Distributed Multi-Criteria Coordination in Multi-Agent Systems

Emma Bowring and Milind Tambe
Computer Science Dept.
University of Southern California
Los Angeles CA 90089
{bowring,tambe}@usc.edu

Makoto Yokoo
Dept. of Intelligent Systems
Kyushu University
Fukuoka, 812-8581 Japan
yokoo@is.kyushu-u.ac.jp

May 11, 2005

Abstract

Distributed constraint optimization (DCOP) has emerged as a key technique for multiagent coordination. Unfortunately, while previous work in DCOP focuses on optimizing a single team objective, domains often require satisfying additional criteria. This paper provides a novel multi-criteria DCOP algorithm, based on two key ideas: (i) transforming multi-criteria problems via virtual variables to harness single-criterion DCOP algorithms; (ii) revealing bounds on criteria to neighbors. These ideas result in interleaved multi-criteria searches, illustrated by modifying Adopt, one of the most efficient DCOP algorithms. Our Multi-Criteria Adopt algorithm (MCA) tailors its performance to whether individual constraints are to be kept private or exploited for efficiency.

1 Introduction

Distributed Constraint Optimization (DCOP)[1, 2, 3, 4, 5, 6] is a useful technique for multiagent coordination with current and potential applicationssuch as distributed meeting scheduling, distributed factory & staff scheduling, and sensor nets [7, 8, 9, 10]. In a DCOP, distributed agents, each in control of a set of variables, assign values to these variables, so as to optimize a single global objective function expressed as an aggregation of utility functions over combinations of assigned values.

While the recent advances in efficient DCOP algorithms are encouraging [1, 2, 11], these algorithms focus on single-criteria optimization, and fail to capture the complexity in many domains, where individual agents' resource constraints necessitate multi-criteria optimization. For instance, in distributed meeting scheduling[8], agents must collectively optimize users' time and, in addition, must adhere to each user's limited travel budget. While in some domains agent must not share the additional individual criteria (e.g., individual user's budgets), in other domains, such as staff allocation for distributed software development, costs of satellite transmission between software-centers are shareable (non-private) criteria.

Thus, there is a need for multi-criteria DCOP algorithms. However, there are two key challenges in designing such algorithms. First, harnessing state-of-the-art DCOP algorithms is crucial for efficiency. This is challenging because the additional criteria are local, they have different domains and they may be private, making it difficult to automatically employ single-criteria algorithms. Furthermore, it is difficult to modify existing algorithms to generate a ranked series of solutions from which to select one that meets all the criteria. Second, while we wish to maintain the privacy of the additional criteria at an agent where specified, the algorithm must exploit sharing the criteria to gain efficiency where not specified.

This paper presents a novel algorithm for multi-criteria DCOP, that allows distributed optimization of a global utility function as well as satisfaction of additional local criteria (e.g. budget) at each agent. To address the first challenge, we modify the original DCOP by adding a virtual variable v' for the additional criteria at each variable v. Variable v' is owned by the same agent as v and has a new utility function that provides high negative utility if the neighbors' values violate a criterion tested by v'. These additional virtual variables create a larger DCOP, but enable single-criterion DCOP algorithms to be exploited for multi-criteria reasoning while maintaining privacy. To address the second challenge, an agent reveals to its neighbors an upper-bound on any non-private criteria. Thus, when optimizing the global objectives, the neighbors only pre-select values that abide by the bounds, significantly improving algorithmic efficiency. In locally acyclic DCOP graphs, these bounds are tightened without sacrificing algorithmic correctness leading to further efficiency improvements — however we show that these tighter bounds can not be applied in general DCOP problems. We also formally define the concept of T-nodes, which are variables whose local graph acyclicity allows for the use of tightened bounds.

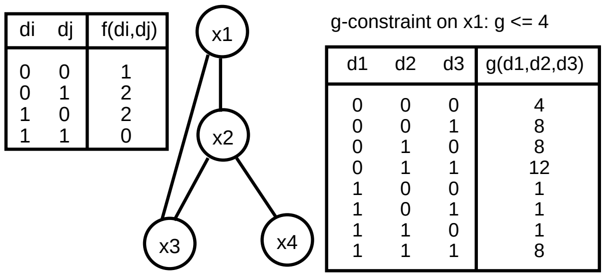
While we illustrate these ideas by building on top of Adopt, one of the most efficient DCOP algorithms [1], our techniques could be applied to other DCOP algorithms, e.g. OptAPO and SynchBB [2, 3]. We present a multi-criteria Adopt algorithm that tailors its performance to three separate cases: 1) when a constraint must be kept private, 2) when a constraint is sharable but the variable is not a T-node, and 3) when a constraint is sharable and the variable is a T-node. We exploit sharability of constraints and local acyclicity in the graph structure to improve efficiency. These different cases can be applied simultaneously to different criteria or different nodes in the same problem. We experimentally compare these techniques and illustrate the tradeoffs in efficiency and privacy. These experiments reveal regions of the space where agents may gain the most efficiency by sharing criteria, and where agents lose no efficiency by maintaining privacy.

