PC-DPOP: A New Partial Centralization Algorithm for Distributed Optimization

Adrian Petcu and **Boi Faltings**
Ecole Polytechnique Fédérale de Lausanne
email: {adrian.petcu, boi.faltings}@epfl.ch

Roger Mailler
AI Center, SRI International
email: mailler@ai.sri.com

Abstract

Fully decentralized algorithms for distributed constraint optimization often require excessive amounts of communication when applied to complex problems. The OptAPO algorithm of [Mailler and Lesser, 2004] uses a strategy of partial centralization to mitigate this problem.

We introduce PC-DPOP, a new partial centralization technique, based on the DPOP algorithm of [Petcu and Faltings, 2005]. PC-DPOP provides better control over what parts of the problem are centralized and allows this centralization to be optimal with respect to the chosen communication structure.

Unlike OptAPO, PC-DPOP allows for a priory, exact predictions about privacy loss, communication, memory and computational requirements on all nodes and links in the network. Upper bounds on communication and memory requirements can be specified.

We also report strong efficiency gains over OptAPO in experiments on three problem domains.

1 Introduction

Constraint satisfaction and optimization are powerful paradigms that model a large 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 in order to find a solution. However, problems are sometimes naturally distributed, so Distributed Constraint Satisfaction (DisCSP) was formalized by Yokoo et al. in [Yokoo *et al.*, 1992]. Here, the problem is divided between a set of agents, which have to communicate among themselves to solve it. To address distributed optimization, complete algorithms like ADOPT, OptAPO and DPOP have been recently introduced.

ADOPT [Modi *et al.*, 2003] is a backtracking based bound propagation mechanism. It operates completely decentralized, and asynchronously. Its downside is that it may require a very large number of small messages, thus producing important communication overheads.

OptAPO is a hybrid between decentralized and centralized methods. It operates as a cooperative mediation process, where agents designated as mediators centralize parts of the problem in dynamic and asynchronous mediation sessions. Message complexity is significantly smaller than ADOPT's. However, it is possible that several mediators solve overlapping problems, thus needlessly duplicating effort. Furthermore, the asynchronous and dynamic nature of the mediation sessions make it impossible to predict what will be centralized where, how much of the problem will be eventually centralized, or how big a computational burden the mediators have to carry. It has been suggested in [Davin and Modi, 2005] that often a handful of nodes centralize most of the problem, and therefore carry out most of the computation.

DPOP [Petcu and Faltings, 2005] is a complete algorithm based on dynamic programming which generates only a linear number of messages. However, *DPOP* may produce large messages and require large amounts of memory, up to space exponential in the *induced width* of the problem. Therefore, *DPOP* may be infeasible for problems with high induced width.

We present PC-DPOP, a new hybrid algorithm that is controlled by a parameter k which characterizes the size of the largest message, and the amount of available memory. The algorithm proceeds as normal DPOP, except in the dense parts of the problem (width greater than k). These are clustered and centralized in the *root* of the cluster, which solves them with a centralized algorithm. Communication requirements over any link in the network are limited thus to $exp(k)$. The linear number of messages is preserved.

The rest of this paper is structured as follows: Section 2 formally describes the optimization problem, and introduces the *DPOP* and *OptAPO* algorithms. Section 3 introduces the PC-DPOP hybrid algorithm, which is evaluated empirically in Section 4. Section 5 relates PC-DPOP to existing work. Section 6 briefly discusses privacy, and Section 7 concludes.

