Incentive Compatible Multiagent Constraint Optimization

Adrian Petcu[1] and Boi Faltings[1]

Ecole Polytechnique Federale de Lausanne (EPFL), CH-1015 Lausanne (Switzerland)
{adrian.petcu, boi.faltings}@epfl.ch
http://liawww.epfl.ch

Abstract. We present in this paper an incentive-compatible distributed optimization method applied to social choice problems. The method works by computing and collecting VCG taxes in a distributed fashion. This introduces a certain resilience to manipulation from the problem solving agents. An extension of this method sacrifices Pareto-optimality in favor of budget-balance: the solutions chosen are not optimal anymore, but the advantage is that the self interested agents pay the taxes between themselves, thus producing no tax surplus. This eliminates unwanted incentives for the problem solving agents, ensuring their faithfulness.

1 Introduction

In this paper we concentrate on social choice problems, which are ubiquitous in our society. Typically, such problems include a set of public entities, that take some decisions based on the preferences expressed by a set of (private) agents. The goal is to adopt the set of decisions that best match the preferences expressed by the private agents, possibly subject to a set of feasibility constraints.

In systems with self interested agents, it is often a problem to guarantee optimal outcomes because each agent may have the incentive to manipulate the system in a way that is profitable to itself. Such manipulations steer the final outcome away from a global optimum which is otherwise achievable.

The VCG tax mechanism is a way to ensure that the agents in the system are always better off by declaring their true preferences, thus allowing for the optimal outcome to be chosen. The mechanism works by fining the participating agents with taxes which are proportional to the damage that they cause to others. Thus, the agents do not have an incentive to understate their valuations because the outcome chosen would not be the best for them, and do not overstate because that would induce a high amount of tax they would have to pay for hurting the others.

Traditionally, such mechanisms have been studied and applied in centralized systems. Feigenbaum and Shenker started in [5] a new line of research in *Distributed Algorithmic Mechanism Design (DAMD)*. DAMD is a fusion of the more traditional algorithmic-oriented AI and the recent interest in distributed computing and incentive compatibility in multiagent environments. They focus on multicast cost sharing and interdomain routing problems. This work is similar in spirit with theirs, but our focus is on constraint optimization applied to public decision problems.

Parkes and Shneidman present in [10] an approach for incentive compatible distributed computation. The goal in their approach is to distribute the computation to the self interested agents themselves, and take the computational burden off the center. They use VCG taxes to make *faithfulness* (see also [12]) an ex-post Nash equilibrium: the agents have the incentive to execute the correct algorithm, without manipulation. Their approach requires the presence of a center that selects an outcome, enforces it, and collects taxes. Trusted channels between the center and the agents are required such that the agents report their types to the center.

In our approach both the optimization itself, and the computation of VCG taxes are done in a distributed fashion by the agents controlling the public decision variables. We present also a budget balanced extension of the algorithm that eliminates all interest of the problem solving agents (public entities) in the problem, therefore ensuring faithfulness in the sense of [12].

In the following, we present in Section 2 some definitions and notation, in Section 3 the basic optimization algorithm that will be used in this paper, and in Section 4 a VCG-based extension of the basic optimization method. Section 5 presents a randomized algorithm that is budget balanced. Section 6 presents experimental results on meeting scheduling problems, and Section 7 concludes.

2 Definitions & Notation

Definition 1. *A discrete* multiagent constraint optimization problem *(MCOP) is a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R} >$ *such that:*

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

$\mathcal{X} = \{X_1, ..., X_m\}$ *is the set of public decision variables/solving agents;*

$\mathcal{D} = \{d_1, ..., d_m\}$ *is a set of finite domains of the variables* $\mathcal{X}$.

$\mathcal{C} = \{c_1, ..., c_q\}$ *is a set of constraints, where a constraint* c_i *is a function* $c_i : d_{i1} \times .. \times d_{ik} \rightarrow \{-\infty, 0\}$ *that returns 0 for all allowed combinations of values of the involved variables, and* $-\infty$ *for disallowed ones.*

$\mathcal{R} = \{r_1, ..., r_p\}$ *is a set of relations, where a relation* r_i^j *is a function* $d_{i1} \times .. \times d_{ik} \rightarrow \mathbb{R}$ *specified by agent* A_j *which denotes how much utility* A_j *assigns to each possible combination of values of the involved variables (negative values can be thought of as costs).* R_j *is the set of relations specified by agent* A_j.