2 Background: DCOP and Adopt

DCOP: A DCOP consists of n variables, $\{x_1, x_2, \ldots, x_n\}$, assigned to a set of agents who control the values they take on. Variable x_i can take on any value from the discrete finite domain D_i. The goal is to choose values for the variables such that the sum over a set of constraints and associated cost functions, $f_{ij} : D_i \times D_j \rightarrow N \cup \infty$, is minimized. More formally, find an assignment, A, s.t. F(A) is minimized: $F(A) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j)$, where $x_i \leftarrow d_i, x_j \leftarrow d_j \in A$

Taking as an example the constraint graph in Figure 1 where x_1, x_2, x_3, and x_4 are variables each with domain $\{0,1\}$ and the f cost function shown (ignoring g), $F((x_1,0),(x_2,0),(x_3,0),(x_4,0)) = 4$ and in this example the optimal would be $(x_1,1),(x_2,1),(x_3,1),(x_4,1)$.

Adopt: Adopt[1] is a complete DCOP algorithm, guaranteed to find the optimal solution to the problem. It starts by organizing agents into a Depth-First Search (DFS) tree in which constraints are allowed between a variable and any of its ancestors or descendants, but not between variables in separate sub-trees. Note that the constraint graph in Figure 1 is organized as a DFS tree. x2 is a child of x1 in this tree, and x3 is a descendant (but not a child) of x1. While for expository purposes it is useful to consider each variable as belonging to a separate agent, our algorithm does not require a single variable per agent.



di	dj	f(di,dj)
0	0	1
0	1	2
1	0	2
1	1	0

d1	d2	d3	g(d1,d2,d3)
0	0	0	4
0	0	1	8
0	1	0	8
0	1	1	12
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	8

Figure 1: Multi-Criteria constraint graph

Adopt uses three basic messages: VALUE, THRESHOLD and COST. Assignments of values to variables are conveyed in VALUE messages that are sent to neighbor nodes, i.e. nodes sharing a constraint with the sender, lower in the DFS tree. When the algorithm starts, nodes take on a random value and send out appropriate VALUE messages to get the flow of computation started. A COST message is sent from children to parents indicating the cost of the sub-tree rooted at the child. A THRESHOLD message is sent from parents to children and contains a backtrack threshold, initially set to zero. A sub-tree explores a line of search until the lower bound on cost accumulated, i.e. the lower bound cost of its child sub-trees plus the cost of its constraints with its ancestors, surpasses its backtrack threshold. When this occurs, it will attempt to change its value to one that has a lower bound cost still under the threshold (unexplored assignments have a lower bound of zero). If it can't then a variable raises its threshold and notifies its parent, who then rebalances the thresholds of its other children. The root's upper and lower bounds represent the upper and lower bounds on the global problem, so when they meet the optimal solution has been found and the algorithm terminates. Since communication is asynchronous messages have a context, i.e. a list of the variable assignments in existence at the time of sending, attached to them to help determine information relevance. The pseudo-code for these procedures is shown in the lines not marked with an 'm' in Figure 2.

3 Problem Definition

We define Multi-Criteria DCOP by adding k additional cost functions at each node x_i, $g_i^1 \dots g_i^k$, that are to be held below $T_i^1 \dots T_i^k$ respectively. Figure 1 shows an example g function and constraint on x_1. In the example, if x_1, x_2, x_3 each take on the value of 1 (the single criterion optimal) then the g-cost is 8, which violates the g-constraint of 4 at x_1. Multiple g-functions may be defined on each variable. Since these g functions cannot be merged with the original f functions, each value must be selected based on multiple criteria, and hence this is a multi-criteria DCOP.

We define g-constraints on variables and the constraints can either be kept private or shared. If the constraint is private then it can be an arbitrary function on the values of x_i and its neighbors. If, however, the constraint is shared then it must be a sum of the g-functions of the links impinging upon x_i. Given the g-functions, we now modify the DCOP objective to be: find A s.t. F(A) is minimized: $F(A) = \sum_{x_i,x_j \in V} f_{ij}(d_i, d_j)$, where $x_i \leftarrow d_i, x_j \leftarrow d_j \in A$
and $\forall x_a \in V$
$g_a^c(d_a, \{d_b | x_b \in neighbors(x_a)\}) \leq T_a^c, 1 \leq c \leq k$

4 Multi-Criteria Adopt

4.1 Basic Ideas

This section is organized as follows: Section 4.1 describes our algorithm, Section 4.2 provides correctness results and Section 4.3 provides complexity analysis.