2 Preliminaries

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

- $\mathcal{X} = \{X_1, ..., X_n\}$ *is a set of variables*
- $\mathcal{D} = \{d_1, ..., d_n\}$ *is a set of finite variable domains*
- $\mathcal{R} = \{r_1, ..., r_m\}$ *is a set of relations, where a relation* r_i *is any function with the scope* $(X_{i_1}, \cdots, X_{i_k})$, $r_i : d_{i_1} \times .. \times d_{i_k} \to \mathbb{R}$, *which denotes how much utility is*

assigned to each possible combination of values of the involved variables. Negative amounts mean costs. [1]

In a DCOP, each variable and constraint is owned by an agent. A simplifying assumption [Yokoo *et al.*, 1992] is that each agent controls a virtual agent for each one of the variables X_i that it owns. To simplify the notation, we use X_i to denote either the variable itself, or its (virtual) agent.

This is a multiagent instance of the *valued CSP* framework as defined in [Schiex *et al.*, 1995]. The goal is to find a complete instantiation $\mathcal{X}^*$ for the variables X_i that *maximizes* the sum of utilities of all individual relations.

We assume here only unary and binary relations. However, DPOP and PC-DPOP can easily extend to non-binary constraints ([Petcu *et al.*, 2006]).

2.1 Depth-First Search Trees (DFS)

PC-DPOP works on a DFS traversal of the problem graph.

Definition 2 (DFS tree) *A DFS arrangement of a graph G is a rooted tree with the same nodes and edges as G and the property that adjacent nodes from the original graph fall in the same branch of the tree (e.g. X_0 and X_{11} in Figure 1).*

Figure 1 shows an example DFS tree that we shall refer to in the rest of this paper. We distinguish between *tree edges*, shown as solid lines (e.g. $X_1 - X_2$), and *back edges*, shown as dashed lines (e.g. $X_{12} - X_8$, $X_4 - X_0$).

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

- **parent** P_i **/ children** C_i: *these are the obvious definitions ($P_4 = X_2$, $C_1 = \{X_2, X_5\}$).*
- **pseudo-parents** PP_i *are X_i's ancestors directly connected to X_i through back-edges ($PP_5 = \{X_0\}$).*
- **pseudo-children** PC_i *are X_i's descendants directly connected to X_i through back-edges ($PC_1 = \{X_4\}$).*
- Sep_i *is the* **separator** *of X_i: ancestors of X_i which are directly connected with X_i or with descendants of X_i (e.g. $Sep_9 = \{X_0, X_8\}$, $Sep_2 = \{X_0, X_1\}$). Removing the nodes in Sep_i completely disconnects the subtree rooted at X_i from the rest of the problem.*

2.2 Optimal Asynchronous Partial Overlay

Optimal Asynchronous Partial Overlay (OptAPO [Mailler and Lesser, 2005]) is a sound and optimal algorithm for solving DCOPs that uses dynamic, partial centralization (DPC). Conceptually, DPC is a technique that discovers difficult portions of a shared problem through trial and error and centralizes these sub-problems into a mediating agent in order to take advantage of a fast, centralized solver. Overall, the protocol exhibits an early, very parallel hill climbing behavior which progressively transitions into a more deliberate, controlled search for an optimal solution. In the limit, depending on the difficulty of the problem and the tightness of the interdependences between the variables, one or more agents may end up centralizing the entire problem in order to guarantee that an optimal solution has been found.

---

[1]Hard constraints (that explicitly forbid/enforce certain value combinations) can be simulated with soft constraints by assigning $-\infty$ to disallowed tuples, and 0 to allowed tuples. Maximizing utility thus avoids assigning such value combinations to variables.

2.3 DPOP: dynamic programming optimization

DPOP is a distributed version of the bucket elimination scheme from [Dechter, 2003], which works on a DFS. *DPOP* has 3 phases.

Phase 1 - a **DFS traversal** of the graph is done using a distributed DFS algorithm like in [Petcu *et al.*, 2006]. The outcome of this step is that all nodes consistently label each other as parent/child or pseudoparent/pseudochild, and edges are identified as tree/back edges. This can be achieved for any graph with a linear number of messages.

