFS^{-i}: $P_k^{-i} = X_l$. Notice that X_l could be different from the parent of X_k from $DFS(\mathcal{A})$. X_k stores the first node from the received token DFS^{-i}, as $root_k^{-i}$: the (provisional) root of the connected component to which X_k belongs in $DCOP(-A_i)$.
   - If this is not the first DFS^{-i} token that arrives, then there are two possibilities:
     – the token received is part of the same DFS^{-i} traversal process. X_k recognizes this by the fact that the first node in the newly received token is the same as the previously stored $root_k^{-i}$. In this case, X_k proceeds as normal, as in Algorithm 2: marks the sender as pseudochild, etc.

– the token received is part of *another* DFS^{-i} traversal process, initiated by another node than $root_k^{-i}$ (see below in text for when this could happen). Let X_r be the first node in the newly received token. X_k recognizes this situation by the fact that X_r is **not** the same as the previously stored $root_k^{-i}$. In this case, the DFS^{-i} traversal initiated by the higher node in $DFS(\mathcal{A})$ prevails, and the other one is dropped. To determine which traversal to pursue and which one to drop, X_k compares the depths of $root_k^{-i}$ and X_r in $DFS(\mathcal{A})$. If X_r is higher, then it becomes the new $root_k^{-i}$. X_k overrides all the previous DFS^{-i} information with the one from the new token. It then continues the token passing with the new token as in Algorithm 2.

To see why it is necessary to also start propagations from the children of removed nodes (step 2), consider again the example from Figure 6. Removing X_{10} and X_3 completely disconnects the subtree $\{X_4, X_6, X_{11}, X_7, X_{12}, X_{13}\}$. Had X_4 not started a propagation, this subtree would not have been visited at all since there are no connections between the rest of the problem and any nodes in the subtree.[16] [17]

Lemma 1 (DFS correctness) *Algorithm 5 constructs a correct DFS arrangement (or forest), DFS^{-i} for $DCOP(-A_i)$ given a correct DFS arrangement $DFS(\mathcal{A})$ for $DCOP(\mathcal{A})$.*

PROOF. First, since a DFS^{-i} is started from each child of a node that was controlled by A_i, and also from the root, it is ensured that each connected component is DFS-traversed at least once (follows from Step 2). Second, each DFS process is similar to a normal DFS construction, in that each node sends the token to all its neighbors (except for the ones controlled by A_i); it is just that they do so in a pre-specified order (the one given by $DFS(\mathcal{A})$). It follows that all nodes in a connected component will eventually be visited (follows from Step 3). Third, higher-priority DFS traversals override the lower priority ones (i.e. DFS traversals initiated by nodes higher in the tree have priority), again by Step 3. Eventually one single DFS-traversal is performed in a single connected component. □

Lemma 2 (DFS robustness) *The DFS arrangement, DFS^{-i}, constructed by Algorithm 5 is non-manipulable by agent A_i, for any input DFS arrangement from the solution phase to $DCOP(\mathcal{A})$.*

PROOF. This follows directly from Step 3, since A_i does not participate in the process at all: its neighbors do not send it any messages (see Algorithm 5, line 1), and any messages it may send are simply ignored (see Algorithm 5, line 7) □

In fact, **no** additional links are created while constructing DFS^{-i}. The only possible changes are that some edges can reverse their direction (parents/children or pseudoparents-pseudochildren

16. Some of the DFS traversals initiated in Step 2 are redundant and the same part of the problem graph can be visited more than once. The simple overriding rule in Step 3 ensures that only a single DFS^{-i} tree is eventually adopted in each connected component, namely the one that is initiated by the *highest node* in the *original* $DFS(\mathcal{A})$. For example, in Figure 6, X_5 starts an unnecessary DFS^{-i} propagation, which is eventually stopped by X_9, which receives a higher priority DFS^{-i} token from X_0. Since X_9 knows that X_0 is higher in $DFS(\mathcal{A})$ than X_5, it drops the propagation initiated by X_5, and relays only the one initiated by X_0. It does so by sending X_5 the token for DFS^{-i} received from X_0 to which it adds itself. Upon receiving the new token from X_9, node X_5 realizes that X_9 is its new parent in DFS^{-i}. Thus, the redundant propagation initiated by X_5 is eliminated and the result is a consistent DFS subtree for the single connected component P_1.
17. A simple time-out mechanism can be used to ensure that each agent knows when its provisional DFS ordering is final (i.e. no higher priority DFS traversals will arrive in the future).

can switch places), and existing back-edges can turn into tree edges. Again, one can see this in Figure 6.[18]

5.2 Phase Two of M-DPOP for a Marginal Problem: $UTIL^{-i}$ propagations

Once DFS^{-i} is built, the marginal problem without A_i is then solved on DFS^{-i}. Utility propagation proceeds as in normal DPOP except that nodes determine whether the *UTIL* message that was sent in $DPOP(\mathcal{A})$ can be reused. This is signaled to their parent by sending a special *null UTIL* message. More specifically, the process is as follows:

- The leaves in DFS^{-i} initiate $UTIL^{-i}$ propagations:

  1. If the leaves in DFS^{-i} observe no changes in their local DFS^{-i} arrangement as compared to $DFS(\mathcal{A})$ then the *UTIL* message they sent in $DCOP(\mathcal{A})$ remains valid and they announce this to their parents by sending instead a $null\ UTIL^{-i}$ message.
  2. Otherwise, a leaf node computes its *UTIL* message anew and sends it to their (new) parent in DFS^{-i}.

- All other nodes wait for incoming $UTIL^{-i}$ messages and:

  1. If *every* incoming messages a node X_k receives from its children is $null$ *and* there are no changes in the parent/pseudoparents then it can propagate a $null\ UTIL^{-i}$ message to its parent.
  2. Otherwise, X_k has to recompute its $UTIL^{-i}$ message. It does so by reusing all the *UTIL* messages that it received in $DCOP(\mathcal{A})$ from children that have sent it $null$ messages in $DCOP(-A_i)$ and joining these with any new *UTIL* messages received.