Problem Transformation To harness the efficiency of single-criterion DCOP algorithms while maintaining privacy of the additional criteria, we add virtual variables enforcing n-ary constraints, e.g. for the problem described in Figure 1, we add x_1' that has an n-ary constraint with x_1, x_2, x_3. These virtual variables (e.g. x_1') are under the control of the agent controlling the original variable (e.g. x_1), and enforce the extra criteria by sending infinite costs when a constraint is violated to the nodes involved. Virtual variables are employed when the constraint is private or the local subgraph is not acyclic. Since typical DCOP algorithms such Adopt are based on binary cost functions, n-ary constraints require additional mechanisms within such algorithms. This problem transformation could be applied to other algorithms such as SynchBB and OptAPO [2, 3].

Mutually Constrained Search When privacy is not essential, we want to mutually constrain the searches to gain efficiency. This is difficult because improvement in one criterion does not necessarily correspond with improvement in the others. We tackle

```
Preprocessing
(1m)  forall $x_i$, if $Tnode_i = false$ or $\exists g_i^c$ s.t. $private_i^c = true$
(2m)      $x_i'$ is a new virtual variable
(3m)      $Neighbors(x_i') \leftarrow Neighbors(x_i) \cup x_i$
(4m)      $Neighbors(x_i) \leftarrow Neighbors(x_i) \cup x_i'$
(5m)      forall $x_j \in Children(x_i)$
(6m)          $x_j$ must be neighbor of at least one $x_k \in Children(x_i)$
(7m)  buildTree($x_1 \ldots x_n$)
(8m)  forall $x_i'$
(9m)      $parent(x_i') \leftarrow$ lowest priority Neighbor of $x_i'$

Initialize
(10)  $threshold \leftarrow 0$; $CurrentContext \leftarrow \{\}$
(11)  forall $d \in D_i, x_l \in Children$
(12)      $lb(d, x_l) \leftarrow 0$; $t(d, x_l) \leftarrow 0$
(13)      $ub(d, x_l) \leftarrow Inf$; $context(d, x_l) \leftarrow \{\}$;
(14m) forall g-constraints $g_i^c$ if $private_i^c = false$ and $Tnode_i = true$
(15m)     forall $x_l \in Children$
(16m)         $gThresh_i^c(x_l) \leftarrow 0$; $gContext_i^c(x_l) \leftarrow \{\}$
(17)  $d_i \leftarrow d$ that minimizes $LB(d)$
(18)  backTrack

when received (THRESHOLD, $t$, $context$)
(19)  if $context$ compatible with $CurrentContext$:
(20)      $threshold \leftarrow t$

when received (VALUE, $x_j$, $d_j$, $gThresh_j^{1 \ldots k}$)
(21)  add ($x_j$,$d_j$) to $CurrentContext$
(22m) forall constraints $g_j^c$ if $private_j^c = false$ and $Tnode_j^c = true$
(23m)     add ($x_j$,$d_j$,$gThresh_j^c$) to $CurrentContext$
(24)  forall $d \in D_i, x_l \in Children$
(25)      if $context(d, x_l)$ incompatible with $CurrentContext$:
(26)          $lb(d, x_l) \leftarrow 0$; $t(d, x_l) \leftarrow 0$
(27)          $ub(d, x_l) \leftarrow Inf$; $context(d, x_l) \leftarrow \{\}$
(28m) forall constraints $g_i^c$ if $private_i^c = false$ and $Tnode_i = true$
(29m)     forall $x_l \in Children$
(30m)         if $gContext^c(x_l)$ incompatible with $CurrentContext$:
(31m)             $gThresh^c(x_l) \leftarrow 0$; $gContext^c(x_l) \leftarrow \{\}$
(32)  backTrack;

when received (COST, $x_k$, $context$, $lb$, $ub$)
(33)  $d \leftarrow$ value of $x_i$ in $context$
(34)  remove ($x_i$,$d$) from $context$
(35)  forall ($x_j$,$d_j$) $\in context$ s.t. $x_j \notin Neighbors(x_i)$
(36)      add ($x_j$,$d_j$) to $CurrentContext$;
(37)  forall $d' \in D_i, x_l \in Children$
(38)      if $context(d', x_l)$ incompatible with $CurrentContext$:
(39)          $lb(d', x_l) \leftarrow 0$; $t(d', x_l) \leftarrow 0$
(40)          $ub(d', x_l) \leftarrow Inf$; $context(d', x_l) \leftarrow \{\}$;
(41)  if $context$ compatible with $CurrentContext$:
(42)      $lb(d, x_k) \leftarrow lb$; $ub(d, x_k) \leftarrow ub$
(43)      $context(d, x_k) \leftarrow context$
(44)  backTrack
```

```
procedure backTrack
(45m) if $x_i$ not a virtual variable
(46)      if $threshold == UB$:
(47)          $d_i \leftarrow d$ that minimizes $UB(d)$
(48)      else if $LB(d_i) > threshold$:
(49m)         $d_i \leftarrow d$ that minimizes $LB(d)$ and
              satisfies $gThresh_j^c$ for all $g_j^c$ where $x_j \in Ancestors(x_i)$ and $private_j^c = false$
(50m)         SEND (VALUE, ($x_i$, $d_i$, $gThresh^{1...m}(x_k)$)) to each lower priority neighbor $x_k$
(51m)             omit $gThresh^c(x_k)$ if $private_i^c = true$
(52)      if $threshold == UB$:
(53)          if TERMINATE received from parent or $x_i$ is root:
(54)              SEND TERMINATE to each child
(55)              Terminate execution;
(56)      SEND (COST, $x_i$, $CurrentContext$, $LB$, $UB$)
              to parent
(57m) else
(58m)     if $g^c(d_i, \{d_j | x_j \in Neighbors(x_i)\}) > gConstraint^c(x_i)$
              $1 \leq c \leq k$
(59m)         SEND (COST, $x_i$, $CurrentContext$, $\infty$, $\infty$)
(60m)     else
(61m)         SEND (COST, $x_i$, $CurrentContext$, 0, 0)
```

