Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

H-DPOP: Using Hard Constraints for Search Space Pruning in DCOP

Akshat Kumar *
akshat@cs.umass.edu
University of Massachusetts
Amherst, MA 01002

Adrian Petcu † and **Boi Faltings**
{adrian.petcu, boi.falting}@epfl.ch
Ecole Polytechnique Federale de Lausanne
Lausane, Switzerland

Abstract

In distributed constraint optimization problems, dynamic programming methods have been recently proposed (e.g. DPOP). In dynamic programming many valuations are grouped together in fewer messages, which produce much less networking overhead than search. Nevertheless, these messages are exponential in size. The basic DPOP always communicates all possible assignments, even when some of them may be inconsistent due to hard constraints. Many real problems contain hard constraints that significantly reduce the space of feasible assignments. This paper introduces H-DPOP, a hybrid algorithm that is based on DPOP, which uses Constraint Decision Diagrams (CDD) to rule out infeasible assignments, and thus compactly represent UTIL messages. Experimental results show that H-DPOP requires several orders of magnitude less memory than DPOP, especially for dense and tightly-constrained problems.

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 (Yokoo et al. 1998). Here, the problem is divided between a set of agents, which have to communicate among themselves to solve it. Many practical problems require the *optimization* of some performance criteria like cost, so the *Distributed Constraint Optimization Problems* (DCOP) has been introduced (Modi et al. 2005).

Several search algorithms (Yokoo et al. 1992; Meisels and Zivan 2003; Modi et al. 2005) have been proposed for DCOP. As in centralized CSP, distributed search algorithms have the advantage that they can operate with low memory requirements, and that they can prune the search space using various consistency techniques, as well as the branch-and-bound principle. All these algorithms are also asynchronous, which can sometimes offer the advantage of better dealing with message delays or message loss, or anytime behavior. Nevertheless, they all suffer from large networking overheads caused by sending an exponential number of small packets, and large algorithmic overheads due to the obligation of attaching full context information to each message because of asynchrony (Petcu and Faltings 2005; Faltings 2006).

Alternatively, (Petcu and Faltings 2005) propose DPOP, a dynamic programming algorithm for DCOP. DPOP adapts the bucket elimination principle (Dechter 2003) to a distributed setting, and works on a DFS traversal of the constraint graph. In DPOP, many assignments and their corresponding costs/utilities are grouped together in larger messages, which avoids the overhead of many small messages. In a problem with n variables, DPOP uses exactly $2(n-1)$ messages, which makes it much more scalable than search in real distributed environments (Faltings 2006; Petcu and Faltings 2005). However, some messages in the bottom-up dynamic programming step may be large, as the largest message is space exponential in the treewidth of the problem, and dynamic programming always propagates all assignments. This may be wasteful, as the messages may contain assignments that are actually inconsistent, and thus do not participate in the optimal solution.

In this paper, we consider how to leverage the hard constraints that may exist in the problem in a dynamic programming framework, so that only feasible partial assignments are computed, transmitted and stored. To this end, we encode combinations of assignments using *constrained decision diagrams*(CDDs) (Cheng and Yap 2005). CDDs eliminate all inconsistent assignments and only include costs or utilities for value combinations that are consistent. The resulting algorithm H-DPOP thus combines the pruning capabilities of search algorithms, and low communication overheads of dynamic programming algorithms.

Background

This section provides a brief description of DPOP and Constraint decision diagrams.

*Kumar was supported by the Air Force Office of Scientific Research under Grant No. FA9550-05-1-0254.

†Petcu was supported by the Swiss National Science Foundation under contract 200020-111862.

Preliminaries

A constraint network is described by a set of variables with finite domains and a set of functions defined over these variables. Variables or sets of variables are denoted by uppercase letters (X, Y, S) and the values of variables are denoted by lowercase letters (x, y, z). An assignment $(X_1 = x_1, \ldots, X_n = x_n)$ is denoted by $x = (x_1, \ldots, x_n)$. For a set of variables S, D_S represents the cartesian product of the domains of variables in S. If $X = X_1, \ldots, X_n$ and $S \subseteq X$, x_S denotes the projection of $x = (x_1, \ldots, x_n)$ over S. Functions are represented by letters e, f, g etc. and the scope (set of arguments) of a function f by $scope(f)$. Our terminology is partly based on (Dechter 2003; Kask et al. 2005).