For example, consider $DCOP(-A_i)$ in Figure 6, where X_{16} and X_{17} are children of X_{14}. X_{14} has to recompute a *UTIL* message and send it to its new parent X_1. To do this, it can reuse the messages sent by X_{16} and X_{17} in $DCOP(\mathcal{A})$, because neither of these sending subtrees contain A_i. By doing so, X_{14} reuses the effort spent in $DCOP(\mathcal{A})$ to compute the messages $UTIL_{20}^{16}$, $UTIL_{21}^{16}$, $UTIL_{16}^{14}$ and $UTIL_{17}^{14}$.

Theorem 4 *The M-DPOP algorithm is a faithful distributed implementation of efficient social choice and terminates with the outcome of the VCG mechanism.*

PROOF. From the partition principle and appeal to Theorem 3 (and in turn to Theorem 2). First, agent A_i cannot prevent the construction of a valid DFS^{-i} for $DCOP(-A_i)$ (Lemmas 1 and 2). Second, agent A_i cannot influence the execution of DPOP on $DCOP(-A_i)$ because all messages that A_i influenced in the main problem $DCOP(\mathcal{A})$ are recomputed by the system without A_i. The rest of the proof follows as for simple-M-DPOP, leveraging the locality of the tax payment messages and the enforcement provided by the bank and via the catastrophic failure assumption. □

18. A simple alternative is to have children of all nodes X_k^i that belong to A_i, create a bypass link to the first ancestor of X_k^i that does not belong to A_i. For example, in Figure 6, X_4 and X_5 could each create a link with X_1 to bypass X_3 completely in $DFS(-A_i)$. However, additional communication links may be required in this approach.

6. Experimental Evaluation: Understanding the Effectiveness of M-DPOP

We present the results of our experimental evaluation of DPOP, Simple M-DPOP and M-DPOP in two different domains: distributed meeting scheduling problems (MS), and combinatorial auctions (CAs). In the first set of experiments we investigate the performance of M-DPOP on a structured constraint optimization problem (MS) which has received a lot of attention in cooperative distributed constraint optimization. In the second set of experiments (CAs), we investigate unstructured domains, and observe the performance – specifically the ability to re-use computation in computing payments – of M-DPOP with respect to problem density. CAs provide an abstract model of many real world allocation problems and are much studied in mechanism design (Cramton, Shoham, & Steinberg, 2006).

6.1 Distributed Meeting Scheduling

In distributed meeting scheduling, we consider a set of agents working for a large organization and representing individuals, or groups of individuals, and engaged in scheduling meetings for some upcoming period of time. Although the agents themselves are self interested, the organization as a whole requires an optimal overall schedule, that minimizes cost (alternatively, maximizes the utility of the agents). This makes it necessary to use a faithful distributed implementation such as M-DPOP. In enabling this, we suppose that the organization distributes a virtual currency to each agent (perhaps using this currency allocation to prioritize particular participants.) All relations held by agents and defining an agent's utility for a solution to the scheduling problem are thus stated in units of this currency.

Each agent A_i has a set of local replicate variables X_j^i for each meeting M_j in which it is involved. The domain of each variable X_j (and thus local replicas X_j^i) represents the feasible time slots for the meeting. An equality constraint is included between replica variables to ensure that meeting times are aligned across agents. Since an agent cannot participate in more than one meeting at once there is an *all-different* constraint on all variables X_i^j belonging to the same agent. This is modeled as a clique constraint between these meeting variables. Each agent assigns a utility to each possible time for each meeting by imposing a unary relation on each variable X_j^i. Each such relation is private to A_i, and denotes how much utility A_i associates with starting meeting M_j at each time $t' \in d_j$, where d_j is the domain for meeting M_j. The social objective is to find a schedule in which the total utility is maximized while satisfying the all-different constraints for each agent.

Following Maheswaran et al. (2004), we model the organization by providing a *hierarchical structure*. In a realistic organization, the majority of interactions are within departments, and only a small number are across departments and even then these interactions will typically take place between two departments adjacent in the hierarchy. This hierarchical organization provides structure to our test instances: with high probability (around 70%) we generate meetings within departments, and with a lower probability (around 30%) we generate meetings between agents belonging to parent-child departments. We generated random problems having this structure, with an increasing number of agents: from 10 to 100 agents. Each agent participates in 1 to 5 meetings, and has a uniform random utility between 0 and 10 for each possible schedule for each meeting in which it participates. The problems are generated such that they have feasible solutions.[19]

19. The test instances can be found at http://liawww.epfl.ch/People/apetcu/research/mdpop/MSexperiments.tgz

For each problem size, we averaged the results over 100 different instances. We solved the main problems using DPOP and the marginal ones using simple-M-DPOP, and M-DPOP respectively. All experiments were performed in the FRODO multiagent simulation environment (Petcu, 2006), on a 1.6Ghz/1GB RAM laptop. FRODO is a simulated multiagent system, where each agent executes asynchronously in its own thread, and communicates with its peers only via message exchange.

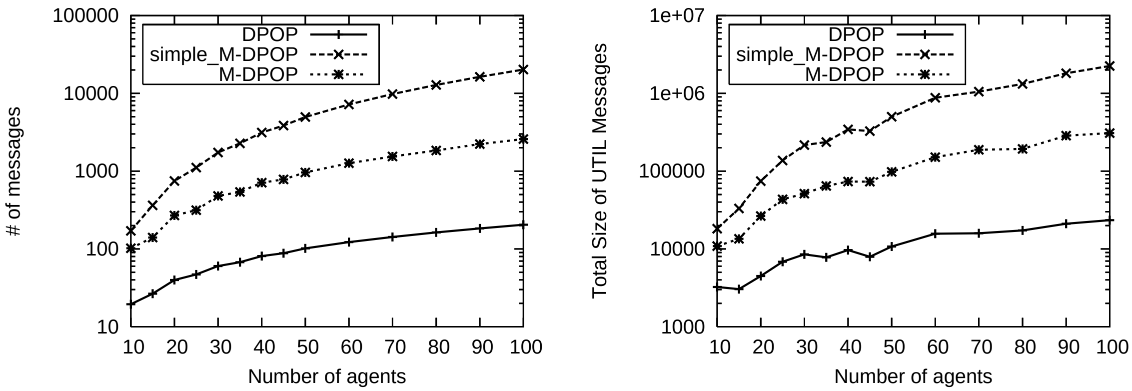
The experiments were geared towards showing how much effort M-DPOP is able to reuse from the main to the marginal problems. Figure 6.1 shows the absolute computational effort in terms of number of messages (Figure 6.1(a)), and in terms of the total size of the messages exchanged, in bytes (Figure 6.1(b)). The curves for DPOP represent just the number of messages (total size of messages, respectively) required for solving the cooperative problem. The curves for simple-M-DPOP and M-DPOP represent the total number (size, respectively) of *UTIL* messages, for both main and marginal economies.