Figure 2: Multi-Criteria Adopt Pseudo-code

this by requiring descendant nodes to only consider assignments that will not violate the g-constraints of their ancestors. This is done by passing them each a bound specifying how large a g-cost they can pass up. This bound can represent an exact bound when the local graph is acyclic and an upper bound otherwise. If a suitable division of g cannot be found then the algorithm will not try to optimize f. It also doesn't attempt to satisfy g-constraints for assignments that are highly suboptimal in f. Thus, the searches mutually constrain each other, leading to performance improvements.

Local Acyclicity (T-nodes) The notion of locally acyclic graph structure is captured more formally via our definition of T-nodes given below. Variable x_i is a T-node if $\forall x_k, x_l \in Children(x_i)$ there does not exist a path of links with constraints $l_{k,1}, l_{1,2}, \ldots, l_{l-1,l}$ passing through intermediary nodes $x_1, \ldots x_{l-1}$ and $x_1, \ldots, x_{l-1}$ are all lower priority than x_i. Note that another way to describe a T-node is as a variable x_i where $Children(x_i) = Successors(x_i)$ where successors are all lower priority neighbors and children are those successors whose parent is x_i. In our original DCOP example in figure 1, x_1 is not a T-node because there is a link between two of its descendants (x_2 and x_3), but x_2, x_3 and x_4 are.

Figure 2 presents pseudo-code for the MCA algorithm; recall that lines marked with m are our modifications to the basic ADOPT algorithm, and that we have eliminated some details (such as termination detection) that do not change from the basic ADOPT algorithm. We will discuss each of our techniques separately, for clarity, and then indicate how they interact with each other in the unified algorithm. For simplicity, each technique will be assigned a name: Multi-Criteria Adopt Private (MCAP) is the case where a constraint may not be revealed to its neighbors, Multi-Criteria Adopt Shared (MCAS) is the case where a constraint is sharable but the variable is not a T-node, and Multi-Criteria Adopt Shared and Acyclic (MCASA) is the case where a constraint is sharable and the variable is a T-node.

We will first discuss MCAP. An example snapshot of its execution on the problem from Figure 1 is shown in Figure 3a. MCAP searches for an optimal solution for f and when an assignment violates any g-constraints, a negative feedback signal (of very high negative cost) is sent to the cluster of nodes involved. The feedback is sent by a virtual variable (x_1') which is responsible for enforcing the g-constraints of a single variable (x_1). The feedback is initially sent to the lowest priority of the variables involved in the broken constraint (x_3) . Using the single criterion optimization mechanisms of Adopt, the feedback will be propagated to the other node(s) that must change their values to find the optimal satisfying solution.

Since feedback must be able to be sent to all the nodes in a constraint, Adopt's DFS tree must be structured to keep all the variables in the same subtree. MCA pre-processing (lines 1-6 in figure 2) creates the virtual variables which are owned by the owner of the variable they represent. The virtual variables need to be placed as leaves lower in the DFS tree than the variables in the g-constraints it enforces (lines 5-6). The virtual variables will only receive VALUE messages and will then determine whether the current assignment violates any of the g-constraints it represents. If so, an infinite cost is passed up, forcing the variables to try a different assignment, otherwise a cost of 0 is passed up (lines 57-61).

While feedback signals have the advantage of maintaining the privacy of the g-functions and the variables involved in constraints, it has a drawback: its partial search of unsatisfying assignments slows the algorithm. When a criterion – the specific g-function and the constraint on it – are non-private, we can reveal the g-function of a link to those vertices connected to it, which is the same privacy loss tolerated for the f-function. With this information we can implement MCAS and MCASA. Snapshots of MCAS and MCASA are shown in Figures 3b and 3c respectively.

