Appears in the Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-07

MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization

Adrian Petcu and Boi Faltings
Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne (Switzerland)
email: {adrian.petcu, boi.faltings}@epfl.ch

Abstract

In distributed combinatorial optimization problems, dynamic programming algorithms like DPOP ([Petcu and Faltings, 2005]) require only a linear number of messages, thus generating low communication overheads. However, DPOP's memory requirements are exponential in the *induced width* of the constraint graph, which may be prohibitive for problems with large width.

We present MB-DPOP, a new hybrid algorithm that can operate with bounded memory. In areas of low width, MB-DPOP operates like standard DPOP (linear number of messages). Areas of high width are explored with bounded propagations using the idea of *cycle-cuts* [Dechter, 2003].

We introduce novel DFS-based mechanisms for determining the cycle-cutset, and for grouping cycle-cut nodes into clusters. We use caching between clusters to reduce the complexity to exponential in the largest number of cycle cuts in a single cluster.

We compare MB-DPOP with ADOPT [Modi *et al.*, 2005], the current state of the art in distributed search with bounded memory. MB-DPOP consistently outperforms ADOPT on 3 problem domains, with respect to 3 metrics, providing speedups of up to 5 orders of magnitude.

1 Introduction

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, problems are sometimes naturally distributed, so Distributed Constraint Satisfaction (DisCSP) was formalized in [Yokoo *et al.*, 1992]. These problems are divided between a set of agents, which have to communicate among themselves to solve them. Complete algorithms like ADOPT, DPOP and OptAPO have been introduced.

ADOPT [Modi *et al.*, 2005] is a backtracking based bound propagation mechanism. It operates completely decentralized, and asynchronously. It requires polynomial memory, but it may produce a very large number of small messages, resulting in large communication overheads.

OptAPO [Mailler and 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 possible that several mediators solve overlapping problems, thus needlessly duplicating effort. This has been shown in [Petcu and Faltings, 2006] to cause scalability problems for OptAPO, especially on dense problems.

DPOP [Petcu and Faltings, 2005] is a complete algorithm based on dynamic programming which generates only a linear number of messages. However, *DPOP* is time and space exponential in the *induced width* of the problem. Therefore, for problems with high induced width, the messages generated in high-width areas get large, therefore requiring exponential communication and memory.

This paper introduces MB-DPOP, a new hybrid algorithm that is controlled by a parameter k which characterizes the amount of available memory. MB-DPOP is a tradeoff of the linear number of messages of DPOP for polynomial memory.

The rest of this paper is structured as follows: Section 2 introduces the distributed optimization problem. Section 3 presents the *DPOP* algorithm, which is extended to the MB-DPOP(k) hybrid in Section 4. Section 5 shows the efficiency of this approach with experiments on distributed meeting scheduling problems. Section 6 discusses related work, and Section 7 concludes.

2 Definitions and Notation

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

- $\mathcal{X} = \{X_1, ..., X_n\}$ *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* $(X_{i_1}, \cdots, X_{i_k})$*,* $r_i : d_{i_1} \times .. \times d_{i_k} \rightarrow \mathbb{R}$*, which denotes how much utility is assigned to each possible combination of values of the involved variables. Negative utilities mean cost.* [1]

[1] Hard constraints (that explicitly forbid/enforce certain value

DCOPs are multiagent instances of the *valued CSP* framework, where each variable and constraint is owned by an agent. The goal is to find a complete instantiation $\mathcal{X}^*$ for the variables X_i that *maximizes* the sum of utilities of individual relations.

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.We also assume here only unary and binary relations [2]

2.1 Depth-First Search Trees (DFS)

MB-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_{12} in Figure 1).*

DFS trees have already been investigated as a means to boost search [Freuder, 1985; Dechter, 2003]. Due to the relative independence of nodes lying in different branches of the DFS tree, it is possible to perform search in parallel on independent branches, and then combine the results.

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. $12 - 8$, $4 - 0$).