Definition 1 (DCOP) *A discrete* distributed constraint optimization problem *(DCOP) is a tuple* $< X, D, F, \otimes >$ *where:* $X = \{X_1, \ldots, X_n\}$ *is a set of variables.* $D = \{D_{X_1}, \ldots, D_{X_n}\}$ *is a set of finite domains.* $F = \{f_1, \ldots, f_r\}$ *is a set of real valued functions over subsets of* X. $\otimes$ *is a combination operator* $(\otimes \in \{\prod, \sum\})$. *In a DCOP, 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* $\sum_{i=1}^{r} f_i$.

Hard Constraints (e.g. capacity constraints, all-diff constraints, sum constraints, inequality, etc) are represented as bi-valued functions with allowed tuples being assigned 0, and disallowed tuples being assigned $-\infty$. This formulation allows any utility maximization algorithm to avoid infeasible assignments and find the optimal solution. However, using this approach naively in an algorithm like DPOP does not take advantage of the search space pruning power of hard constraints. We propose a new algorithm *H-DPOP* which exploits hard constraints efficiently for search space reduction.

The next section provides some background on DPOP and its different phases.

DPOP: Dynamic Programming OPtimization

Definition 2 *The primal graph of a constraint network is an undirected graph that has variables as the vertices and an edge connects any two variables that appear in the scope of the same constraint function.*

We consider a distributed primal graph where each variable and constraint is owned by an agent.

Definition 3 (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.*

DPOP works on the DFS traversal of the primal graph for the given constraint network. 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. We use in the following several notations, and we exemplify on the DFS from Figure 1(a): $P(X_i)$ is the parent of the node X_i (e.g. $P(X_4) = X_3$), $C(X_i)$ are X_i's children ($C(X_3) = \{X_4, X_5\}$), $PP(X_i)$ are the pseudo parents of X_i ($PP(X_4) = \{X_1, X_2\}$) and Sep_i is the separator of X_i ($Sep_3 = \{X_2, X_1\}$).

Algorithm 1: *Phase 2 of DPOP*

Phase 2: Util Message propagation

1 $U_{X_i}^{PP} \leftarrow ComputeUtil(X_i \cup P(X_i), PP(X_i))$
2 Receive all U_i util messages from $Children(X_i)$
3 $U_{X_i}^{Join} \leftarrow Join(U_{X_i}^{PP}, Util_1, \ldots, U_{|Children(X_i)|})$
4 $U_{X_i} \leftarrow Project(X_i, U_{X_i}^{Join})$
5 send U_{X_i} to $P(X_i)$

$Phase\ 2\ finishes$

DPOP has three phases:

Phase 1 - a *DFS traversal* of the graph is generated using any distributed algorithm. The outcome of this protocol 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 thus obtained serves as a communication structure for the other 2 phases of the algorithm.

Phase 2 - *UTIL propagation*: The agents (starting from the leaves) send *UTIL* messages to their parents.

Definition 4 (UTIL message) *The UTIL message sent by agent* X_i *to agent* X_j *is a multidimensional matrix, with one dimension for each variable present in* Sep_i. $dim(UTIL_i^j)$ *is the set of individual variables in the message. Note that always* $X_j \in dim(UTIL_i^j)$.

Algorithm 1 shows this phase. Leaves start first by computing UTIL messages consisting of their relations with parent/pseudo parents (line 1). Each leaf agent projects its variable out, and sends the resulting message to its parent. Subsequently, all nodes compute the utility of their relations with parent/pseudo parents (line 1). Then they wait for all messages from their children (lines 2) and join them with the utility computed in line 1 (line 3). After this step they project themselves out from the join (line 4) and send the resulting message to their parent (line 5). The operations $join$ and $project$ are defined in definition 5, 6.

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 (this step is similar to DPOP).

Definition 5 (Join) *Given a set of functions* $h_1, \ldots, h_k$ *defined over the sets* $S_1, \ldots, S_k$, *the Join operator* $(\sum_j h_j)$ *is defined over* $U = \bigcup_j S_j$ *such that for every* $u \in D_U$: $(\sum_j h_j)(u) = \sum_j h_j(u_{S_j})$.