We notice several interesting facts. First, the number of messages required by DPOP increases linearly with the number of agents because DPOP's complexity in terms of number of messages is always linear in the size of the problem. On the other hand, the number of messages of simple-M-DPOP increases roughly quadratically with the number of agents, since it solves a linear number of marginal economies from scratch using DPOP, each requiring a linear number of messages. The performance of M-DPOP lies somewhere between the DPOP and simple-M-DPOP with more advantage achieved over simple-M-DPOP as the size of the problem increases, culminating with almost an order of magnitude improvement over Simple M-DPOP for the largest problem sizes (i.e. with 100 agents in the problem). Similar observations can be made about the total size of the *UTIL* messages, also a good measure of computation, traffic and memory requirements, by inspecting Figure 6.1(b). For both metrics we find that the performance of M-DPOP is only slightly super-linear in the size of the problem.

Figure 8 shows the percentage of the additional effort required for solving the marginal problems that can be reused from the main problem, i.e. the probability that a *UTIL* message required in solving a marginal problem can be taken directly from the message already used in the main problem. We clearly see that as the problem size increases we can actually reuse more and more computation from the main problem. The intuition behind this is that in large problems, each individual agent is localized in a particular area of the problem. This translates into the agent being localized in a specific branch of the tree, thus rendering all computation performed in other branches reusable for the marginal problem that corresponds to that respective agent. Looking also at the percentage of reuse when defined in terms of message size rather than the number of messages we see that this is also trending upwards as the size of the problem increases.

6.2 Combinatorial Auctions

Combinatorial Auctions (CAs) are a popular means to allocate resources to multiple agents. In CAs, bidders can bid on *bundles* of goods (as opposed to bidding on single goods). Combinatorial bids can model both complementarity and substitutability among the goods, i.e. when the valuation for the bundle is more, respectively less than the sum of the valuations for individual items. In our setting the agents are distributed (geographically or logically), and form a problem graph in which neighbors are agents with whom their bids overlap. The objective is to find the feasible solution (i.e. declare bids as winning or losing such that no two winning bids share a good) that maximizes the total utility of the agents.



(a) Number of messages (b) Total size of *UTIL* messages (in valuations)

Figure 7: Meeting scheduling problem: measures of absolute computational effort (in terms of the number of messages sent and the total size of the *UTIL* messages) in DPOP, simple-M-DPOP and M-DPOP. The curves for DPOP represent effort spent just on the main problem, while the ones for simple-M-DPOP and M-DPOP represent total effort over the main and the marginal problems.

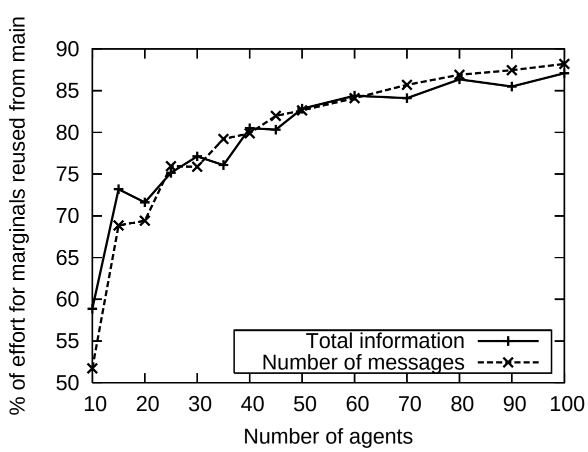


Figure 8: Meeting scheduling problem: Percentage of effort required for the marginal problems that is reused by M-DPOP from the main problem. Reuse is measured both in terms of the percentage of the *UTIL* messages that can be reused (dashed) and also in terms of the total size of the *UTIL* messages that are reused as a fraction of the total *UTIL* message size (solid).

CAs are adopted here as a stylized model of distributed allocation problems such as airport slot allocation and wireless spectrum allocation as discussed in the Introduction. The CA instances also provide a counterpoint to the meeting scheduling problems because they represent problems with less structure. In our DCOP model, each agent holds a variable for each one of its bids, with two possible values: 0 when the bid is rejected, and 1 when the bid is accepted. Any pair of overlapping bids (bids that share at least one good) is connected by a "at most one" constraint that specifies that they cannot be both accepted. When multiple bids are submitted by an agent then they can be connected by additional constraints to capture the bid logic, for instance exclusive-or constraints if only one bid can be accepted.