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

- **parent** P_i **/ children** C_i*: these are the obvious definitions (e.g. $P_4 = X_2$, $C_0 = \{X_1, X_8\}$).*
- **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. $Sep_{15} = \{X_9\}$ and $Sep_{11} = \{X_0, X_8, X_9, X_{10}\}$). Removing the nodes in Sep_i completely disconnects the subtree rooted at X_i from the rest of the problem.*

3 DPOP: dynamic programming optimization

The basic utility propagation scheme *DPOP* has been introduced in [Petcu and Faltings, 2005]. *DPOP* is an instance of the general bucket elimination scheme from [Dechter, 2003], which is adapted for the distributed case, and uses a DFS traversal of the problem graph as an ordering.

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], which works for any graph requiring a linear number of messages. The outcome is that all nodes consistently label each other as parent/child or pseudoparent/pseudochild, and edges are identified as tree/back edges. The DFS tree serves as a communication structure for the other 2 phases of the algorithm: UTIL messages (phase 2) travel bottom-up, and VALUE messages (phase 3) travel top down, only via tree-edges.

Phase 2 - **UTIL propagation**: the agents (starting from the leaves) 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 size-exponential in the separator size (which is in turn 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 computation from phase 2 and the *VALUE* message it has received from its parent. Then, it sends this value to its children through *VALUE* messages.

It has been proved in [Petcu and Faltings, 2005] that *DPOP* produces a linear number of messages. Its complexity lies in the size of the *UTIL* messages: the largest one is space-exponential in the width of the DFS ordering used.

4 *MB-DPOP(k)* - bounded inference hybrid

We introduce the control parameter k which specifies the maximal amount of inference (maximal message dimensionality). This parameter is chosen s.t. the available memory at each node is greater than d^k, (d is the domain size).

The algorithm identifies subgraphs of the problem that have width higher than k, where it is not possible to perform full inference as in DPOP. Each of these areas (clusters) are bounded at the top by the lowest node in the tree that has separator of size k or less. We call these nodes *cluster roots* (CR).

In such areas, cycle-cut nodes (CC) are identified such that once these are removed, the remaining problem has width k or less. Subsequently, in each area all combinations of values of the CC nodes are explored using k-bounded DPOP utility propagation. Results are cached ([Darwiche, 2001]) by the respective cluster roots and then integrated as messages into the overall DPOP-type propagation.

If $k \geq w$ (w is the induced width), full inference is done throughout the problem; MB-DPOP is equivalent with DPOP.

If $k = 1$, only linear messages are used, and a full cycle cutset is determined. MB-DPOP is similar to the AND/OR cycle cutset scheme from [Mateescu and Dechter, 2005].

If $k < w$, $MB - DPOP(k)$ performs full inference in areas of width lower than k, and bounded inference in areas of width higher than k.

In the following we explain how to determine high-width areas and the respective cycle-cuts (Section 4.1) and what changes we make to the *UTIL* and *VALUE* phases (Section 4.2 and Section 4.3). The algorithm is described in Algorithm 1.

4.1 MB-DPOP - Labeling Phase

This is an intermediate phase between the DFS and UTIL phases. It has the goal to delimit high-width areas, and

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] However, all *-DPOP algorithms easily extend to non-binary constraints, as in [Petcu and Faltings, 2005; Petcu *et al.*, 2006].