In MCAS and MCASA we exploit the sharing of the g-functions by using a mechanism similar to the THRESHOLD message in Adopt. In addition to the THRESHOLD message associated with f, parents send their descendants g-thresholds in the VALUE messages (lines 21-23) indicating that the child may take on no value with g greater than the g-threshold. In the snapshots from Figures 3b and 3c we can see the g-thresholds being passed down from node x_1 to nodes x_2 and x_3. (Note that as discussed earlier, in case a g-function can be shared, we assume an additive g-function, enabling distinct g-thresholds to be sent to each node.) If the variable is not a T-node (in MCAS), the g-thresholds represent an upper bound, constituting the total g-constraint minus the minimum g usable on each link. In the example in Figure 3b, the total g-threshold on x_1 is 4, and the minimum usable on each link is 1, and hence the messages set the g-threshold upper-bound of 3 for each child. Given this upper-bound, a node can prune out certain values from its domain as violating the g-constraint (e.g., 0 is pruned from the domain of x_3) leading to speedups over MCAP. For T-nodes it is possible to calculate an exact bound (MCASA)— in Figure 3c, the exact bound is 0 for node x_3 and 4 for node x_2 – enabling more values to be pruned out, further speeding up the search. Note that Figure 3c is slightly modified from the original example to show T-nodes.

Until now we've discussed each technique separately for clarity. However, they can be applied simultaneously to different criteria within the same problem. If there are c separate criteria then the c different criteria can be enforced independently because they are independent functions. This is even true when a single variable x_i has one g-criterion that is MCASA and another that is either MCAP. It can perform the tree transformation for the MCAP constraint without losing the T-node property for the MCASA constraint because the new links have no constraint functions on them. Thus the main challenge lies in how to enforce a single criterion g^c that employs MCAP, MCAS and MCASA at different nodes in the network. Since the g-criteria are all local criteria we only need worry about situations where a parent and a child are using different techniques:

– Parent = MCAS/MCASA, Child = MCAP: The child can restrict its choice of domain values based upon the g-threshold of its parent. The parent will automatically change its value if the child's virtual variable passes up an infinite feedback cost.

– Parent = MCAP, Child = MCAS/MCASA: The child will automatically adjust its value if the parent's virtual variable passes up an infinite feedback cost. From the parent's perspective it makes no difference what technique the child is using.

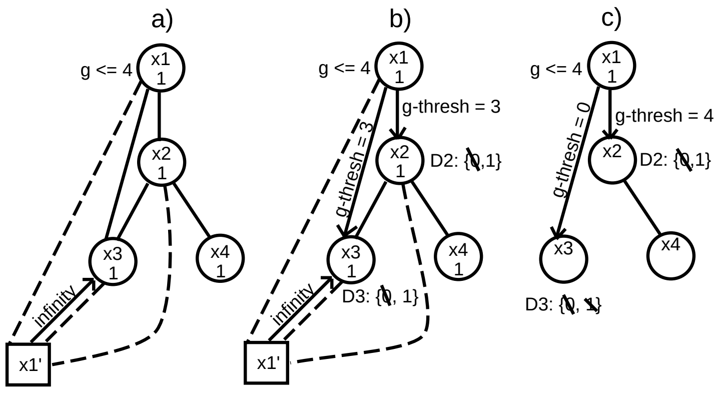


Figure 3: a) MCAP b) MCAS c) MCASA

4.2 Correctness of Multi-Criteria Adopt

In this section we will again separate out the proofs for each technique for the sake of clarity.

Proposition 1 *For each node x_i, MCAP finds the assignment whose f-cost, local cost (δ) plus the sum of x_i's children's costs, is minimized while satisfying the g-constraint:*
$OPT(x_i, context) \overset{def}{=}$
$min_{d \in D_i}[\delta(d) + \sum_{x_l} OPT(x_l, context \cup (x_i, d))]$
where $\delta(d) \overset{def}{=} \sum_{x_j \in ancestors} f_{ij}(d_i, d_j)$
if $g_i^c(d_i, \{d_j | x_j \in Neighbors(x_i)\}) \leq T^c, 1 \leq c \leq k$
∞ *otherwise*

Proof: To show this we start from the correctness of the original Adopt algorithm which was proven in [1]. Single-criterion Adopt will find at every node x_i:
$OPT'(x_i, context) \overset{def}{=}$
$min_{d \in D_i}[\delta'(d) + \sum_{x_l} OPT'(x_l, context \cup (x_i, d))]$
where $\delta'(d) \overset{def}{=} \sum_{x_j \in ancestors} f_{ij}(d_i, d_j)$

To show that MCAP finds $OPT(x_i, context)$ we show that 1) it never returns an assignment containing a violated g-constraint, unless the problem is unsatisfiable and 2) it finds the minimum f-cost solution.