Phase 2 - **UTIL propagation** is a bottom-up process, which starts from the leaves and propagates upwards only through tree edges. The agents send *UTIL* messages to their parents. The subtree of a node X_i can influence the rest of the problem only through X_i's separator, Sep_i. Therefore, a message contains the optimal utility obtained in the subtree for each instantiation of Sep_i. Thus, messages are exponential in the separator size (bounded by the induced width).

Phase 3 - **VALUE propagation** top-down, initiated by the root, when phase 2 has finished. Each node determines its optimal value based on the messages from phase 2 and the *VALUE* message it has received from its parent. Then, it sends this value to its children through *VALUE* messages.

3 *PC-DPOP(k)* - partial centralization hybrid

To overcome the space problems of DPOP, we introduce the control parameter k that bounds the message dimensionality. This parameter should be chosen s.t. the available memory at each node and the capacity of its link with its parent is greater than d^k, where d is the maximal domain size.

As with DPOP, PC-DPOP also has 3 phases: a *DFS construction* phase, an *UTIL* phase, and a *VALUE* phase.

3.1 PC-DPOP - *UTIL* Phase

This is where the algorithm is different from DPOP. The propagation proceeds as in DPOP where possible, and reverts to partial centralization where the width exceeds k:

1. the *UTIL* propagation starts bottom-up exactly like in DPOP, and proceeds as long as the dimensionality of the outgoing *UTIL* messages is below the threshold k.

2. as soon as a node's outgoing *UTIL* message has more than k dimensions, centralization begins (see Section 3.1). The node does not compute its *UTIL* message, but sends to its parent a *Relation* message that contains the set of relations (arity at most k) that the node would have used as an input for computing the *UTIL* message. It annotates this message with its ID.

3. Upon receiving such a *Relation* message, a node tests to see if its separator is smaller than k. The separator is determined as the intersection between the nodes on the root path of the current node and the set of nodes encountered as dimensions in the messages received from its children.

   - If this is the case, the node becomes a cluster root, reconstructs the subproblem from the incoming *Relation* messages and then solves it
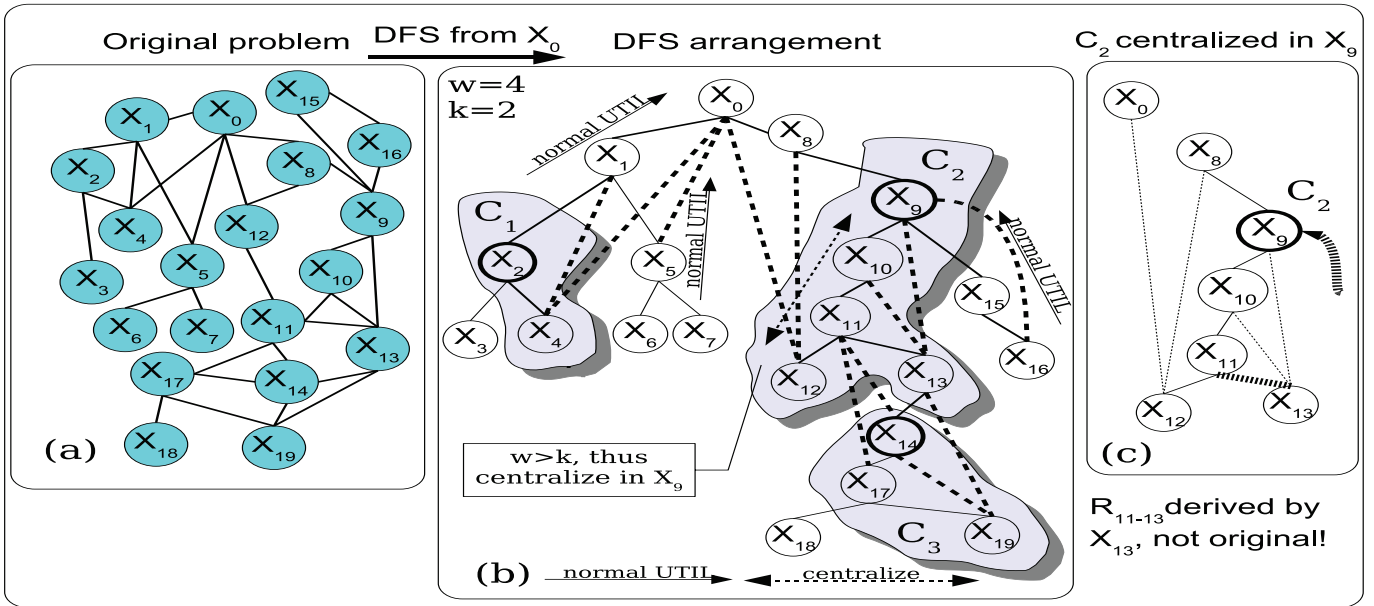
Figure 1: *A problem graph (a) and a DFS tree (b). In low-width areas the normal UTIL propagation is performed. In high width areas (shaded clusters C_1, C_2 and C_3 in (b)) bounded UTIL propagation is used. All messages are of size at most d^k. Clusters are centralized and solved in the cluster roots (the bold nodes X_2, X_9, X_{14}).*