The above definition describes the join of k $UTIL$ messages. The sets S_i correspond to respective $dim(UTIL_i)$. Dimensions of $UTIL^{Joined}$ are given by the union set U. The function $h_j(u_{S_j})$ corresponds to the function $util_j$

which gives the utility for assignment u_{S_j} in the message $UTIL_j$.

Definition 6 (Projection) *Given is a function h with the scope being a subset of variables S. For any variable $X \in S$, we define the Projection operator (h_X^{max}) over $U = S - X$. For every $U = u$ and the extended tuple (u, x) where x is an assignment to X: $(h_X^{max})(u) = max_x h(u, x)$*

The function h gives the utility for any assignment r in a util message $\mathcal{U}$ with $S = dim(\mathcal{U})$. The variable to be projected is X. The util message after projection has dimensions as U.

The next section introduces constraint decision diagrams (CDDs).

Constraint Decision Diagrams (CDDs)

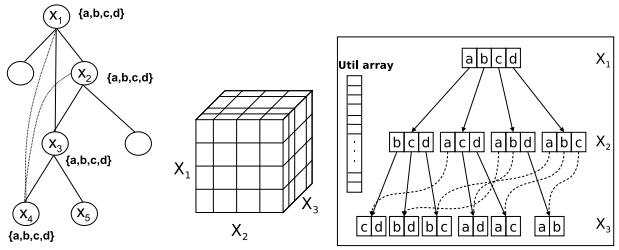
A Constraint Decision Diagram (Cheng and Yap 2005) $\mathcal{G} =< \Gamma, G >$ encodes the consistent assignments for a set of constraints Γ in a rooted direct acyclic graph $G = (V, E)$ by means of constraint propagation. A node in G is called a CDD node. The terminal nodes are either $true$ or $false$ implying consistent or inconsistent assignment. By default a CDD represents consistent assignments omitting the $true$ terminal.

Every nonterminal node has a number of successors $U \subseteq V$. A non terminal node v is a non empty set $\{(c_1, u_1), \ldots, (c_n, u_m)\}$. Each branch (c_j, u_j) consists of a constraint $c_j(x_1, \ldots, x_k)$ and a successor u_j of v. Figure 1(c) shows a CDD representation of the UTIL message sent from X_4 to X_3 in the DFS from Figure 1(a). The CDD has a level for each variable in $Sep_4 = \{X_1, X_2, X_3\}$. The root node shows that X_1 can take any value {a,b,c,d}. Subsequently, the second level corresponds to variable X_2, and the possible values of X_2 that are available depend on the assignment of X_1. For example, if $X_1 = a$ (the left-most branch connecting the root node to its left-most child), then X_2 can only take values from {b,c,d} ($X_2 = a$ is forbidden by the inequality constraint $X_1 \neq X_2$). Going another level down to X_3, we notice that the space of possible values is even more restricted, as it depends on the assignments of X_1 and X_2. For example, if $X_1 = a$ and $X_2 = b$, X_3 can only take values from {c,d}. Notice that the CDD represents just $4 \times 3 \times 2 = 24$ assignments, as opposed to $4^3 = 64$ for the hypercube representation used by DPOP, thus, quite a significant space saving.

It is easy to see that the CDD is actually a compact representation of a solution space, and similar to AND/OR search spaces (Mateescu and Dechter 2006).

H-DPOP - Pruning the Search Space Using Hard Constraints

The H-DPOP algorithm leverages the pruning power of hard constraints by using CDDs to effectively reduce the message size. As in DPOP, H-DPOP has three phases as well: the DFS arrangement of the problem graph, bottom up UTIL propagation and top down VALUE propagation. The DFS and VALUE phases are identical to the ones of DPOP. The UTIL phase is described below.



(a) DFS tree (b) Hypercube (c) CDD Message

Figure 1: (a) DFS tree; $X_1 - X_4$ are connnected by inequality constraints; (b) $UTIL_4^3$ message from X_4 to X_3 has $4^3 = 64$ assignments; (c) CDD equivalent of $UTIL_4^3$ has 24 assignments.