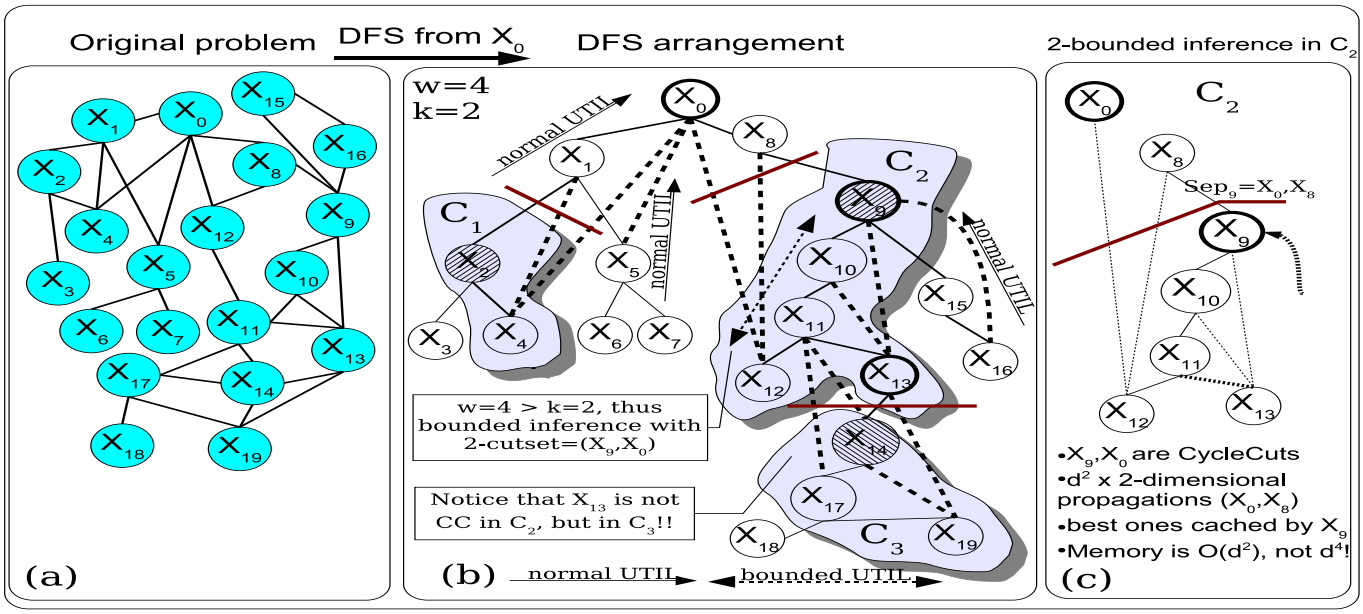

Figure 1: *A problem graph (a) and a DFS tree (b). In low-width areas the normal UTIL propagation is performed. In high width areas (shaded clusters C_1, C_2 and C_3 in (b)) bounded UTIL propagation is used. All messages are of size at most d^k. Cycle-cut nodes are in bold (X_0, X_9, X_{13}), and cluster roots are shaded (X_2, X_9, X_{14}). In (c) we show a 2-bounded propagation.*

identify CC nodes in these areas. We emphasize that this process is described as a separate phase only for the sake of clarity; these results can be derived with small modifications from either of the original DFS or UTIL phases.

Labeling works bottom-up like the UTIL phase. Each node X_i waits for $LABEL_j^i$ messages from its children X_j, computes its own label $LABEL_i^{P_i}$, and sends it to its parent P_i.

A $LABEL_i^{P_i}$ message is composed of 2 parts: the *separator* Sep_i of the node, and a list CC_i of CC nodes.

Each node X_i can easily determine its separator Sep_i as the union of: (a) separators received from its children, and (b) its parent and pseudoparents, minus itself (see Definition 3). Formally, $Sep_i = \cup_{X_j \in C_i} Sep_j \cup P_i \cup PP_i \setminus X_i$.

The second part of the *LABEL* message is the list CC_i of the cycle-cut nodes to send to the parent. Each node computes this list through a heuristic function based on the separator of the node, and on the lists of cycle-cuts received from the children (see next section).

Heuristic labeling of nodes as CC

Let $label(Sep_i, CClists, k)$ be a heuristic function that takes as input the separator of a node, the lists of cycle-cuts received from the children, and an integer k, and it returns another list of cycle cutset nodes.

It builds the set $N_i = Sep_i \setminus \{\cup CClists\}$: these are nodes in X_i's separator that are not marked as CC nodes by X_i's children. If $|N_i| > k$ (too many nodes not marked as CC), then it uses any mechanism to select from N_i a set CC_{new} of $|N_i| - k$ nodes, that will be labeled as CC nodes. The function returns the set of nodes $CC_i = \cup CClists \cup CC_{new}$.