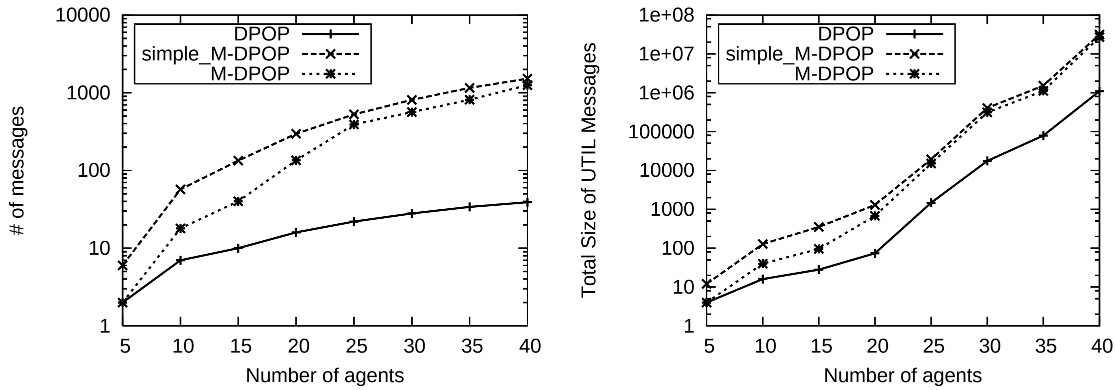
We generated random problems using CATS (Leyton-Brown, Pearson, & Shoham, 2000), using the L3 distribution from Sandholm (2002). L3 is the *Constant* distribution in which each agent demands a bundle of 3 goods, selected uniformly at random, and with a value distributed uniformly on $[0, 1]$. In our simulations we consider a market with 50 goods and vary the number of agents between 5 and 40. We recorded the performance of DPOP, simple-MDPOP and M-DPOP in the graphs from Figures 9 and 10. Figure 9 shows that as the density of the problems increase, all three algorithms require more effort in solving them (both in terms of number of messages, and in terms of total information exchange).

Figure 10 shows how reusability varies with problem density: one can see that for loose problems the reusability is very good, close to 100% for problems with 5 agents. As the density of the problems increases with the number of agents, reusability decreases as well, and is around 20% for the most dense problems, with 40 agents. We explain this phenomenon as follows: for very loose problems (many goods and few bidders), the bids are mostly non-overlapping, which in turn ensures that removing individual agents for solving the marginal problems does not affect the computation performed while solving the main problem. At the other end of the spectrum, very dense problems tend to be highly connected, which produces DFS trees which are very similar to chains. In such a case, removing agents which are close to the bottom of the chain invalidates much of the computation performed while solving the main problem. Therefore, only a limited amount of computation can be reused.

While noting that L3 is recognized as one of the hardest problem distributions in the CATS suite (Leyton-Brown et al., 2000), we remark that we need to limit our experiments to this distribution because other problems have a large induced tree width (and high density problem graphs). Consider for example a problem in which every agent bids for a bundle that overlaps with every other agent. The problem graph is a clique and DPOP does not scale. While we leave a detailed examination for future work, a recent extension of DPOP – H-DPOP (Kumar, Petcu, & Faltings, 2007) – can immediately address this issue. In H-DPOP, consistency techniques are used in order to compactly represent UTIL messages, and on tightly constrained problems, orders of magnitude improvements over DPOP are reported (see Section 7.1).

7. Discussion

In this section we discuss alternatives for improving the computational performance of M-DPOP, the possibility of faithful variations of other DCOP algorithms (ADOPT (Modi et al., 2005) and OptAPO (Mailler & Lesser, 2004)), and the loss in utility for the agents that can occur due to the transfer of payments to the bank, mentioning an approach to address this problem.



(a) Number of messages (b) Total size of *UTIL* messages (in valuations)

Figure 9: Combinatorial Auctions problems: measures of absolute computational effort (in terms of the number of messages sent and the total size of the *UTIL* messages) in DPOP, simple-M-DPOP and M-DPOP. The curves for DPOP represent effort spent just on the main problem, while the ones for simple-M-DPOP and M-DPOP represent effort on both the main and the marginal problems. The higher the number of agents (and thus bids, and thus constraints in the problem graph and problem density), the greater the computational effort to solve the problem.

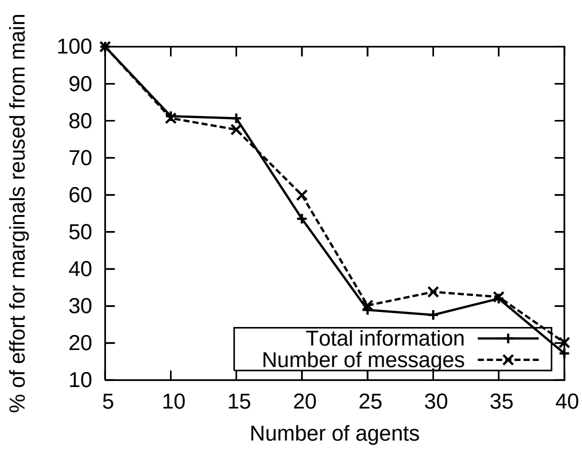


Figure 10: Combinatorial Auctions problems: Percentage of effort required for the marginal problems that is reused by M-DPOP from the main problem. Reuse is measured both in terms of the percentage of the *UTIL* messages that can be reused (dashed) and also in terms of the total size of the *UTIL* messages that are reused as a fraction of the total *UTIL* message size (solid).

7.1 Algorithmic Alternatives for Improved Performance

M-DPOP scales very well with problem size as long as the induced width of the problem remains low. This is a characteristic M-DPOP inherits from DPOP, on which it is based. For problems with high induced width, DPOP/M-DPOP require producing, sending and storing large messages, which may be unfeasible or undesirable. To mitigate this problem, several advances to the basic DPOP algorithm have been recently proposed. Some of these new algorithms sacrifice optimality in return for computational tractability, which makes them difficult to combine with a VCG payment mechanism in such a way that faithfulness be guaranteed. Nevertheless, H-DPOP (Kumar et al., 2007) and MB-DPOP (Petcu & Faltings, 2007) employ two different techniques that preserve the optimality guarantees, and can be fitted to M-DPOP.

H-DPOP leverages the observation that many real problems contain hard constraints that significantly reduce the space of feasible assignments. For example, in auctions, it is not possible to allocate an item to more than one bidder. In meeting scheduling, it is not possible to set two different start times for a given meeting. Unfortunately, DPOP does not take advantage of the pruning power of these hard constraints, and sends messages that explicitly represent all value combinations, including many infeasible ones. H-DPOP addresses this issue by using Constraint Decision Diagrams (CDD) introduced by Cheng and Yap (2005) to compactly represent UTIL messages by excluding unfeasible combinations. Performance improvements of several orders of magnitude can be achieved, especially on highly constrained problems (Kumar et al., 2007).