This framework allowes us to model social choice-like problems, where a set of "public" agents $\mathcal{X}$ jointly choose an overall optimal outcome out of a set of possible solutions. The feasibility of a solution is decided by the constraints in $\mathcal{C}$, which are domain dependent, and are imposed by the agents $\mathcal{X}$. The choice between several feasible solutions is made according to the preferences of the "private" agents $\mathcal{A}$, stated through the relations $\mathcal{R}$. Formally, the optimal solution to such an optimization problem is a complete instantiation X^* of all variables in $\mathcal{X}$, s.t. $X^* = argmax_X(\sum_{r_i \in \mathcal{R}} r_i(X) + \sum_{c_i \in \mathcal{C}} c_i(X))$, where the values of r_i and c_i are their corresponding values for the particular instantiation X. Notice that the second sum is either $-\infty$ if X is an infeasible assignment, or 0 if it is feasible. We restrict our attention to problems that have feasible solutions. Notice that simply maximizing utility is sufficient

to find the optimal, feasible solution, since if any of the constraints in $\mathcal{C}$ are unsatisfied, the resulting overall utility is $-\infty$.

In this paper we deal with unary and binary relations, being well known that higher arity relations can also be expressed in these terms with little modifications.

This framework is similar to a weighted CSP framework where we allow both positive and negative costs and we do utility maximization as opposed to cost minimization.

3 DPOP: a dynamic programming algorithm for MCOP

Each agent $A_i \in \mathcal{A}$ has a set of preferences on the outcome of the optimization problem, expressed by the set of relations $R_i \subset \mathcal{R}$. The agents $\mathcal{A}$ declare their relations to the agents in $\mathcal{X}$ concerned by those relations. Afterwards, the agents X_i execute a distributed optimization procedure yielding an assignment $\mathcal{X}^*$ of the variables X_i that maximizes the overall utility for the agents $\mathcal{A}^*$.

The optimization algorithm of choice is *DPOP* ([11]). *DPOP* is an instance of the general bucket elimination scheme from [2], which is adapted for the distributed case, and uses a DFS traversal of the problem graph as an ordering.

For now, we assume the agents $\mathcal{A}$ declare their relations *truthfully*, therefore *DPOP* produces the correct optimal solution. In Section 4 we adapt the VCG mechanism to *DPOP* to ensure truthfulness.

DPOP has 3 phases. In the first phase (see Section 3.1), the pseudotree structure is established. One node is chosen among the nodes from $\mathcal{X}$, and a custom distributed DFS algorithm is initiated from that node. The second phase (see section 3.2) is a bottom-up utility propagation, and the third phase (see section 3.3) is a top-down value assignment propagation. A formal description (pseudocode) can be found at the end of section 3.3.

It has been proved in [11] that *DPOP* produces a linear number of messages. Its complexity lies in the size of the *UTIL* messages (the *VALUE* messages have linear size). The largest *UTIL* message produced by Algorithm 1 is space-exponential in the width of the pseudotree induced by the DFS ordering used.

3.1 Pseudotrees

Definition 2. *A pseudo-tree arrangement of a graph G is a rooted tree with the same nodes as G and the property that adjacent nodes from the original graph fall in the same branch of the tree (e.g. X_0 and X_{12} in Figure 1).*

As it is already known, a DFS (depth-first search) tree is also a pseudotree, although the inverse does not always hold. We thus use as pseudotree a DFS tree, generated through a distributed DFS algorithm. Due to lack of space, we can only sketch this algorithm here. The process is started from the root, and the nodes pass messages to their neighbors, adding themselves in the context of these messages. Whenever a node receives a message from a neighbor, with itself in the context, then pseudo parent/pseudo child relationships are established, otherwise parent/child relationships. The result of this algorithm is that all nodes consistently label each other as parent/child or pseudoparent/pseudochild.