Algorithm 2: MakeCDD(X,Γ,u)

X:variable set **Γ**:constraint set **u**:partial assignment
Returns: The root node of the CDD for X

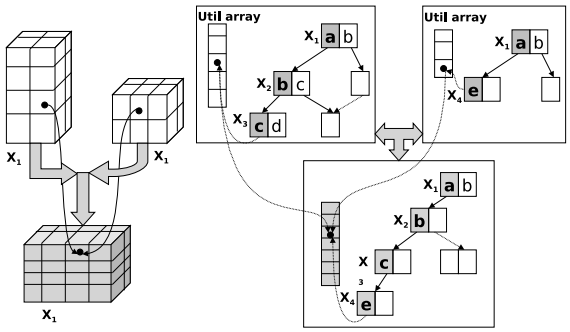
```
1  if X = φ then
2  |  Return terminal true
3  choose a X_k ∈ X
4  Edges_{X_k} ← φ, D^C_{X_k} ← φ
5  forall d ∈ D_{X_k} do
6  |  r ← (u, d)
7  |  if ¬Consistent(Γ, r) then
8  |  |  Skip to next d ∈ D_{X_k}
9  |  u^G ← MakeCDD(X\X_k, Γ, r)
10 |  Edges_{X_k} ← Edges_{X_k} ∪ {< d, u^G >}
11 |  D^C_{X_k} ← D^C_{X_k} ∪ {d}
12 if D^C_{X_k} = φ then
13 |  Return terminal false
14 v^G ← mkNode(X_k, Edges_{X_k}, D^C_{X_k})
15 Return v^G
```

The UTIL message in DPOP, a multidimensional hypercube, is replaced by a $CDDMessage$ in H-DPOP. This phase is described in Algorithm 1 but with a different $Join$ and $Project$ operations. We define the CDDmessage structure followed by $Join$ and $Project$ operations on it.

Definition 7 (CDDMessage) *A CDDMessage $\mathcal{M}_i^j$ sent by agent X_i to agent X_j is a tuple $< \overrightarrow{u}, \mathcal{G} >$ where $\overrightarrow{u}$ is a vector of real valued utilities, $\mathcal{G}$ is a CDD defined over variables in Sep_i, and $dim(\mathcal{M}_i^j) = Sep_i$. The set of constraints for $\mathcal{G}$ is $\Gamma = \{f_i | scope(f_i) \subseteq Sep_i\}$.*

The utility vector $\overrightarrow{u}$ is indexed by the DFS numbering of the paths (the assignment from root till the leaf) of the graph G in CDD $\mathcal{G}$. The functions $f_i \in F$ are defined in definition 1. They include both the soft and hard constraints, the latter being used for pruning via consistency check.

Algorithm 2 shows the construction of a CDD for the CDDMessage at a leaf X_i. The variable set X is initialized with Sep_i. The constraints set Γ is defined in definition 7. The partial assignment u is initially empty.



(a) Hypercube Join (b) CDDMessage join

Figure 2: Join of two hypercubes and CDDMessages. Hypercubes are joined by simply adding the values in the corresponding cells. CDDMessages are joined by iterating through all possible compatible paths, and adding the values in the corresponding cells of the Util arrays.

Algorithm 3: Join($\mathcal{M}_1, \ldots, \mathcal{M}_n$)

$\mathcal{M}_1, \ldots, \mathcal{M}_n$: The CDDMessages to be joined
Returns: The joined CDDMessage $\mathcal{M}_{joined}$

1 $U \leftarrow \cup_i dim(\mathcal{M}_i)$
2 $\Gamma \leftarrow \cup_i e_{\mathcal{M}_i} \cup \{f_i | scope(f_i) \subseteq U\}$
3 $v^{\mathcal{G}} \leftarrow MakeCDD(U,\ \Gamma,\ \phi)$
4 $\mathcal{M}_{joined} \leftarrow < \overrightarrow{v}, \mathcal{G} >$
forall *assignment* $r \in \mathcal{G}$ **do**
5 $\quad util(r) \leftarrow (\sum_i util_i)(r)$
6 $\quad$ set $util(r)$ at appropriate place in $\overrightarrow{v}$
7 Return $\mathcal{M}_{joined}$