(see Section 3.1). Then it continues the *UTIL* propagation as in DPOP.

Later on, during the VALUE phase, when the node receives the *VALUE* message from its parent, it retrieves the solution from its local cache and informs nodes in the cluster of their optimal values via VALUE messages.

- Otherwise, it cannot become cluster root and has to pass on to its parent all the relevant relations (the ones received from its children and its own), that it would otherwise use to compute its *UTIL* message. For details, see Section 3.1.

PC-DPOP - Centralization

This process occurs in high-width areas of the problem (e.g. the shaded areas in Figure 1). It is initiated by any node that cannot compute and send its *UTIL* messages because that would exceed the dimensionality limit imposed by k. Any such node produces the union of the relations and UTIL messages received from children, and its own relations with its parent/pseudoparents. All these relations can be sent to its parent in a *Relation* message, annotated by appending the node's ID to the path.

On one hand, this ensures the dimensionality limit k is observed, as no relation with arity larger than k is produced or sent over the network. On the other hand, it will allow the cluster root to reconstruct the subproblem that has to be centralized, and enable the use of structure sensitive algorithms like DPOP, AOBB, etc.

Alternatively, to save bandwidth, avoid overload on cluster root nodes, and also improve privacy, a node can selectively join subsets of these relations/*UTIL* messages, s.t. the dimensionality of each of the resulting relations is less than k.

The resulting set of relations is then packaged as a *Relation* message, annotated with the node's ID, and sent to the parent (see Section 3.1). This happens as follows:

1. node X_i receives all *UTIL/Relation* messages from its children, if any

2. X_i forms the union U_i of all relations in the *UTIL/Relation* messages and the relations it has with its parent and pseudoparents

3. X_i matches pairs of relations in U_i s.t. by joining them the resulting relation will have k dimensions or less (the dimensionality of the resulting relation is the union of the dimensions of the inputs). If the join was successful, remove both inputs from U_i, and add the result instead. Try until no more joins are possible between relations in U_i. This process is linear in the size of U_i.

4. The resulting U_i set is sent to X_i's parent in a *Relation* message

This process proceeds bottom-up until a node X_r with a separator smaller than k is reached. X_r declares itself a cluster root and executes the procedures from Section 3.1.

The result of this phase is that *minimal*, difficult (high-width) areas of the problem are identified in a distributed fashion. In these areas the algorithm reverts to partial centralization, by having nodes send to their parents not high dimensional *UTIL* messages, but lower arity *inputs* that could be used to generate those *UTIL* messages.