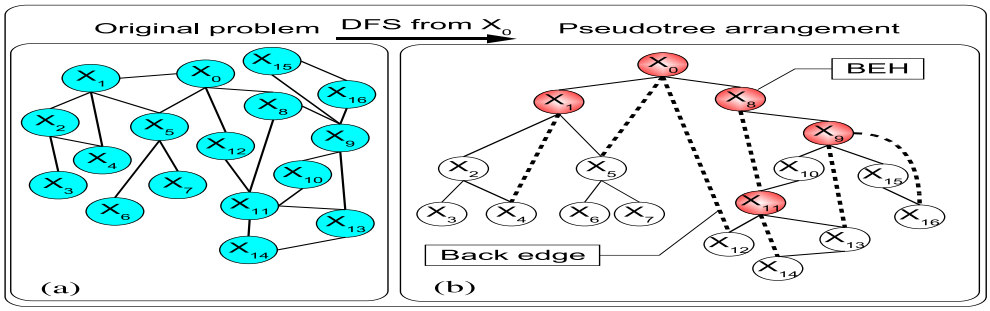

Fig. 1. *A problem graph and one of its possible rooted DFS trees.*

Figure 1 shows an example of a pseudotree that we shall refer to in the rest of this paper. It consists of *tree edges*, shown as solid lines, and *back edges*, shown as dashed lines, that are not part of the DFS tree. We call a path in the graph that is entirely made of tree edges, a *tree-path*. A *tree-path associated with a back-edge* is the tree-path connecting the two nodes involved in the back-edge (such a tree path is always unique, and included in a branch of the tree). For each back-edge, the higher node involved in that back-edge is called the *back-edge handler* - BEH (e.g. 0, 1, 8).

We define the following elements (refer to Figure 1):

Definition 3. ***P(****X****)*** *- the parent of a node X: the single node on a higher level of the pseudotree that is connected to the node X directly through a tree edge (e.g. $P(X_4) = X_2$).* ***C(****X****)*** *- the children of a node X: the set of nodes lower in the pseudotree that are connected to the node X directly through tree edges (e.g. $C(X_1) = \{X_2, X_5\}$).* ***PP(****X****)*** *- the pseudo-parents of a node X: the set of nodes higher in the pseudotree that are connected to the node X directly through back-edges ($PP(X_4) = \{X_1\}$).* ***PC(****X****)*** *- the pseudo-children of a node X: the set of nodes lower in the pseudotree that are connected to the node X directly through back-edges (e.g. $PC(X_0) = \{X_5, X_{12}\}$).*

3.2 Bottom-up *UTIL* propagation

Definition 4. *$UTIL_i^j$ - the UTIL message sent by agent X_i to agent X_j; this is a multidimensional matrix, with one dimension for each variable present in the context. $dim(UTIL_i^j)$ - the whole set of dimensions (variables) of the message ($X_j \in dim(UTIL_i^j)$ always). The semantics of such a message is similar to an n-ary relation having as scope the variables in the context of this message (its dimensions). The size of such a message is the product of the domain sizes of the variables from the context.*

Definition 5. *The $\oplus$ operator (join): $UTIL_i^j \oplus UTIL_k^j$ is the join of two UTIL matrices. This is also a matrix with $dim(UTIL_i^j) \cup dim(UTIL_k^j)$ as dimensions. The value of each cell in the join is the sum of the corresponding cells in the two source matrices.*

The semantics of this operation is the creation of a new relation between the union of the variables, equivalent to the two relations.

Definition 6. *The $\perp$ operator (projection): if $X_k \in dim(UTIL_i^j)$, $UTIL_i^j \perp_{X_k}$ is the projection through optimization of the $UTIL_i^j$ matrix along the X_k axis: for each tuple of variables in $\{dim(UTIL_i^j) \setminus X_k\}$, all the corresponding values from $UTIL_i^j$ (one for each value of X_k) are tried, and the best one is chosen. The result is a matrix with one less dimension (X_k).*

This projection has the semantics of a precomputation of the optimal utility achieved with the optimal values of X_k, for each instantiation of the other variables. It can also be seen as eliminating variable X_k and producing a new relation on the rest of the variables.