The proof of part 1) starts with the fact that the virtual variables introduce an infinite f-cost into the subtree containing the violated constraint below any variable in the constraint. Since any assignment that does not violate the constraint will have a finite f-cost, it follows from the correctness of Adopt that by choosing the assignment that minimizes f, MCAP will never take on an assignment that violates a g-constraint unless the g-constraint is unsatisfiable. Part 2) follows directly from the correctness of Adopt because the virtual nodes report a zero cost if all constraints are satisfied, which means that by Adopt's normal mechanisms it will find the minimum f-cost solution. ■

Proving that MCAS is correct requires adding to the MCAP proofs that if the g-constraints for each node x_i are $\sum_{x_j \in Neighbors(x_i)} g_{ij}^c(d_i, d_j) < T_i^c$, then no satisfying solution can contain on link l_{il} a g greater than $T_i^c - \sum_{x_j \in Neighbors(x_i) \neq x_l} min$ $g_{ij}^c(d_i, d_j)$ $1 \leq c \leq k$. This requirement easily follows from the fact that each link consumes a certain minimum g-cost, and we are only subtracting the sum of the absolute minimum costs on all links.

We next turn to MCASA and discuss the exact bounds communicated to children of T-nodes in MCASA. We first show that when a node is not a T-node, i.e. its set of successors includes more nodes than its children, then exact bounds do not apply.

Proposition 2 *Given only the g-functions of links impinging on a variable x_i, it is not guaranteed to be able to calculate an exact bound on g for each successor if x_i is not a T-node.*

Proof: If x_i is not a T-node then there must exist at least one successor x_k who is not a child of x_i. There must exist a set of variables $x_1 \ldots x_{k-1}$ where x_{t+1} is a child of x_t that connects x_i to x_k. When x_i attempts to calculate the optimal split of g between x_1 and x_k, it will know only the g-functions on the links $l_{i,1}$ and $l_{i,k}$. However, the optimal g for each link is a function of the g-functions on all of the links $l_{t,t+1}$ and $l_{i,k}$. ■

Proposition 3 *If a node x_i is a T-node, then given its children x_1 through x_k, x_i is guaranteed to be able to calculate an exact bound on for each of its children.*

Proof: Since x_1 through x_k are all children of x_i, there are no links $l_{t,t+1}$ such that x_t is in the subtree rooted at one child and x_{t+1} is in the subtree rooted at another. Thus the f reported by each $x_l \; inChildren(x_i)$ will be independent of the g-threshold imposed upon any $x_m \in Children(x_i) \neq x_l$. Thus by knowing the relationship of g to f for each link l_{it} where $1 \leq t \leq k$, x_i can calculate the split of g between its children that will minimize the total f that will be reported to x_i. ■

Termination of Adopt is demonstrable because the $CurrentContext$ of each node eventually stabilizes, starting at the root which has $\{\}$ as its $CurrentContext$. This is unchanged in the MCA family of algorithms.

4.3 Complexity

The original DCOP problem is NP-hard, and by illustrating a solution to the multi-criteria problem which only adds linear number of additional nodes (in the number of criteria) we show that the complexity class has not worsened.

With respect to space, a key feature of Adopt is that its space complexity is linear in the number of nodes, n, specifically $|D_i|\,n$. In MCAP and MCAS, the space used at each regular node is the same, but we add up to n virtual variables, so the space complexity for MCAP and MCAS is $(|D_i| + 1)n$. In MCASA there are no virtual variables, but each node stores g information for each of its neighbors. If there are k g constraints on each node, the space complexity is $(k + 1)\,|D_i|\,n$.

In terms of messaging, MCAP and MCAS cause no increase in message size, but the virtual variables require sending up to n^2 extra VALUE messages and n COST messages. In contrast, MCASA sends no additional messages and due to the reduction in search space may send fewer messages than Adopt. However, the size of VALUE messages increases by a constant to incorporate k integer fields for the g-thresholds.

5 Experimental Results

This section presents results on four separate domain settings. Setting 1 focused on 20 node (agent) problems, each with 3 values per node, with a general graph (not acyclic), with average link density of 2.2 and maximum link density of 4. In this setting, both the f and g cost were randomly chosen from a uniform distribution varying from cost of 0 to 40. The g-cost was assumed to be an additive function (see Section 3). Setting 2 is similar to setting 1, except that we focused on 10 node problems. Settings 3 and 4 are essentially settings 1 and 2 respectively, but with acyclic graphs to emphasize the speedups due to MCASA. In each of the graphs below, each data-point is an average of 15 problem instances, i.e. we created 15 problem instances of each of our four domain settings and ran it for each experimental setup described.