Subproblem reconstruction

Let us assume a cluster root node X_i has received a set of relations R_{C_i} from its children. Each relation $r_i \in R_{C_i}$ has a set of variables that it involves ($scope(r_i)$), and the path that

it traveled through until it reached X_i. X_i creates an internal copy of all the nodes found in the scopes of the relations received. Then, X_i reconstructs the subtree by placing each internal variable X_k in its position as follows:

- if X_k is an ancestor of X_i, then it is identified as such, because X_i knows the path from root to itself.

- if X_k is a child of X_i, then it is identified as such.

- otherwise, it is a descendant of X_i. In this case X_i deduces its place in the subtree by analyzing backwards the path recorded in the relations involving X_k. For example, in Figure 1, X_9 receives r_{12}^0, r_{12}^{11} and r_{12}^8, all annotated with the path $X_{12} - X_{11} - X_{10} - X_9$. X_{10} is already a child of X_9, so it is then easy to see that X_{11} is X_{10}'s child and X_{12} is X_{11}'s child. Thus, the whole branch $X_9 - X_{10} - X_{11} - X_{12}$ is reconstructed.

It is interesting to note that this technique allows for identifying different branches as well. Consider again X_9 that also receives r_{13}^9, r_{13}^{10} and r_{13}^{11}, all annotated with the path $X_{13} - X_{11} - X_{10} - X_9$. Following the same reasoning as before, X_9 can infer that X_{13} is also a child of X_{11}, like X_{12}. Therefore, X_{12} and X_{13} lie in different branches.

This makes it possible for a cluster node to reconstruct the subproblem *while preserving structural information*. This is important because it enables the use of high-performance optimization algorithms that also take advantage of problem structure. Examples include AOBB [Marinescu and Dechter, 2005], a centralized DPOP [Petcu and Faltings, 2005], etc.

Solving centralized subproblems

The centralized solving occurs in the roots of the clusters determined by the high-width areas. In the example of Figure 1, such a cluster is the shaded area containing $X_9, X_{10}, X_{11}, X_{12}, X_{13}$.

The root of the cluster (e.g. X_9) maintains a *cache table* that has as many locations as there are possible assignments for its separator (in this case $d^k = d^2$ locations). As a normal node in *DPOP*, the root also creates a table for the outgoing *UTIL* message, with as many dimensions as the size of the separator. Each location in the cache table directly corresponds to a location in the *UTIL* message that is associated with a certain instantiation of the separator. The cache table stores the best assignments of the variables in the centralized subproblem that correspond to each instantiation of the separator.

Then the process proceeds as follows:

- for each instantiation of Sep_i, the cluster root solves the corresponding centralized subproblem. The resulting utility and optimal solution are stored in the location of the *UTIL* message (cache table location, respectively) that correspond to this instantiation.

- when all Sep_i instantiations have been tried, the *UTIL* message for the parent contains the optimal utilities for each instantiation of the separator (exactly as in DPOP), and the cache table contains the corresponding solutions of the centralized subproblem that yield these optimal utilities.

- the cluster root sends its UTIL message to its parent, and the process continues just like in normal DPOP.

There are multiple possibilities for choosing an optimization algorithm for solving subproblems: a centralized version of DPOP [Petcu and Faltings, 2005], AOBB [Marinescu and Dechter, 2005], etc.

Algorithm 1: *PC-DPOP - partial centralization DPOP.*

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

***UTIL* propagation protocol**

1   wait for $UTIL/Relation$ messages from all children
2   **if** $Sep_i \leq k$ *(can send outgoing $UTIL_i^{P_i}$)* **then**
3   **if** *incoming are all UTIL (normal DPOP node)* **then**
      compute $UTIL_i^{P_i}$ as in DPOP and send it to P_i
4   **else** *(this means X_i is the root of a cluster)*
5   reconstruct subproblem from received relations
6   solve subproblem for each $s \in \langle Sep_i \rangle$ and store utility in $UTIL_i^{P_i}$ and solution in local cache
7   send $UTIL_i^{P_i}$ to P_i
8   **else** *(this means X_i is part of a cluster)*
9   Join subsets of incoming UTIL/Relation and relations with (p)parent with same dimension s.t. for each join, $dim(join) \leq k$
10   send joins to P_i