The *UTIL* propagation starts bottom-up from the leaves and propagates up to the root only through tree edges. The leaf nodes initiate this process, and then each node X_i relays these messages only to its parent:

- Wait for *UTIL* messages from all children. Perform join, project self out of the join and send the result to the parent.
- If root node, X_i receives all its *UTIL* messages as vectors with a single dimension, itself. It can then compute the optimal overall utility corresponding to each one of its values (by joining all the incoming *UTIL* messages) and pick the optimal value for itself (project itself out).

Top down *UTIL* propagation After the bottom-up propagation, the root has global information, but all other nodes have accurate *UTIL* information only about their subtrees. We extend the *UTIL* propagation by making it *uniform*: now it also goes top-down, from each node to its children. A *UTIL* message from a parent to its child summarizes the utility information from all the problem except the subtree of that child.

If a node joins all the messages received from all its tree neighbors (parent and children), then that node obtains a global view of the system, thus becoming logically equivalent to the root. Projecting everything out (itself included) of this join gives the optimal value in the overall optimal solution.

The process is initiated by the root. Each X_i (root included) computes for each of its children X_j a $UTIL_i^j$ message. The computation is similar to the bottom-up one: *UTIL* messages from all neighbors except the respective child are joined, projections are applied, and the message is send to the child.

3.3 *VALUE* propagation

The *VALUE* phase is a top-down propagation phase, initiated by the root after receiving all *UTIL* messages. Based on these *UTIL* messages, the root assigns itself the optimal value that maximizes the sum of utility of all its subtrees (overall utility). Then it announces its decision to its children and pseudochildren by sending them a *VALUE($X_i \leftarrow v_i^*$)* message.

Upon receipt of the *VALUE* message from its parent, each node is able to pick the optimal value for itself in a similar fashion, and then in its turn, send its *VALUE* messages. When the *VALUE* propagation reaches the leaves, all variables in the problem are instantiated to their optimal values, and the algorithm terminates.

Algorithm 1: *DPOP - distributed pseudotree optimization procedure.*

DPOP($\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}$): each agent X_i does:

Construct DFS tree; after completion, X_i knows $P(i), PP(i), C(i), PC(i)$

Bottom-up *UTIL* propagation protocol

1 wait for *UTIL* messages (X_k, $UTIL_k^i$) from all children $X_k \in C(i)$

2 $JOIN_i^{P(i)} = \left(\left(\bigoplus_{c \in C(i)} UTIL_c^i\right) \oplus \left(\bigoplus_{c \in \{P(i) \cup PP(i)\}} R_i^c\right)\right)$

3 **if** X_i *is root* **then** start *VALUE* propagation, and top-down *UTIL* propagation

4 **else** compute $UTIL_{X_i}^{P(i)} = JOIN_i^{P(i)} \perp_{X_i}$ and send it to P(i)

Top-down *UTIL* propagation protocol

5 **foreach** $X_k \in C(X_i)$ **do** compute $UTIL_i^k$ and send it to X_k

***VALUE* propagation protocol**

6 get and store in $agent\_view$ all *VALUE* messages ($X_k \leftarrow v_k^*$)

7 $v_i^* \leftarrow argmax_{X_i}\left(JOIN_{X_i}^{P(i)}[v(P(i)), v(PP(i))]\right)$

8 Send *VALUE*($X_i \leftarrow v_i^*$) to all $C(i)$ and $PC(i)$

4 VCG-based incentive compatible optimization protocol

We now consider that the agents $A_i \in \mathcal{A}$ are self interested, and thus will try to adjust their declarations such that they obtain beneficial manipulations of the optimization process.

It was shown in [6] that the only possible incentive compatible mechanism for optimization is of the form of a VCG mechanism ([13, 1, 7]). Ephrati and Rosenschein show in [3] for the first time how Clarke taxes can be used in multiagent systems for coordination problems in a way that induces incentive compatibility. We show in the following how to compute the Clarke taxes in a distributed fashion, and present a modified version of the *DPOP* algorithm that induces incentive compatibility.