To really highlight the tradeoff between efficiency and privacy, we first show the performance of each technique when applied to all the nodes in a problem i.e. we either apply MCAP to all the nodes or MCAS to all the nodes or MCASA to all the

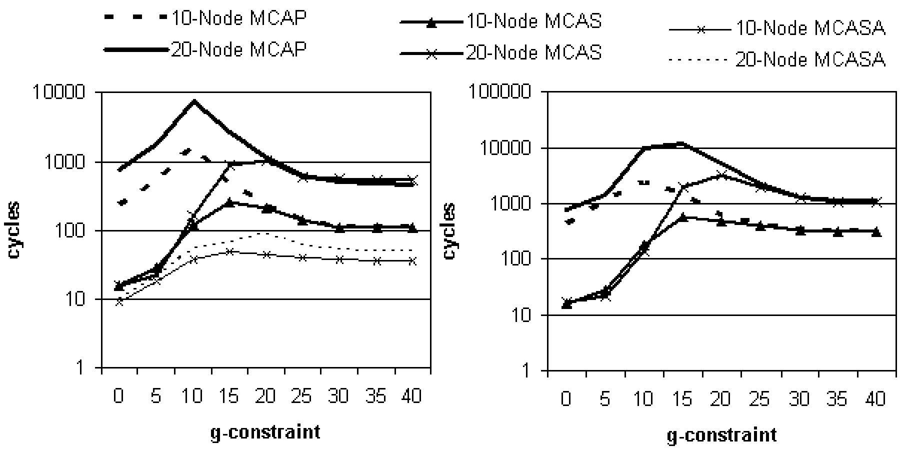


Figure 4: g-threshold vs. run-time for a) acyclic problems b) cyclic problems

nodes. Figure 4a shows the results of experiments measuring and comparing run-times of the MCA algorithms on 2-criteria problems. Each data point on the graph shows an average over the 15 instances from either setting 3 or setting 4. The x-axis shows the g-constraint applied to all variables and ranges from 0, which is unsatisfiable, to 40, which is effectively a single criterion optimization. The y-axis shows runtime — as with other DCOP systems, run-time is measured in cycles of execution where one cycle consists of all agents receiving incoming messages, performing local processing and sending outgoing messages[1]. The y-axis is logarithmically scaled.

The graphs show that the run-times of all the techniques are lowest when the search is either unconstrained due to high g or so constrained due to low g that little search space needs to be explored. Due to the privacy requirement, MCAP has the poorest performance. The upper bounds calculated by sharing information in MCAS improve performance, while the exact bounds and lack of tree-restructuring in MCASA give it the best performance. Figure 4b demonstrates similar results for cyclic problems (settings 1 and 2). There is a significant increase in runtime switching from acyclic to cyclic problem (note the y-axes are not identical). Interestingly, for tighter g-constraints, the MCAS algorithm outperforms MCAP, however, for looser constraints, there is very little difference in performance — the reason is that when g-constraints are not tight, MCAS upper-bounds provide very little pruning. In other words, if g-constraints are loose, there is no gain to sharing them with other agents, and agents may then at least retain their privacy.

In order to demonstrate the benefits of the per-criteria, per-node application of the MCAP and MCAS techniques, we took the 10-node examples from Figure 4a and b and randomly assigned first 25% and then 50% of the nodes to have private g-constraints while the remaining were assumed to be non-private. We then compared their performance to that of MCAP (100% private) and MCAS (0% private). The results are shown

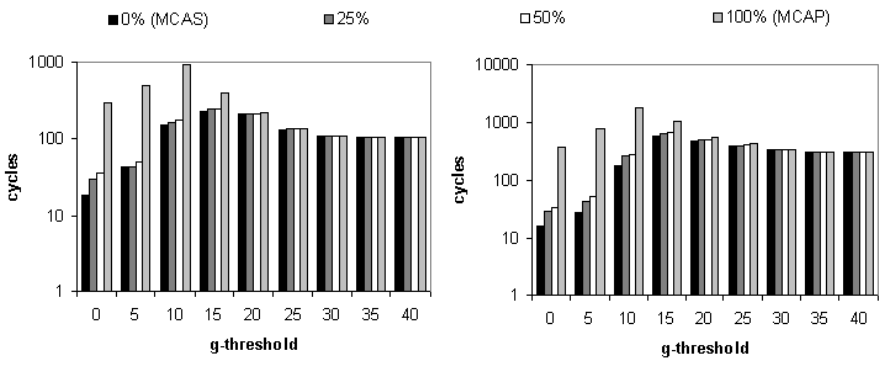


Figure 5: g-threshold vs. runtime for differing percentages of private constraints on a) acyclic problems and b) cyclic problems

in Figure 5a and b. The x-axis again shows the g-threshold applied and the y-axis measures the runtime in cycles on a logarithmic scale. Each bar in the graph shows an average over the 15 instances and we can see that as the percentage of nodes whose additional criterion is private increases, the runtime increases for smaller g-thresholds. However, as was found when comparing MCAS against MCAP in the previous figure, when the g-threshold on each variable is loose enough the runtimes converge because no pruning takes place as a result.