If the separator Sep_i of X_i contains more than k nodes, then this ensures that enough of them will be labeled as cycle-cuts, either by the children of X_i or by X_i itself. If $|Sep_i| \leq k$, the function simply returns an empty list.

Mechanism 1: highest nodes as CC The nodes in N_i are sorted according to their tree-depth (known from the DFS phase). Then, the highest $|N_i| - k$ nodes are marked as CC.

For example, in Fig. 1, let $k = 2$. Then, $Sep_{12} = \{X_0, X_8, X_{11}\}$, $CClists_{12} = \emptyset \Rightarrow N_{12} = Sep_{12} \Rightarrow CC_{12} = \{X_0\}$ (X_0 is the highest among X_0, X_8, X_{11})

Mechanism 2: lowest nodes as CC This is the inverse of Mechanism 1: the lowest $|N_i| - k$ nodes are marked as CC.

For example, in Fig. 1, let $k = 2$. Then, $Sep_{12} = \{X_0, X_8, X_{11}\}$, $CClists_{12} = \emptyset \Rightarrow N_{12} = Sep_{12} \Rightarrow CC_{12} = \{X_{11}\}$ (X_{11} is the lowest among X_0, X_8, X_{11})

4.2 MB-DPOP - *UTIL* Phase

The labeling phase (Section 4.1) has determined the areas where the width is higher than k, and the corresponding CC nodes. We describe in the following how to perform bounded-memory exploration in these areas; anywhere else, the original UTIL propagation from DPOP applies.

Memory-Bounded UTIL Propagation

The labeling phase has identified the cluster root nodes for each high-width area, and furthermore, the roots have received the lists of cycle-cutset nodes, as designated by their children.

Let X_i be the root of such a cluster. Just like in DPOP, X_i creates a $UTIL_i^{P_i}$ table that stores the best utilities its subtree can achieve for each combination of values of the variables in Sep_i. X_i's children X_j that have separators smaller than k

Algorithm 1: *MB-DPOP - memory bounded DPOP.*

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

Labeling protocol:

wait for all $LABEL_j^i$ msgs from children
if $|Sep_i| \leq k$ **then**
  if $\cup CClists \neq \emptyset$ **then** label self as CR
  else label self as normal
  $CC_i \leftarrow \emptyset$
else
  let $N = Sep_i \setminus \cup CClists$
  select a set CC_{new} of $|N| - k$ nodes from N
  return $CC_i = CC_{new} \cup CClists$
send $LABEL_i^{P_i} = [Sep_i, CC_i]$ to P_i

***UTIL* propagation protocol**

wait for $UTIL_k^i$ messages from all children $X_k \in C(i)$
if $X_i =$ *normal node* **then** do UTIL / VALUE as DPOP
else
  do propagations for all instantiation of $CClists$
  if X_i *is cluster root* **then**
    update UTIL and CACHE for each propagation
    when propagations finish, send UTIL to parent

***VALUE* propagation**(X_i receives Sep_i^* from P_i)

if X_i *is cluster root* **then**
  find in cache the CC^* corresponding to Sep_i^*
  assign self according to cached value
  send CC^* to nodes in CC via VALUE messages
else
  perform last UTIL with CC nodes assigned to CC^*
  assign self accordingly
Send *VALUE*$(X_i \leftarrow v_i^*)$ to all $C(i)$ and $PC(i)$

($|Sep_j| \leq k$) send X_i normal $UTIL_j^i$ messages, as in DPOP; X_i waits for these messages, and stores them.

For the children X_j that have a larger separator ($|Sep_j| > k$), X_i creates a $Cache$ table with one entry $Cache(sep_i)$ that corresponds to each particular instantiation of the separator, $sep_i \in \langle Sep_i \rangle$ (size of the $Cache$ table is thus $O(exp(|Sep_i|))$).

X_i then starts exploring through k-bounded propagation all its subtrees that have sent non-empty $CClists$. It does this by cycling through all instantiations of the CC variables in the cluster. Each one is sent down to its children via a *context* message. Children propagate the context messages again to their children that have sent non-empty $CClists$. This propagates only to nodes that have separators larger than k, down to the lowest nodes in the cluster that have separators larger than k.