MB-DPOP (Petcu & Faltings, 2007) uses the idea of *cycle cutsets* (Dechter, 2003) to explore parts of the search space sequentially. Dense parts of the problem are explored by iterating through assignments of a subset of nodes designated as "cycle cuts", and for each assignment performing a limited UTIL propagation similar to the one from DPOP. Easy parts of the problem are explored with one-shot UTIL messages, exactly as in DPOP. MB-DPOP offers thus a configurable tradeoff between the number of the messages exchanged, and the size of these messages and the memory requirements.

7.2 Achieving Faithfulness with other DCOP Algorithms

The partition principle, described in Section 4.3, is algorithm independent. The question as to whether another, optimal DCOP algorithm can be made faithful therefore revolves, critically, around whether the algorithm will satisfy the robustness requirement of the partition priciple. We make the following observations:

- Robustness in the first sense, i.e. that no agent A_i can influence the solution to the efficient SCP without agent A_i, is always achievable at the cost of restarting computation on the marginal problem with each agent removed in turn, just as we proposed for simple-M-DPOP.

- Robustness in the second sense, i.e. that no agent A_i can influence the report(s) that the bank receives about the negative externality that A_i imposes on the rest of the system, conditioning on the solutions to the main problem and the problem without A_i, requires that the DCOP algorithm terminates with every agent knowing the part of the solution that is relevant in defining its own utility; the robustness property then follows by disaggregation of payments.

Thus, if one is content to restart the DCOP algorithm multiple times, then the same kinds of results that we provide for simple-M-DPOP are generally available. This is possible because of the already mentioned locality property of payments, which follows from the disaggregation of the VCG payment across agents in Eq. (10) and because of the information and communication structure of DCOP.

The other useful property of DCOP in the context of self-interested agents, and worth reemphasizing, is that it is possible to retain faithfulness even when one agent plays a *pivotal role* in connecting the problem graph. Suppose that problem, $DCOP(-A_i)$, becomes disconnected without A_i. But, if this is the case then its optimal solution is represented by the union of the optimal solutions in each connected subcomponent of the problem, and no information needs to flow between disconnected components either for the purpose of solving the problem or for the purpose of reporting the components of agent A_i's tax.

We discuss in the following two sections the adaptation of the two other most prominent complete DCOP algorithms: ADOPT (Modi et al., 2005) and OptAPO (Mailler & Lesser, 2004).

We discuss in the following two sections the adaptation of the two other most prominent complete DCOP algorithms: ADOPT (Modi et al., 2005) and OptAPO (Mailler & Lesser, 2004). We consider the computational aspects of making these algorithms faithful, specifically the issues related to the efficient handling of replica variables and to providing for reusability from the main to the marginal problems.

7.2.1 USING ADOPT FOR FAITHFUL, EFFICIENT SOCIAL CHOICE

ADOPT is a polynomial-space search algorithm for DCOP that is guaranteed to find the globally optimal solution while allowing agents to execute asynchronously and in parallel. The agents in ADOPT make local decisions based on conservative cost estimates. ADOPT also works on a DFS arrangement, constructed as detailed in Section 3.1.1. Roughly speaking, the main process that is executed in ADOPT is a backtrack search on the DFS tree.

Adaptation of ADOPT to the DCOP Model with Replicated Variables. ADOPT's complexity is given by the number of messages, which is exponential in the height of the DFS tree. Similar to DPOP, using the DCOP model with replicated variables could artificially increase the complexity of the solving process. Specifically, the height of the DFS tree is increased when using replicated variables compared to the centralized problem graph. ADOPT can be modified to exploit the special structure of these replicated local variables in a similar way as DPOP. Specifically, ADOPT should explore sequentially only the values of the original variable, and ignore assignments where replicas of the same variable take different values. This works by allowing just the agent that owns the highest replica of each variable to freely choose values for the variable. This agent then announces the new value of the variable to all other agents owning replicas of the variable. These other agents would then consider just the announced value for their replicas, add their own corresponding utilities, and continue the search process. Using this special handling of the replica variables, the resulting complexity is no longer exponential in the height of the distributed DFS tree, but in the height of the DFS tree obtained by traversing the original problem graph. For example, in Figure 2, it is sufficient to explore the values of M_3^2, and directly assign these values to M_3^3 and M_3^1 via *VALUE* messages, without trying all the combinations of their values. This reduces ADOPT's complexity from exponential in 6, to exponential in 3.

Reusability of Computation in ADOPT. Turning to the re-use of computation from the main to the marginal problems, we note that because ADOPT uses a DFS arrangement then it is easy to identify which parts of the DFS arrangement for the main problem are impossible for an agent to manipulate, and therefore can be "reused" while computing the solution to the marginal problem with that agent removed. However, a major difference between DPOP and ADOPT is that in DPOP, each agent stores its outgoing *UTIL* message, and thus has available all the utilities contingent to all assignments of the variables in the agent's separator. This makes it possible for the agent to simply reuse that information in all marginal economies where the structure of the DFS proves that this is safe. In contrast, ADOPT does not store all this information because of its linear memory policy. This in turn makes it impossible to reuse computation from the main problem to the marginal problems. All marginal problems have to be solved from scratch, and thus the performance would scale poorly as problem size increases and even in structured problems such as meeting scheduling.

We see two alternatives for addressing this problem: (a) renounce linear memory guarantees, and use a caching scheme like for example in NCBB (Chechetka & Sycara, 2006): this would allow for a similar reusability as in M-DPOP, where previously computated utilities can be extracted from the cache instead of having to be recomputed. Alternatively, (b) one can devise a scheme where the previously computed best solution can be saved as a reference, and subsequently used as an approximation while solving the marginal problems. This could possibly provide better bounds and thus allow for better pruning, such that some computation could be saved. Both these alternatives are outside the scope of this paper, and considered for future work.