Notice that since *DPOP* is a complete algorithm, it does not suffer from the non-truthfulness problem of approximate methods, as shown by Nisan and Ronen in [9].

To be able to compute the Clarke taxes, we need a mechanism that systematically leaves out an agent from the optimization process throughout the whole problem. We achieve this by simply including into each UTIL message that travels through the system, a corresponding part for each of the agents in $\mathcal{A}$. Namely, a $UTIL_i^j$ message sent by agent X_i to X_j is the union of n $UTIL_i^j(A_l)^*$ messages and n $UTIL_i^j(-A_l)$ messages (one for each agent A_l). The complexity of this scheme is thus $2 \times n \times O(DPOP)$, where $n = |\mathcal{A}|$. A message $UTIL_i^j(A_l)^*$ is equivalent to a normal $UTIL_i^j$ message, but is computed by aggregating only the utility agent A_l obtains in the optimal solution. A message $UTIL_i^j(-A_l)$ is equivalent to the utility of all agents but A_l, and is computed by aggregating the utility of all agents except A_l, in the solution obtained by systematically ignoring A_l's relations.

When all propagations are completed, all agents X_i are able to compute the VCG taxes for all agents A_l. They do this as detailed in Algorithm 2, lines 8-13. In words,

Algorithm 2: *Truthful Distributed VCG-based optimization procedure.*

ICDPOP($\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}$**)-** changes from DPOP:

UTIL Propagation()

wait for *UTIL* messages (X_k, $UTIL_k^i$) from all children $X_k \in C(i)$

foreach $A_l \in \mathcal{A}$ **do**

$JOIN_i^{P_i}(A_l)^* = \left(\bigoplus_{c \in C_i} UTIL_c^i(A_l)^*\right) \oplus \left(\bigoplus_{c \in \{P_i \cup PP_i\}} R_i^c(A_l) \oplus C_i^c\right)$

$JOIN_i^{P_i}(-A_l) = \left(\bigoplus_{c \in C_i} UTIL_c^i(-A_l)\right) \oplus \left(\bigoplus_{c \in \{P_i, PP_i\}} R_i^c(\mathcal{A} \setminus A_l) \oplus C_i^c\right)$

$UTIL_i^{P(i)}(A_l)^* = JOIN_i^{P(i)}(A_l)^* \perp_{X_i}$

$UTIL_i^{P(i)}(-A_l) = JOIN_i^{P(i)}(-A_l) \perp_{X_i}$

end

send $UTIL$ message to parent, $UTIL_i^j = \bigcup_{A_l} \{UTIL_i^{P(i)}(A_l)^*, UTIL_i^{P(i)}(-A_l)\}$

when last *UTIL* message arrives (from $P(X_i)$), execute Compute_taxes()

Compute_taxes()

foreach $A_l \in \mathcal{A}$ **do**

$JOIN(A_l)^* = \oplus_{X_j \in TreeNeighbors(X_i)} UTIL_j^i(A_l)^*$ (see section 3.2)

$JOIN(-A_m) = \oplus_{X_j \in TreeNeighbors(X_i)} UTIL_j^i(A_l)^{-A_m}$ (see section 3.2)

$UTIL(A_l)^* = JOIN(A_l)^* \perp_{\mathcal{X}}$

$UTIL(-A_m) = JOIN(-A_m) \perp_{\mathcal{X}}$

$TAX(A_l) = UTIL(-A_l) - \sum_{A_m \neq A_l} UTIL(A_m)^*$

cash in $\frac{TAX(A_l)}{m}$ from A_l

end

the tax that A_l has to pay equals the difference between the utility of the other agents in the solution when A_l is not present, and their utility when A_l is present.

We imagine that the agents X_i can split up between themselves the amount of tax that they collect from the agents A_l, in order to cover their costs for running the optimization process. Because the taxes are computed in a distributed fashion, and all agents $\mathcal{X}$ compute and receive the same $TAX(A_i)/m$ from the agents A_i it is more difficult for an agent X_j to simply claim exaggerated taxes from the agents A_i. Alternatively, the tax can be wasted ([3] shows that it must not return to the agents $\mathcal{A}$, otherwise incentive compatibility is broken).