If the set of variables is empty, the terminal $true$ is returned (line 2), else we choose an un-instantiated variable X_k (line 3). The set $Edges_{X_k}$ stores the branches of the CDD node, $D^{\mathcal{C}}_{X_k}$ is the set of *consistent* domain values for X_k, both initially empty. We then iterate through the domain of X_k and subsequently prune inconsistent values (line 7). After this we find a CDD node $u^{\mathcal{G}}$ for the subproblem (line 9). This node becomes a child of the CDD node for X_k and edges and consistent domain values are updated (line 10,11). If after iterating through the complete domain of X_k there are no consistent values we return a terminal $false$ (line 13). Otherwise we invoke a procedure $mkNode$ which makes a new CDD node given the variable, its children and consistent domain values. This method is similar to (Cheng and Yap 2005). It makes a new CDD node such that the resulting CDD is reduced i.e. of minimal size.

The function $Consistent$ (line 7) is of special importance for pruning. It checks every partial assignment r for consistency against a set of constraint Γ. Any inconsistent assignment is pruned.

We also define a boolean function $e_{\mathcal{M}}$ for the CDDMessage $\mathcal{M}$. It takes as input any assignment r to a set of variables S, and returns $true$ if $r_{dim(\mathcal{M})\cap S}$ is consistent w.r.t to the CDD in $\mathcal{M}$, otherwise $false$.

Figure 2 and algorithm 3 describes the $Join$ operation on

Algorithm 4: Project($X,\ \mathcal{M}_k$)

Projection of variable $\mathbf{X}$ from CDDMessage $\mathcal{M}_{\mathbf{k}}$
Returns: The resulting message $\mathcal{M}_{project}$

1 $U \leftarrow dim(\mathcal{M}_k) - X$
2 $\Gamma \leftarrow e_{\mathcal{M}_k}$
3 $v^{\mathcal{G}} \leftarrow MakeCDD(U,\ \Gamma,\ \phi)$
4 $\mathcal{M}_{project} \leftarrow < \overrightarrow{v}, \mathcal{G} >$
forall *assignment* $r \in \mathcal{G}$ **do**
5 $\quad util(r) \leftarrow util^{max}_X(r)$
6 $\quad$ set $util(r)$ at appropriate place in $\overrightarrow{v}$
7 Return $\mathcal{M}_{project}$

n CDDMessages. $util(r)$ denotes the utility for the assignment r in the UTIL message in context. The constraint set Γ for the resulting CDDMessage is augmented by the $e_{\mathcal{M}_i}$ function for the combining messages (line 2). This step ensures that every assignment in the joined message is consistent w.r.t the source messages. The utility vector $\overrightarrow{v}$ is computed by joining the respective utilities in the source messages (lines 5,6, also see definition 5 for the join operation).

Algorithm 4 describes the projection of a variable X from a CDDMessage $\mathcal{M}_k$. The constraint set Γ includes the $e_{\mathcal{M}_k}$ function to make the resulting CDD consistent w.r.t. the original CDD of $\mathcal{M}_k$. The utility vector of the resulting $\mathcal{M}_{project}$ is set by the project operation (see definition 6). The index of $util(r)$ in the vector $\overrightarrow{v}$ is found out by the DFS numbering of assignment r in the CDD $\mathcal{G}$ (line 6).

The next section describes various experiments illustrating the pruning power of hard constraints in H-DPOP.

Experiments

This section discusses the performance of H-DPOP w.r.t DPOP on message size. The experiments were performed on the FRODO platform (publicly available, (Petcu 2006)). For the size comparisons we use the logical size of the hypercubes in DPOP and the CDDMessage in H-DPOP. The size of the hypercube is the number of entries in the hypercube, the size of the CDDMessage is the number of entries in the utility vector combined with the logical size of the CDD graph (each entry in the CDD Node corresponds to 1 unit in the space measurement, links to children are also counted as 1 unit).

Graph Coloring

We performed experiments on randomly generated distributed graph coloring problems. In our setup each node in the graph is assigned an agent (or a variable in DCOP terms). The constraints among agents define the cost of having a particular color combination. The cost of two neighboring agents choosing the same color is kept very high (10000) to disallow such combinations. This is the $hard\ constraint$ in the problem. The *domain* of each agent is the set of available colors. The mutual task of all the agents is to find an optimal coloring assignment to their respective nodes.