   ***VALUE* propagation**(X_i gets $Sep_i \leftarrow Sep_i^*$ from P_i)
11   **if** X_i *is cluster root* **then**
12   find in cache Sol^* that corresponds to Sep_i^*
13   assign self according to Sol^*
14   send Sol^* to nodes in my cluster via VALUE msgs
15   **else** continue VALUE phase as in DPOP

3.2 PC-DPOP - *VALUE* Phase

The labeling phase has determined the areas where bounded inference must be applied due to excessive width. We will describe in the following the processing to be done in these areas; outside of these, the original VALUE propagation from DPOP applies.

The VALUE message that the root X_i of a cluster receives from its parent contains the optimal assignment of all the variables in the separator Sep_i of X_i (and its cluster). Then X_i can simply retrieve from its cache table the optimal assignment corresponding to this particular instantiation of the separator. This assignment contains its own value, and the values of all the nodes in the cluster. X_i can thus inform all the nodes in the cluster what their optimal values are (via *VALUE* messages). Subsequently, the *VALUE* propagation proceeds as in DPOP.

3.3 PC-DPOP - Complexity

In low-width areas of the problem, PC-DPOP behaves exactly as DPOP: it generates a linear number of messages that are at most d^k in size. In areas where the width exceeds k, the clusters are formed.

Theorem 1 $PC - DPOP(k)$ *requires communication* $O(exp(k))$. *Memory requirements vary from* $O(exp(k))$ *to* $O(exp(w))$ *depending on the algorithm chosen for solving centralized subproblems (w is the width of the graph).*

PROOF. Section 3.1 shows that whenever the separator of a node is larger than k, that node is included in a cluster. It also shows that within a cluster, *UTIL* messages with more than k dimensions are never computed or stored, but their input components sent out instead. It can be shown recursively that these components have always less than k dimensions, which proves the first part of the claim.

Assuming that $w > k$, memory requirements are at least $O(exp(k))$. This can easily be seen in the roots of the clusters: they have to store the *UTIL* messages and the cache tables, both of which are $O(exp(Sep = k))$.

Within a cluster root, the least memory expensive algorithm would be a search algorithm (e.g. AOBB(1)) that uses linear memory. The exponential size of the cache table and UTIL message dominates this, so memory overall is $O(exp(k))$.

The most memory intensive option would be to use a centralized version of DPOP, that is proved to be exponential in the induced width of the subtree induced by the cluster. Overall, this means memory is exponential in the maximal width of any cluster, which is the overall induced width.

□

4 Experimental evaluation

We performed experiments on 3 different problem domains: distributed sensor networks (DSN), graph coloring[2] (GC), and meeting scheduling (MS). Our versions of OptAPO and PC-DPOP used different centralized solvers, so in the interest of fairness, we did not compare their runtimes. Instead, we compared the effectiveness of the centralization protocols themselves, using 2 metrics: communication required (see Figure 3), and amount of centralization (see Figure 2). Figures 2 and 3 present the results on the GC problems.

The bound k has to be at least as large as the maximal arity of the constraints in the problem; since these problems contain only binary constraints, we ran PC-DPOP(k) with k between 2 and the width of the problem. As expected, the larger the bound k, the less centralization occurs. However, message size and memory requirements increase.

DSN The DSN instances are very sparse, and the induced width is 2, so $PC - DPOP(k \geq 2)$ always runs as DPOP: no centralization, message size is $d^2 = 25$. In contrast, in OptAPO almost all agents centralize some part of the problem. Additionally, in the larger instances some agents centralize up to half the problem.

Meeting scheduling we generated a set of relatively large distributed meeting scheduling problems. The model is as in [Maheswaran *et al.*, 2004]. Briefly, an optimal schedule has to be found for a set of meetings between a set of agents. The problems were large: 10 to 100 agents, and 5 to 60 meetings, yielding large problems with 16 to 196 variables.