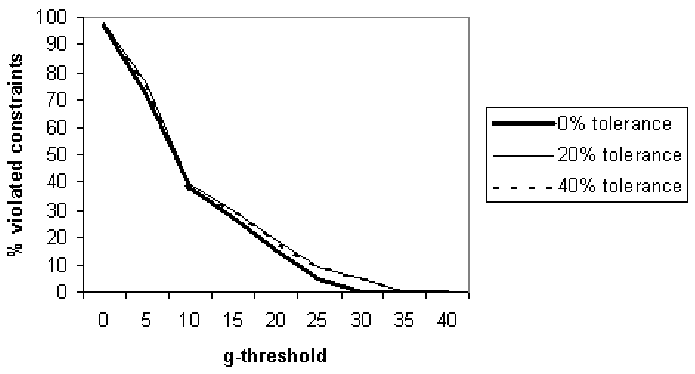


Figure 6: g-constraint violation by Adopt

While we developed MCA, one key question was whether single-criteria algorithms could automatically satisfy multiple criteria simply by loosening the tolerances on optimal solutions. It is possible to imagine specifying a tolerance on f and then running a single criterion DCOP algorithm on the problem ignoring g. We measure the degree to which a second g-criterion is violated by Adopt when ignoring g but specifying a tolerance on f. Figure 6 shows the results with each data point representing an average over 5 randomly generated runs. The x-axis gives the g-constraint applied to each variable and again varies from 0 to 40. The y-axis is the percentage of nodes whose g-constraint was not met by the solution obtained by Adopt. The tolerance is a percentage of the difference between the optimal and maximum cost solutions. The looser the g-constraint, the easier it is to meet without reasoning about f. Interestingly, a larger tolerance does not necessarily lead to improved g satisfaction, because since f and g are independent, increases in f do not necessarily correlate with decreases in g.

6 Conclusion and Related Work

DCOP has emerged as a key technology for multiagent coordination as it enables negotiation while protecting privacy. Previous work in DCOP focuses on optimizing a single global objective. This paper provides a novel multi-criteria DCOP algorithm, based on two key ideas: (i) transforming multi-criteria problems via extra virtual variables to harness single-criterion DCOP algorithms; (ii) revealing bounds on criteria to neighbors. These ideas result in interleaved multi-criteria searches and were realized in the context of Adopt, one of the most efficient current DCOP algorithms. The Multi-Criteria Adopt (MCA) algorithm modulates its performance based on the privacy requirements of individual constraints and the local network structure.

In terms of related work, there is significant continued progress in single criteria DCOP algorithms [1, 3, 4, 5, 6], e.g. OptAPO is shown to compare favorably with Adopt in some domains and vice versa[2]. Our work is complementary to these advances, and techniques developed here, such as extra virtual variables would enable algorithms such as OptAPO to tackle multi-criteria optimization. Previous work in multi-criteria collaboration [12, 13, 9, 10, 14] has focused on applications such as distributed planning, but it did not benefit from recent research that formalized DCOP and developed efficient algorithms for it. We build on these efficient algorithms leading to more efficient MCA algorithms. Approaches to multi-criteria constraint satisfaction and optimization problems have either tackled the problem using centralized methods [15] or have used distributed methods to solve a single agent problem [16]. Our approach tackles a distributed problem using a distributed method.

References

[1] Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence Journal **161** (2005) 149–180

[2] Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS. (2004)

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

[4] Yokoo, M.: Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems. Springer (2001)

[5] Ali, S., Koenig, S., Tambe, M.: Preprocessing techniques for distributed constraint optimization. In: International Joint Conference on Principles and Practices of Constraint Programming (CP). (2004)

[6] Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: International Joint Conference on Artificial Intelligence (IJCAI 05), Edinburgh, Scotland (2005)

[7] Lesser, V., Ortiz, C., Tambe, M.: Distributed sensor nets: A multiagent perspective. Kluwer Academic Publishers (2003)

[8] Meisels, A., Lavee, O.: Using additional information in discsps search. In: Distributed Constraint Reasoning Workshop (DCR). (2004)

[9] Murthy, S., Goodwin, R.: Cooperative multi-objective decision-support for the paper industry. Interfaces **29** (1999) 5–30

[10] Hanne, T., Nickel, S.: A multi-objective evolutionary algorithm for scheduling and inspection planning in software development projects. Institute for Technical and Economic Mathematics (ITWM) Technical Report **42** (2003)

[11] Maheswaran, R., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world : Efficient complete solutions for distributed event scheduling. In: AAMAS. (2004)

[12] Moraitis, P., Tsoukias, A.: A multicriteria approach for distributed planning and negotiation in multiagent systems. In: International Conference on Multiagent Systems. (1996)

[13] Fallah, A.E., Moratis, P., Tsoukias, A.: An aggregation-disaggregation approach for automated negotiation in multi-agent systems. In: MultiAgents and Mobile Agents. (2000)

[14] Matsatsinis, N., Delias, P.: A multi-criteria protocol for multi-agent negotiations. Lecture Notes in Computer Science **3025** (2004)

[15] Gavanelli, M.: An algorithm for multi-criteria optimization in CSPs. In: ECAI. (2002) 136–140

[16] Jaafar, I., Khayati, N., Ghedira, K.: Multicriteria optimization in CSPs: Foundations and distributed solving approach. In: AIMSA. (2004) 459–468