For generating the graphs we keep number of agents fixed to 10 and vary the constraint density to see its effect on the pruning power of hard constraints.

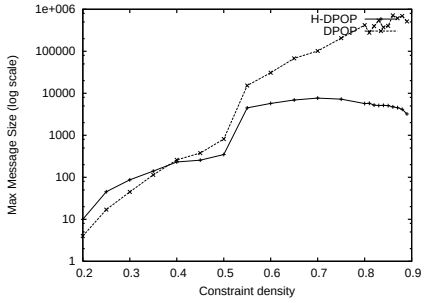


Figure 3: Graph Coloring: H-DPOP Vs DPOP performance

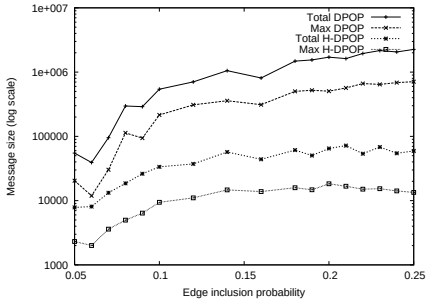


Figure 4: 6×6 N-Queens problem using graph coloring

Figure 3 shows the maximum message size in DPOP and H-DPOP for problems with a range of constraint densities(0.2-0.89). The chromatic number χ for problems within densities 0.2-0.5 was 4, within densities 0.5-0.9 was 6. For each constraint density we generated 50 random problems and the results shown are the the averages.

For the *low density* region (0.2-0.4) DPOP performs better than H-DPOP. At low density the size of the hypercube is small. Hard constraints at low density do not provide sufficient pruning as the search space is also small. So the pruning done by the CDDs is over-accounted by the size of the CDD graph in the CDDMessage. For the *medium density* region (0.4-0.7) H-DPOP is much superior to DPOP. This is due to the pruning by hard constraints.

The *high density* region (0.7-0.9) provides interesting results for H-DPOP. In DPOP, as expected, the maximum message size increases with the density. However we see an opposite trend in H-DPOP. The reason is that at high constraint density the extent of pruning achievable by hard constraints is also very high. So although the problem becomes more complex with high connectivity, the increased pruning using hard constraints overcome this increase in complexity.

N-Queens problem using graph coloring: In random graph coloring agents interact in an unstructured manner. We further experimented with graph coloring instances which exhibit structured interactions. For a $n \times n$ chessboard a queen graph contains n^2 nodes, each corresponding to a square of the board. Two nodes are connected by an edge if the corresponding squares are in the same row, column, or diagonal. The queen problem is to place n sets of n queens on the board so that no two queens of the same set attack each other. This problem is solvable by using graph coloring if the coloring number of the graph is at least n. The test instances are available at Stanford Graphbase (Knuth 1993).

For a 5-colorable 5×5 queen graph (width 19, density 0.53) DPOP was unable to execute (Maximum message size 19073486328125). H-DPOP successfully executed in 15 secs with a maximum message size 9465 because of high pruning provided by CDDs. However for board sizes 6×6 (7 colorable with width 31, density 0.46) and above H-DPOP was also unable to execute due to increased width and domain size. We relaxed the queen graph constraints so that the inclusion of any edge in the graph is done with an edge inclusion probability p.

Figure 4 shows the message size statistics for DPOP and H-DPOP on queen graph. H-DPOP is again much superior to DPOP in message size and for higher problem complexity (determined by edge inclusion probability) the message size nearly becomes constant due to increased pruning.

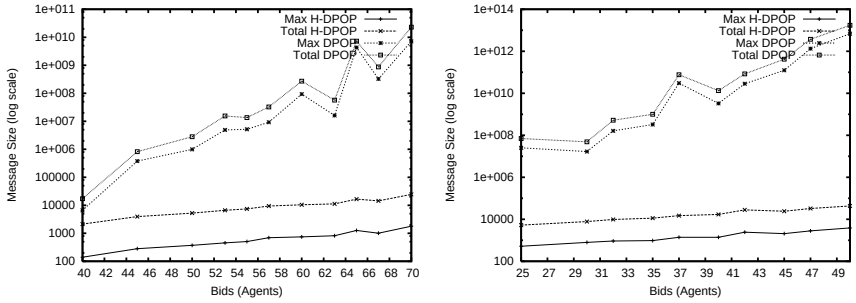
Winner Determination in Combinatorial Auctions

Combinatorial Auctions (CA) provide an efficient means to allocate resources to multiple agents. The agents are distributed (geographically or logically) and have information about *only those* agents with whom their bids overlap. The task of agents is to find a solution (assign winning or losing to bids) which maximizes the revenue.