The leaves of the cluster then start bounded propagation, with the nodes specified in the context message instantiated to their current value. These propagation are guaranteed to involve k dimensions or less, and they proceed as in normal DPOP, until they reach X_i. X_i then updates the best utility values found so far for each $sep_i \in \langle Sep_i \rangle$, and also updates the cache table with the current instantiation of the CC nodes in case a better utility was found.

When all the instantiations are explored, X_i simply sends to its parent the updated $UTIL_i^{P_i}$ table that now contains the best utilities of X_i's subtree for all instantiations of variables in Sep_i, exactly as in DPOP. P_i then continues propagation as in normal DPOP; all the complexity of X_i's processing is transparent to it.

Example In Figure 1, let $k = 2$; then $C_2 = \{X_9, X_{10}, X_{11}, X_{12}, X_{13}\}$ is an area of width higher than 2. X_9 is the root of C_2, as the first node (lowest in the tree) that has $Sep_i \leq k$. Using the Mechanism 1 for selecting CC nodes, we have X_9, X_0 as CC in C_2. X_9 cycles through all the instantiations $\langle X_9, X_0 \rangle$, and sends its child(ren) context messages of the form $X_9 = a, X_0 = b$. These messages travel to X_{10}, X_{11}, X_{12} and X_{13} (no other nodes in the cluster have sent non-empty $CClists$). Upon receiving the context messages, X_{12} and X_{13} start 2-bounded *UTIL* propagation (X_{12} with X_{11} and X_8 as dimensions, and X_{13} with X_{11} and X_{10} as dimensions).

It is easy to see that using this scheme, the memory requirements are still $O(exp(k))$ for the propagation, and $O(exp(|Sep|))$ for the cache tables. Since $|Sep| \leq k$ (see section 4.1), overall we observe the memory limit $O(exp(k))$.

4.3 MB-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). X_i retrieves 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 CC nodes in the cluster. X_i informs all the CC nodes in the cluster what their optimal values are (via *VALUE* messages).

As the non-CC nodes in the cluster could not have cached their optimal values for all instantiations of the CC nodes, it follows that a final *UTIL* propagation is required in order to re-derive the utilities that correspond to the particular instantiation of the CC nodes that was determined to be optimal. However, this is not an expensive process, since it is a single propagation, with dimensionality bounded by k (the CC nodes are instantiated now). Thus, it requires only a linear number of messages that are at most $exp(k)$ in size.

Subsequently, outside the clusters, the *VALUE* propagation proceeds as in DPOP.

4.4 MB-DPOP(k) - Complexity

In low-width areas of the problem, MB-DPOP behaves exactly as DPOP: it generates a linear number of messages that are at most d^k in size. Clusters are formed where the width exceeds k. Let T be such a cluster, let $|T|$ be the number of nodes in the cluster, and let $|CC(T)|$ be the number of cycle cut nodes in that cluster. MB-DPOP executes $d^{|CC(T)|}$ k-bounded propagation in that cluster, each

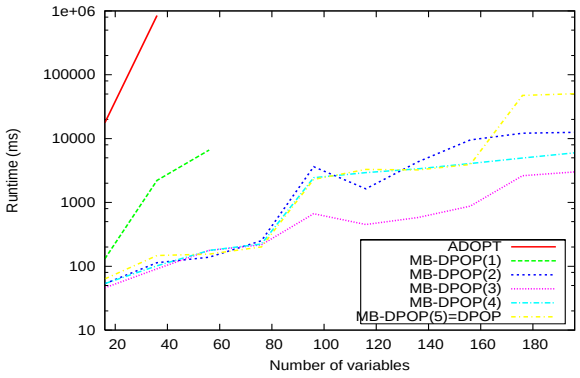

Figure 2: MB-DPOP(k) vs ADOPT - runtime

requiring $|T| - 1$ messages. The size of these messages is bounded by d^k.