7.2.2 USING OPTAPO FOR FAITHFUL, EFFICIENT SOCIAL CHOICE

OptAPO (Mailler & Lesser, 2004) is the other most popular algorithm for DCOP. Similar to the adaptations of DPOP and ADOPT to social choice, OptAPO can also be made to take advantage of the special features of the DCOP model with replicated variables. Its complexity then would not be artificially increased by the use of this DCOP model. OptAPO has the particularity that it uses "mediator agents" to *centralize subproblems* and solve them in dynamic and asynchronous mediation sessions, i.e. partial centralization. The mediator agents then announce their results to the other agents, who have previously sent their subproblems to the mediators. This process alone would introduce additional possibility for manipulation in a setting with self interested agents. However, using the VCG mechanism addresses this concern and agents will choose to behave correctly according to the protocol.

As with ADOPT, the main issue with using OptAPO for faithful social choice is the reusability of computation from the main to the marginal problems. Specifically, consider that while solving the main problem, a mediator agent A_i has centralized and aggregated the preferences of a number of other agents, while solving mediation problems as dictated by the OptAPO protocol. Subsequently, when trying to compute the solution to the marginal problem without agent A_i, all this computation has to go to waste, as it could have been manipulated by A_i while solving the main problem. Furthermore, since OptAPO's centralization process is asynchronous and conflict-driven as opposed to structure-driven as in M-DPOP, it is unclear whether *any* computation from the main problem could be safely reused in any of the marginal problems. To make matters worse, experimental studies (Davin & Modi, 2005; Petcu & Faltings, 2006) show that in many situations, OptAPO ends up relying on a single agent in the system to centralize and solve the whole problem. This implies that while solving the marginal problem without that agent, one can reuse zero effort from the main problem.

7.3 Loss in Utility due to Wasting the VCG Taxes

In the VCG mechanism, each agent's *net utility* is the difference between the utility it derives from the optimal solution and the VCG tax it has to pay. The net utility of the whole group of agents is the sum of individual net utilities of the agents, i.e. the total utility from the assignment of values to variables but net of the total payment made by agents to the bank. *This loss in utility while using M-DPOP can be as great as 35% of the total utility of the optimal solution in the meeting scheduling domain.* As the problem size increases, more and more money has to be burnt in the form of VCG taxes. Similar waste has been observed by others; e.g., Faltings (2004), also in the context of efficient social choice.

One cannot naively redistribute the payment back to the agents, for instance sharing the payments equally across all agents would break faithfulness. For example, agent A_i would prefer for the other agents to make greater payments, in order to receive a larger repayment from the bank. The faithfulness properties of M-DPOP would unravel. On the other hand, when the problem has inherent structure then it is possible to redistribute some fraction of the payments back to agents. This idea of careful redistribution was suggested in Bailey (1997), and subsequently extended by Cavallo (2006), Guo and Conitzer (2007) and Moulin (2007). Another approach, advocated for example by Faltings (2004), is to simply preclude an agent from the problem and transfer the payments to this agent. All this work is in a centralized context.

An important issue for future work, then, is to study the budget surplus that accrues to the bank in M-DPOP and seek to mitigate this welfare loss in a setting of distributed implementation. We defer any further discussion of this topic to future work, in which we will investigate methods to leverage structure in the problem in redistributing the majority of these payments back to agents without compromising either efficiency or faithfulness.

8. Conclusions

We have developed M-DPOP, which is a faithful, distributed algorithm with which to solve efficient social choice problems in multi-agent systems with private information and self-interest. No agent can improve its utility either by misreporting its local information or deviating from any aspect of the algorithm (e.g., computation, message-passing, information revelation.) The only centralized component is that of a bank that is able to receive messages about payments and collect payments. In addition to promoting efficient decisions, we minimize the amount of additional computational effort required for computing the VCG payments by reusing effort from the main problem. A first set of experimental results shows that a significant amount of the computation required in all the marginal problems can be reused from the main problem, sometimes above 87%. This provides near-linear scalability in massive, distributed social choice problems that have local structure so that the maximal induced tree width is small. A second set of experiments performed on problems without local structure shows that as the problem density increases, the amount of effort required increases, and the reusability of computation decreases. These results suggest that M-DPOP is a very good candidate for solving loose problems that exhibit local structure such that the induced width remains small. In addition to addressing the need to reduce the total payments made by agents to the bank, one issue for future work relates to the need to provide robustness when faced with *adversarial* or *faulty* agents: the current solution is fragile in this sense, with its equilibrium properties relying on other agents following the protocol. Some papers (Lysyanskaya & Triandopoulos, 2006; Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth, 2005; Shneidman & Parkes, 2003) provide robustness to mixture models (e.g. some rational, some adversarial) but we are not aware of any work with these mixture models in the context of efficient social choice. Another interesting direction is to find ways to allow for approximate social choice, for example with memory-limited DPOP variations (Petcu & Faltings, 2005a) while retaining incentive properties, perhaps in approximate equilibria. Future research should also consider the design of distributed protocols that are robust against false-name manipulations in which agents can participate under multiple pseudonyms (Yokoo et al., 2004), and seek to mitigate the opportunities for collusive behavior and the possibility of multiple equilibria that can exist in incentive mechanisms (Ausubel & Milgrom, 2006; Andelman, Feldman, & Mansour, 2007; Katz & Gordon, 2006).

Acknowledgments

Parkes is supported in part by National Science Foundation grants IIS-0238147, IIS-0534620 and an Alfred P. Sloan Foundation award. Petcu was supported by the Swiss National Science Foundation grant 200020-103421/1. The authors would like to thank Wei Xue for valuable feedback on several parts of the paper. We thank Jeffrey Shneidman for his feedback on an early version of this paper. We also thank Aaron Bernstein for valuable insights on the DFS reconstruction process. The three anonymous reviewers also provided excellent suggestions for improving the exposition of this work.

An earlier version of this paper appeared in the Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2006.