The *variables* in our setting are the *bids* presented by the agents. Each agent is responsible for the bid it presents. The *domain* is the set $\{wining,\ losing\}$. Hard constraints are formulated among bids sharing one or more goods disallowing them to be assigned *winning* in the optimal solution. The gain provided by a winning bid is modeled as the unary constraint on the variable.

We generated random problems using CATS (Combinatorial Auctions Test Suite, (Leyton-Brown, Pearson, and Shoham 2000) using the *Paths* and the *Arbitrary* distributions. For the paths distribution, we vary the number of bids for a fixed number of goods (100). In the *paths* distribution, the goods are the edges in the network of cities. Agents place bids on a path from one city to another based on their utilities. We fixed the number of cities to 100 with initial connection 2 (link density). Since the city network structure is fixed, as the number of bids increase we expect a higher number of bids to overlap with each other and increase the problem complexity. For the *Arbitrary* distribution we use all the default CATS parameters. The number of goods is 50 and the number of bids varies from 25 to 50 increasing the problem complexity. Each data point is the average of 20 instances.

Figure 5(a) shows the maximum and total message size comparison of DPOP with H-DPOP on the *paths* distribution. DPOP as expected increases in the message complexity with the number of bids. The savings in the message size provided by H-DPOP are more than an order of magnitude and increase with the number of bids. H-DPOP is able to solve problems with very high width (35, bids=70) exploiting increased pruning by hard constraints, where memory



(a) Paths Distribution (b) Arbitrary Distribution

Figure 5: Combinatorial Auctions: H-DPOP Vs DPOP comparison

requirements for DPOP are prohibitively expensive. We see a similar trend for the *arbitrary* distribution (figure 5(b)).

As far as execution time is concerned, for a relatively small number of bids (below 45) DPOP performs better than H-DPOP. The construction of CDDs is computationally expensive. But H-DPOP is not significantly slower (both of them finish within a few seconds for these problem sizes). However as the number of bids increases, memory requirements as well as execution time increase much faster for DPOP. Beyond bids=58, DPOP fails to execute altogether, while H-DPOP still scales well. The maximum time H-DPOP took for any instance was 205s for bids=70 for the paths distribution.

Conclusions and Future Work

Distributed constraint optimization has seen steady progress over the past few years (Faltings 2006). Dynamic programming algorithms are attractive because the number of messages grows only linearly with problem size, but have the drawback that messages can become excessively large. Often, much of the information they carry is redundant because the corresponding value combinations are not consistent with the hard constraints of the problem.

We have shown how to maintain a more efficient representation of dynamic programming messages in the form of decision diagrams. In problems that are strongly constrained by hard constraints, they allow to dramatically reduce the size of the messages, making it possible to solve problems like combinatorial auctions by a distributed algorithm. We believe that there are numerous other applications where decentralized optimization is useful, and this technique can make it practical.

References

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

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

Faltings, B. 2006. *Distributed Constraint Programming*. Foundations of Artificial Intelligence. Elsevier. 699–729.

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

Kask, K.; Dechter, R.; Larrosa, J.; and Dechter, A. 2005. Unifying cluster-tree decompositions for automated reasoning in graphical models. *Artificial Intelligence*.

Knuth, D. E. 1993. The stanford graphbase: a platform for combinatorial algorithms. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, 41 – 43.

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

Mateescu, R., and Dechter, R. 2006. Compiling Constraint Networks into AND/OR Multi-Valued Decision Diagrams (AOMDDs). In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP'06)*.

Meisels, A., and Zivan, R. 2003. Asynchronous forward-checking on DisCSPs. In *Proceedings of the Distributed Constraint Reasoning Workshop, IJCAI 2003, Acapulco, Mexico*.

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

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

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/`.

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

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem - formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 10(5):673–685.