Therefore, it is easy to see that the overall time/message complexity is $O(exp(|CC(T_{max})|))$ where T_{max} is the cluster that has the maximal number of CC nodes.

Memory requirements are $O(exp(k))$ by design.

5 Experimental evaluation

We performed experiments on 3 different problem domains[3]: distributed sensor networks (DSN), graph coloring (GC), and meeting scheduling (MS).

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

The experimental results are presented in Figure 3 (number of messages and total information exchanged - sum of all message sizes, in bytes), and Figure 2 (runtime in milliseconds)[4]. Please notice the logarithmic scale! ADOPT did not scale on these problems, and we had to cut its execution after a threshold of 2 hours or 5 million messages. The largest problems that ADOPT could solve had 20 agents (36 variables).

We also executed MB-DPOP with increasing bounds k. As expected, the larger the bound k, the less nodes will be designated as CC, and the less messages will be required[5]. However, message size and memory requirements increase.

It is interesting to note that even $MB-DPOP(1)$ (which uses linear-size messages, just like ADOPT) performs much better than ADOPT: it can solve larger problems, with a smaller number of messages. For example, for the largest problems ADOPT could solve, $MB-DPOP(1)$ produced improvements of 3 orders of magnitude. $MB-DPOP(2)$

[3]All experiments are run on a P4 machine with 1GB RAM

[4]Each data point is an average over 10 instances

[5]Mechanism 1 for CC selection was used.

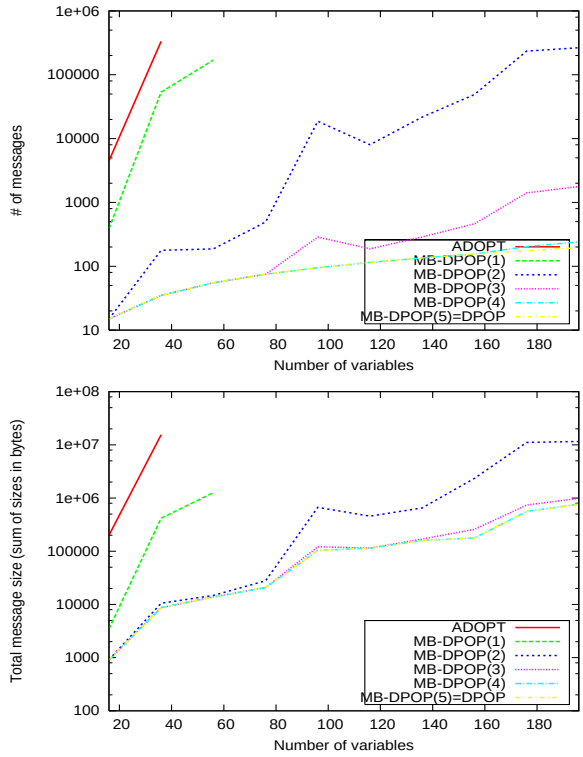

Figure 3: MB-DPOP(k) vs ADOPT - message exchange

improved over ADOPT on some instances for 5 orders of magnitude.

Also, notice that even though $MB-DPOP(k>1)$ sends larger messages than ADOPT, overall, it exchanges much less information (Fig 3, low). We believe there are 2 reasons for this: ADOPT sends many more messages, and because of its asynchrony, it has to attach the full context to all of them (which produces extreme overheads).

The DSN and GC problems are the same as the ones used in [Maheswaran *et al.*, 2004]. For lack of space, we do not present detailed results here, but they align well with the meeting scheduling ones.

DSN The DSN instances are very sparse, and the induced width is 2, so $MB-DPOP(k \geq 2)$ always runs with a linear number of messages (from 100 to 200 messages) of size at most 25. Runtime varies from 52 ms to 2700 ms. In contrast, ADOPT sends anywhere from 6000 to 40.000 messages, and requires from 6.5 sec to 108 sec to solve the problems. Overall, these problems were very easy for MB-DPOP, and we have experienced around 2 orders of magnitude improvements in terms of CPU time and number of messages.