References

Abu-Amara, H. H. (1988). Fault-tolerant distributed algorithm for election in complete networks. *IEEE Trans. Comput.*, *37*(4), 449–453.

Aiyer, A. S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., & Porth, C. (2005). Bar fault tolerance for cooperative services. In *20th ACM Symposium on Operating Systems Principles*.

Andelman, N., Feldman, M., & Mansour, Y. (2007). Strong price of anarchy. In *ACM-SIAM Symposium on Discrete Algorithms 2007 (SODA'07)*.

Arnborg, S. (1985). Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, *25*(1), 2–23.

Ausubel, L., Cramton, P., & Milgrom, P. (2006). The clock-proxy auction: A practical combinatorial auction design. In Cramton et al. (Cramton et al., 2006), chap. 5.

Ausubel, L., & Milgrom, P. (2006). The lovely but lonely Vickrey auction. In Cramton et al. (Cramton et al., 2006), chap. 1.

Bailey, M. J. (1997). The demand revealing process: To distribute the surplus. *PublicChoice*, 107–126.

Ball, M., Donohue, G., & Hoffman, K. (2006). Auctions for the safe, efficient, and equitable allocation of airspace system resources. In Cramton, Shoham, S. (Ed.), *Combinatorial Auctions*. MIT Press.

Barbosa, V. (1996). *An Introduction to Distributed Algorithms*. The MIT Press.

Bayardo, R., & Miranker, D. (1995). On the space-time trade-off in solving constraint satisfaction problems.. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI-95*, Montreal, Canada.

Bidyuk, B., & Dechter, R. (2004). On finding minimal w-cutset. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 43–50, Arlington, Virginia, United States. AUAI Press.

Bikhchandani, S., de Vries, S., Schummer, J., & Vohra, R. V. (2002). Linear programming and Vickrey auctions. In Dietrich, B., & Vohra, R. (Eds.), *Mathematics of the Internet: E-Auction and Markets*, pp. 75–116. IMA Volumes in Mathematics and its Applications, Springer-Verlag.

Cavallo, R. (2006). Optimal decision-making with minimal waste: Strategyproof redistribution of vcg payments. In *Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS'06)*.

Chechetka, A., & Sycara, K. (2006). An any-space algorithm for distributed constraint optimization. In *Proceedings of AAAI Spring Symposium on Distributed Plan and Schedule Management*.

Cheng, K. C. K., & Yap, R. H. C. (2005). Constrained decision diagrams.. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, pp. 366–371, Pittsburgh, USA.

Cheung, T.-Y. (1983). Graph traversal techniques and the maximum flow problem in distributed computation.. *IEEE Trans. Software Eng.*, *9*(4), 504–512.

Cidon, I. (1988). Yet another distributed depth-first-search algorithm. *Inf. Process. Letters*, *26*(6), 301–305.

Collin, Z., Dechter, R., & Katz, S. (1991). On the Feasibility of Distributed Constraint Satisfaction. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91*, pp. 318–324, Sidney, Australia.

Collin, Z., Dechter, R., & Katz, S. (1999). Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*.

Collin, Z., & Dolev, S. (1994). Self-stabilizing depth-first search. *Information Processing Letters*, *49*(6), 297–301.

Cramton, P., Shoham, Y., & Steinberg, R. (Eds.). (2006). *Combinatorial Auctions*. MIT Press.

Davin, J., & Modi, P. J. (2005). Impact of problem centralization in distributed constraint optimization algorithms. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 1057–1063, New York, NY, USA. ACM Press.

Davis, R., & Smith, R. G. (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 63–109.

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

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dechter, R., & Mateescu, R. (2006). AND/OR search spaces for graphical models. *Artificial Intelligence*. To appear.

Dunne, P. E. (2005). Extremal behaviour in multiagent contract negotiation. *Journal of Artificial Intelligence Research (JAIR)*, *23*, 41–78.

Dunne, P. E., Wooldridge, M., & Laurence, M. (2005). The complexity of contract negotiation. *Artificial Intelligence Journal*, *164*(1-2), 23–46.

Endriss, U., Maudet, N., Sadri, F., & Toni, F. (2006). Negotiating socially optimal allocations of resources. *Journal of Artificial Intelligence Research*, *25*, 315–348.

Ephrati, E., & Rosenschein, J. (1991). The Clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-91*, pp. 173–178, Anaheim, CA.

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

Faltings, B., Parkes, D., Petcu, A., & Shneidman, J. (2006). Optimizing streaming applications with self-interested users using M-DPOP. In *COMSOC'06: International Workshop on Computational Social Choice*, pp. 206–219, Amsterdam, The Netherlands.

Feigenbaum, J., Papadimitriou, C., Sami, R., & Shenker, S. (2002). A BGP-based mechanism for lowest-cost routing. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing*, pp. 173–182.

Feigenbaum, J., Ramachandran, V., & Schapira, M. (2006). Incentive-compatible interdomain routing. In *Proceedings of the 7th Conference on Electronic Commerce*, pp. 130–139.

Feigenbaum, J., & Shenker, S. (2002). 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*, pp. 1–13.

Freuder, E. C., & Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, IJCAI-85*, pp. 1076–1078, Los Angeles, CA.

Gershman, A., Meisels, A., & Zivan, R. (2006). Asynchronous forward-bounding for distributed constraints optimization. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI-06)*, Riva del Garda, Italy.

Greenstadt, R., Pearce, J. P., & Tambe, M. (2006). Analysis of privacy loss in distributed constraint optimization. In *Proc. of Twenty-First National Conference on Artificial Intelligence (AAAI-06)*.

Guo, M., & Conitzer, V. (2007). Worst-case optimal redistribution of vcg payments. In *Proceedings of the 8th ACM Conference on Electronic Commerce (EC-07)*, pp. 30–39.

Huebsch, R., Hellerstein, J. M., Lanham, N., et al. (2003). Querying the Internet with PIER. In *VLDB*.

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

Jackson, M. O. (2000). Mechanism theory. In *The Encyclopedia of Life Support Systems*. EOLSS Publishers.