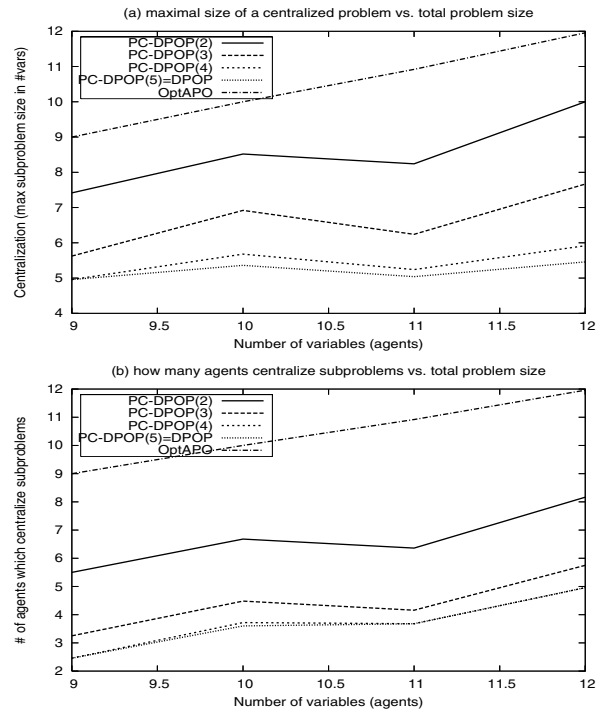[2]DSN and GC instances taken from [Maheswaran *et al.*, 2004].

Figure 2: Graph coloring: Centralization. (a) In OptAPO there is always an agent which centralizes all the problem. (b) In OptAPO all agents centralize some subproblem.
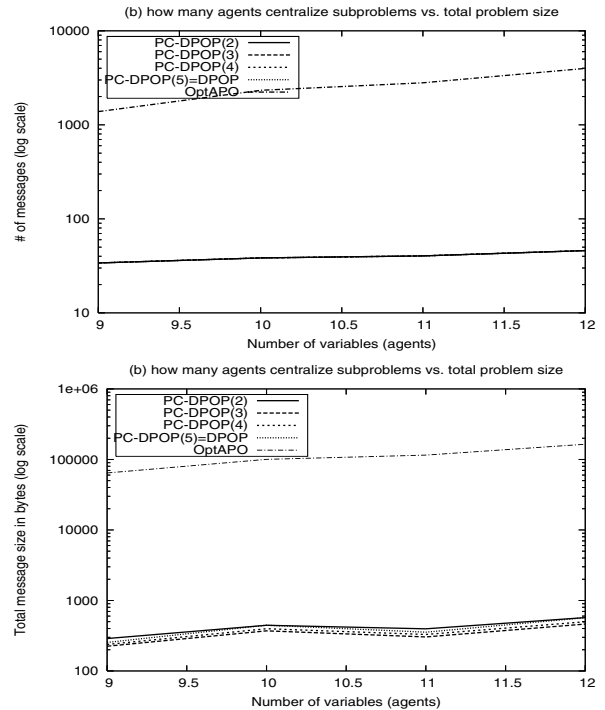

Figure 3: Graph coloring: message exchange. (a) All PC-DPOP variants use a linear # of messages. ; (b) Total information exchange (bytes) is much lower for PC-DPOPs.

The larger problems were also denser, therefore even more difficult (induced width from 2 to 5).

OptAPO managed to terminate successfully only on the smallest instances (16 variables), and timeout on all larger instances. We believe this is due to OptAPO's excessive centralization, which overloads its centralized solver. Indeed, OptAPO centralized almost all the problem in at least one node, consistent with [Davin and Modi, 2005].