The resulting algorithm is described in Algorithm 2. The *VALUE* phase is the same (as in DPOP, the optimal solution is chosen).

5 Budget balanced VCG-based distributed optimization

As it was already shown in game theory ([6, 8]), all mechanisms applied to general social choice problems that generate optimal outcomes must use a VCG-like tax, and cannot be budget balanced.

This poses sometimes a problem, since the collected tax can create undesired incentives for the entity collecting it (an auctioneer will introduce false bids to drive up the prices, a power plant operator will create artificial shortages, etc.) In our case, the agents $\mathcal{X}$ have the incentive to manipulate the optimization such that bad solutions are

obtained when all the agents $\mathcal{A}$ are present, and good solutions are obtained when individual agents A_i are left out. The differences in utility translate into VCG taxes that they may collect afterwards. This problem can be solved either by throwing away the tax (utility is wasted), or by designing a budget balanced scheme that generates no tax surplus. Either one of these alternatives ensures *algorithm faithfulness* as defined by Shneidman and Parkes in [12], because the agents $\mathcal{X}$ do not have any interest to cheat anymore.

It has been shown in [4] that if one renounces Pareto optimality (not necessarily optimal solutions are generated), then it is possible to have a budget-balanced, incentive compatible protocol that generally generates good solutions. The basic idea is to randomly leave one agent out of the optimization process, and make the others pay their taxes to the one which was left out. The mechanism is obviously budget balanced, since the taxes are paid between the agents, but it is no longer Pareto optimal, because the solution obtained in the end is not the optimal one (the relations of the excluded agent were left out of the optimization). It was shown in [4] that the mechanism is also incentive compatible and individually rational. The solutions found are good overall, since only a single agent is excluded from the optimization. Also, the excluded agent gets the tax surplus from the other agents as a compensation for the possible loss that it incurred by not having its relations included in the optimization.

We adapt this idea to our case by having the nodes $\mathcal{X}$ select randomly an agent A_i who is going to be left out of the optimization process. This can be done at the same time as choosing the root of the DFS tree, using a similar mechanism. Alternatively, the agents $\mathcal{A}$ themselves can select one of them to be excluded. Subsequently, all the *UTIL* propagations are performed ignoring A_i's relations.

The process is similar to the previous one applied to a problem that does not include A_i. The differences from *ICDPOP* are listed in Algorithm 3. The solution obtained now is the optimum for $\mathcal{A} \setminus A_i$. Also, the taxes computed by the new algorithm are not collected by $\mathcal{X}$ anymore, but by A_i. Thus, the agents $\mathcal{X}$ do not have any interest to manipulate the process anymore. This holds unless collusion with a subset of agents $\mathcal{A}_i$ is possible. Collusion is a well-known problem of the VCG tax, so here we assume it is prevented by an external mechanism.

Algorithm 3: *Budget balanced distributed incentive compatible optimization.*

BBICDPOP($\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}$**)-** changes from ICDPOP, when agent A_k is excluded:

UTIL Propagation()

...

1 compute $UTIL$ msgs: $UTIL = \bigcup_{A_l \neq A_k} \{UTIL_i^{P(i)}(A_l)^*, UTIL_i^{P(i)}(-A_l, A_k)\}$

Compute_taxes()

foreach $A_l \in \{\mathcal{A} \setminus A_k\}$ **do**

...

2 $TAX(A_l) = UTIL(-A_l, A_k) - \sum_{A_m \neq A_l, A_k} UTIL(A_m)^*(-A_k)$

3 instruct A_l to pay $\frac{TAX(A_l)}{m}$ to A_k

end

6 Experimental evaluation

We experimented with distributed meeting scheduling problems. These problems can be thought of as social choice problems if we have on one hand a set of agents $\mathcal{A}$ who want to schedule meetings, and on the other hand a set of agents $\mathcal{X}$ who will host these meetings. The COP model of such problems consists of a variable for each meeting, denoting its start time. There are inequality constraints between the meetings that share a participant (an agent A_i cannot participate in 2 meetings simultaneously). The agents $\mathcal{A}$ have preferences about the starting time of each meeting they participate in, stated through unary constraints on the respective variables.