Jackson, M. O. (2001). A crash course in Implementation theory. *Social Choice and Welfare*, *18*(4), 655–708.

Katz, J., & Gordon, S. D. (2006). Rational secret sharing, revisited. In *Proc. Security and Cryptography for Networks*.

Kloks, T. (1994). *Treewidth, Computations and Approximations*, Vol. 842 of *Lecture Notes in Computer Science*. Springer.

Krishna, V. (2002). *Auction Theory*. Academic Press.

Kumar, A., Petcu, A., & Faltings, B. (2007). H-DPOP: Using hard constraints to prune the search space. In *IJCAI'07 - Distributed Constraint Reasoning workshop, DCR'07*, pp. 40–55, Hyderabad, India.

Lavi, R., Mu'alem, A., & Nisan, N. (2003). Towards a characterization of truthful combinatorial auctions. In *Proc. 44th Annual Symposium on Foundations of Computer Science*.

Leyton-Brown, K., Pearson, M., & Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of ACM Conference on Electronic Commerce, EC-00*, pp. 235–245.

Leyton-Brown, K., & Shoham, Y. (2006). A test suite for combinatorial auctions. In Cramton, P., Shoham, Y., & Steinberg, R. (Eds.), *Combinatorial Auctions*, chap. 18. MIT Press.

Lysyanskaya, A., & Triandopoulos, N. (2006). Rationality and adversarial behavior in multi-party computation. In *26th Annual Int. Cryptology Conference (CRYPTO'06)*.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., & Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS-04*.

Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, *1*, 438–445.

Mailler, R., & Lesser, V. (2005). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research (JAIR)*.

Mas-Colell, A., Whinston, M. D., & Green, J. R. (1995). *Microeconomic Theory*. Oxford University Press.

Mishra, D., & Parkes, D. (2007). Ascending price Vickrey auctions for general valuations. *Journal of Economic Theory*, *132*, 335–366.

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

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

Moulin, H. (2007). Efficient, strategy-proof and almost budget-balanced assignment. Tech. rep., Rice University.

Mu'alem, A. (2005). On decentralized incentive compatible mechanisms for partially informed environments. In *Proc. ACM Conf. on Electronic Commerce (EC)*.

Ostrovsky, R., Rajagopalan, S., & Vazirani, U. (1994). Simple and efficient leader election in the full information model. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 234–242, New York, NY, USA. ACM Press.

Parkes, D. C., Kalagnanam, J. R., & Eso, M. (2001). Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 1161–1168.

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

Parkes, D. C., & Ungar, L. H. (2000). Iterative combinatorial auctions: Theory and practice. In *Proc. 17th National Conference on Artificial Intelligence (AAAI-00)*, pp. 74–81.

Petcu, A. (2006). FRODO: A FRamework for Open and Distributed constraint Optimization. Technical report no. 2006/001, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland. `http://liawww.epfl.ch/frodo/`.

Petcu, A., & Faltings, B. (2005a). A-DPOP: Approximations in distributed optimization. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, pp. 802–806, Sitges, Spain.

Petcu, A., & Faltings, B. (2005b). DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, pp. 266–271, Edinburgh, Scotland.

Petcu, A., & Faltings, B. (2006). PC-DPOP: A partial centralization extension of DPOP. In *In Proceedings of the Second International Workshop on Distributed Constraint Satisfaction Problems, ECAI'06*, Riva del Garda, Italy.

Petcu, A., & Faltings, B. (2007). MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-07*, Hyderabad, India.

Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., & Seltzer, M. (2006). Network-Aware Operator Placement for Stream-Processing Systems. In *ICDE*.

Rassenti, S. J., Smith, V. L., & Bulfin, R. L. (1982). A combinatorial mechanism for airport time slot allocation. *Bell Journal of Economics*, *13*, 402–417.

Roberts, K. (1979). The characterization of implementable rules. In Laffont, J.-J. (Ed.), *Aggregation and Revelation of Preferences*, pp. 321–348. North-Holland, Amsterdam.

Rosenschein, J. S., & Zlotkin, G. (1994). Designing conventions for automated negotiation. *AI Magazine*. Fall.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, *135*, 1–54.

Sandholm, T. W. (1993). An implementation of the Contract Net Protocol based on marginal-cost calculations. In *Proc. 11th National Conference on Artificial Intelligence (AAAI-93)*, pp. 256–262.

Sandholm, T. W. (1996). Limitations of the Vickrey auction in computational multiagent systems. In *Second International Conference on Multiagent Systems (ICMAS-96)*, pp. 299–306.

Shneidman, J., & Parkes, D. C. (2003). Rationality and self-interest in peer to peer networks. In *2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*.

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

Silaghi, M.-C., Sam-Haroud, D., & Faltings, B. (2000). Asynchronous search with aggregations. In *AAAI/IAAI*, pp. 917–922, Austin, Texas.

Solotorevsky, G., Gudes, E., & Meisels, A. (1996). Modeling and Solving Distributed Constraint Satisfaction Problems (DCSPs). In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (CP'96)*, pp. 561–562, Cambridge, Massachusetts, USA.

Sycara, K., Roth, S. F., Sadeh-Koniecpol, N., & Fox, M. S. (1991). Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(6), 1446–1461.

Wellman, M. P. (1993). A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, *1*, 1–23.

Wellman, M. P. (1996). Market-oriented programming: Some early lessons. In Clearwater, S. H. (Ed.), *Market-Based Control: A Paradigm for Distributed Resource Allocation*, chap. 4, pp. 74–95. World Scientific.

Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pp. 614–621.

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

Yokoo, M., Sakurai, Y., & Matsubara, S. (2004). The effect of false-name bids in combinatorial auctions: New Fraud in Internet Auctions. *Games and Economic Behavior*, *46*(1), 174–188.

Zhang, W., & Wittenburg, L. (2003). Distributed breakout algorithm for distributed constraint optimization problems - DBArelax. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-03)*, Melbourne, Australia.

Zlotkin, G., & Rosenschein, J. S. (1996). Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, *5*, 163–238.