In contrast, PC-DPOP managed to keep the centralized subproblems to a minimum, therefore successfully terminating on even the most difficult instances. *PC-DPOP(2)* (smallest memory usage) centralized at most 10% of the problem in a single node, and *PC-DPOP(4)* (maximal k) centralized at most 5% in a single node. *PC-DPOP(5)* is equivalent to DPOP on these problems (no centralization).

Overall, our results show that both OptAPO and PC-DPOP centralize more in dense problems; however, PC-DPOP's structure-based strategy performs much better.

5 Related Work

Tree clustering methods (e.g. [Kask *et al.*, 2005]) have been proposed for a long time for time-space tradeoffs. PC-DPOP uses the concept loosely and in many parts of the problem transparently. Specifically, in areas where the width is low, there is no clustering involved, the nodes following the regular DPOP protocols. In high-width areas, PC-DPOP creates clusters based on the context size of the outgoing UTIL messages and bounds the sizes of the clusters to a minimum using the specified separator size.

6 A Note on Privacy

[Maheswaran *et al.*, 2006] show that in some settings and according to some metrics (complete) centralization is not worse (privacy-wise) than some distributed algorithms.

Even though the nodes in a cluster send relations to the cluster root, these relations may very well be the result of aggregations, and **not** the original relations. For example, in Figure 1, X_{13} sends X_9 (via X_{11} and X_{10}) 3 relations: r_{13}^{11}, r_{13}^{10} and r_{13}^{9}. Notice that r_{13}^{11} that is sent to X_9 like this is not the real r_{13}^{11}, but the result of the aggregation resulting from the partial join performed with the UTIL message that X_{13} has received from X_{14}. Therefore, inferring true valuations may be impossible even in this scenario.

7 Conclusions and future work

We have presented an optimal, hybrid algorithm that uses a customizable message size and amount of memory. PC-DPOP allows for a priory, exact predictions about privacy loss, communication, memory and computational requirements on all nodes and links in the network.

The algorithm explores loose parts of the problem without any centralization (like DPOP), and only small, tightly connected clusters are centralized and solved by the respective cluster roots. This means that the privacy concerns associated with a centralized approach can be avoided in most parts of the problem. We will investigate more thoroughly the privacy loss of this approach in further work.

Experimental results show that PC-DPOP is particularly efficient for large optimization problems of low width. The intuition that dense problems tend to require more centralization is confirmed by experiments.

References

[Davin and Modi, 2005] John Davin and Pragnesh Jay Modi. 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*, pages 1057–1063, New York, NY, USA, 2005. ACM Press.

[Dechter, 2003] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Kask *et al.*, 2005] Kalev Kask, Rina Dechter, and Javier Larrosa. Unifying cluster-tree decompositions for automated reasoning in graphical models. *Artificial Intelligence*, 2005.

[Maheswaran *et al.*, 2004] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS-04*, 2004.

[Maheswaran *et al.*, 2006] R.T. Maheswaran, J.P. Pearce, E. Bowring, P. Varakantham, and M Tambe. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *JAAMAS*, 2006.

[Mailler and Lesser, 2004] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, 2004.

[Mailler and Lesser, 2005] Roger Mailler and Victor Lesser. Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research (JAIR)*, 2005. to appear.

[Marinescu and Dechter, 2005] Radu Marinescu and Rina Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, Edinburgh, Scotland, Aug 2005.

[Modi *et al.*, 2003] P. J. Modi, W. M. Shen, and M. Tambe. An asynchronous complete method for distributed constraint optimization. In *Proc. AAMAS*, 2003.

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

[Petcu *et al.*, 2006] Adrian Petcu, Boi Faltings, and David Parkes. M-DPOP: Faithful Distributed Implementation of Efficient Social Choice Problems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-06)*, Hakodate, Japan, May 2006.

[Schiex *et al.*, 1995] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI-95*, Montreal, Canada, 1995.

[Yokoo *et al.*, 1992] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.