Graph Coloring The GC instances are also small (9 to 12 variables), but they are more tightly connected, and are even more challenging for ADOPT. ADOPT terminated on all of them, but required up to 1 hour computation time, and 4.8 million messages for a problem with 12 variables.

All three domains showed strong performance improvements of MB-DPOP over the previous state of

the art algorithm, ADOPT. On these problems, we noticed up to 5 orders of magnitude less computation time, number of messages, and overall communication.

6 Related Work

The w-cutset idea was introduced in [Rish and Dechter, 2000]. A *w-cutset* is a set CC of nodes that once removed, leave a problem of induced width *w* or less. One can perform search on the w-cutset, and exact inference on the rest of the nodes. The scheme is thus time exponential in $d^{|CC|}$ and space exponential in k.

If separators smaller than k exist, $MB - DPOP(k)$ isolates the cutset nodes into different clusters, and thus it is time exponential in $|CC(T_{max})|$ as opposed to exponential in $|CC|$. Since $|CC(T_{max})| \leq |CC|$, $MB-DPOP(w)$ can produce exponential speedups over the w-cutset scheme.

AND/OR w-cutset is an extension of the w-cutset idea, introduced in [Mateescu and Dechter, 2005]. The w-cutset nodes are identified and then arranged as a *start-pseudotree*. The lower parts of the pseudotree are areas of width bounded by w. Then AND/OR search is performed on the w-cutset nodes, and inference on the lower parts of bounded width. The algorithm is time exponential in the depth of the start pseudotree, and space exponential in w.

It is unclear how to apply their technique to a distributed setting, particularly as far as the identification of the w-cutset nodes and their arrangement as a start pseudotree are concerned. MB-DPOP solves this problem elegantly, by using the DFS tree to easily delimit clusters and identify w-cutsets. Furthermore, the identified w-cutsets are already placed in a DFS structure.

That aside, when operating on the same DFS tree, MB-DPOP is superior to the AND/OR w-cutset scheme without caching on the start pseudotree. The reason is that MB-DPOP can exploit situations where cutset nodes along the same branch can be grouped into different clusters. Thus MB-DPOP's complexity is exponential in the largest number of CC nodes in a single cluster, whereas AND/OR w-cutset is exponential in the total number of CC nodes along that branch. MB-DPOP has the same asymptotic complexity as the AND/OR w-cutset with w-bounded caching.

Finally, tree clustering methods (e.g. [Kask *et al.*, 2005]) have been proposed for time-space tradeoffs. MB-DPOP uses the concept loosely, only in high-width parts of the problem. For a given DFS tree, optimal clusters are identified based on the bound k and on node separator size.

7 Conclusions and future work

We have presented a hybrid algorithm that uses a customizable amount of memory and guarantees optimality. The algorithm uses cycle cuts to guarantee memory-boundedness and caching between clusters to reduce the complexity. The algorithm is particularly efficient on loose problems, where most areas are explored with a linear number of messages (like in DPOP), and only small, tightly connected components are explored using the less efficient bounded inference. This means that the large overheads associated with the sequential exploration can be avoided in most parts of the problem.

Experimental results on three problem domains show that this approach gives good results for low width, practically sized optimization problems. MB-DPOP consistently outperforms the previous state of the art in DCOP (ADOPT) with respect to 3 metrics. In our experiments, we have observed speedups of up to 5 orders of magnitude.

References

[Darwiche, 2001] Adnan Darwiche. Recursive conditioning. *Artif. Intell.*, 126(1-2):5–41, 2001.

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

[Freuder, 1985] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(14):755–761, 1985.

[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.

[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.

[Mateescu and Dechter, 2005] Robert Mateescu and Rina Dechter. AND/OR cutset conditioning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, Edinburgh, Scotland, Aug 2005.

[Modi *et al.*, 2005] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AI Journal*, 161:149–180, 2005.

[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 and Faltings, 2006] Adrian Petcu and Boi Faltings. 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, August 2006.

[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.

[Rish and Dechter, 2000] Irina Rish and Rina Dechter. Resolution versus search: Two strategies for SAT. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000.

[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.