Random meetings are generated, each with a certain utility for each agent. The agents $\mathcal{X}$ try to find the schedule that maximizes the overall utility for the agents $\mathcal{A}$.

Table 1 shows how our algorithm scales up with the size of the problems. The columns denote (in order): $|\mathcal{A}|$ is the number of self interested agents, $|\mathcal{X}|$ is the number of public decision variables, then the width of the resulting problems, the total number of messages sent during the algorithm, the maximal message size for simple *DPOP*, and the maximal message size for the VCG based *DPOP*.

As expected, we notice that the complexity increases across two dimensions: first the complexity of the underlying optimization problem (given by the induced width), and second, the number of self interested agents. The dependence of the complexity on the induced width produces very good results for loose problems, where the interests of the agents $\mathcal{A}$ are relatively decoupled. In these cases, where not all agents $\mathcal{A}$ are interested in all public variables $\mathcal{X}$, the resulting problems are loose, and easy to solve by the agents $\mathcal{X}$. The second complexity dimension can be observed by comparing consecutive rows that have the same induced width, but different numbers of self interested agents, e.g: rows 20-30, 56-70 and 80-100.

In any case, the fact that the algorithm produces a linear number of messages (even if they are big) is a great advantage in a distributed system, where a large number of small messages produce important overheads. For example, a backtracking based algorithm like a distributed branch and bound or ADOPT explore sequentially a large number of states, and produce an exponential number of small messages. This is why we think that a dynamic programming approach like *DPOP* is better suited for optimization tasks in distributed environments.

Agents ($\|\mathcal{A}\|$)	Meetings ($\|\mathcal{X}\|$)	Width	Messages	Max size(DPOP)	Max size(ICDPOP)
10	4	2	24	64	1280
20	5	3	33	512	20480
30	14	3	95	512	30720
40	15	4	109	4096	320K
56	27	5	201	32768	3.5M
70	34	5	267	32768	4.375M
80	41	6	324	262144	40M
100	50	6	373	262144	50M

Table 1. Evaluation on distributed meeting scheduling problems.

7 Conclusions and future work

We presented an incentive-compatible distributed optimization method, that computes and collects VCG taxes in a distributed fashion. We also present a budget-balanced extension of this method, that sacrifices Pareto-optimality. This eliminates unwanted incentives for the problem solving agents. We believe that this dynamic programming approach is a very good choice for multiagent systems, especially when the underlying problems are loosly connected.

As future work, we consider using approximate versions of *DPOP* to deal with difficult optimization problems, and computational complexity to counter the loss of incentive compatibility.

References

1. Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 18:19–33, 1971.
2. Rina Dechter. Bucket elimination: A unifying framework for processing hard and soft constraints. *Constraints: An International Journal*, 7(2):51–55, 1997.
3. E. Ephrati and J.S. Rosenschein. The Clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-91*, pages 173–178, Anaheim, CA, July 1991.
4. Boi Faltings. A budget-balanced, incentive-compatible scheme for social choice. In *Workshop on Agent-mediated E-commerce (AMEC) VI*. Springer Lecture Notes in Computer Science, 2004.
5. Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13. ACM Press, New York, 2002.
6. J. Green and J.J. Laffont. Incentives in public decision making. *Studies in public economics*, 1, 1979.
7. Theodore Groves. Incentives in teams. *Econometrica*, 41(4):617–31, 1973.
8. R.B. Myerson and M.A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29:265–281, 1983.
9. Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. In *ACM Conference on Electronic Commerce*, pages 242–252, 2000.
10. David C. Parkes and Jeffrey Shneidman. Distributed implementations of Vickrey-Clarke-Groves mechanisms. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-04)*, pages 261–268, New York, USA, 2004.
11. Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, Edinburgh, Scotland, Aug 2005.
12. Jeffrey Shneidman and David C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*.
13. William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.