Optimize My Schedule But Keep It Flexible: Distributed Multi-Criteria Coordination for Personal Assistants

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

Abstract

Research projects have begun focusing on deploying personal assistant agents to coordinate users in such diverse environments as offices, distributed manufacturing or design centers, and in support of first responders for emergencies. In such environments, distributed constraint optimization (DCOP) has emerged as a key technology for multiple collaborative assistants to coordinate with each other. Unfortunately, while previous work in DCOP only focuses on coordination in service of optimizing a single global team objective, personal assistants often require satisfying additional individual user-specified criteria. This paper provides a novel DCOP algorithm that enables personal assistants to engage in such multi-criteria coordination while maintaining the privacy of their additional criteria. It uses n-ary NOGOODS implemented as private variables to achieve this. In addition, we've developed an algorithm that reveals only the individual criteria of a link and can speed up performance for certain problem structures. The key idea in this algorithm is that interleaving the criteria searches — rather than sequentially attempting to satisfy the criteria — improves efficiency by mutually constraining the distributed search for solutions. These ideas are realized in the form of *private-g and public-g Multi-criteria ADOPT*, built on top of ADOPT, one of the most efficient DCOP algorithms. We present our detailed algorithm, as well as some experimental results in personal assistant domains.

Introduction

Research projects have begun focusing on deploying personal assistant agents to coordinate users in such diverse environments as offices, distributed manufacturing or design centers, and in support of first responders for emergencies(Scerri *et al.* 2002; Maheswaran *et al.* 2004; SRI 2004). These personal assistant agents must often negotiate among each other in a collaborative fashion on behalf of their users, to coordinate their schedules, coordinate allocations of tasks and delivery of key results to each other, to jointly plan for future activities, etc. Distributed Constraint Optimization (DCOP) has emerged as a promising technology to enable such collaborative conflict resolutions. A DCOP includes a set of variables, where a subset of the variables is assigned to an agent who has control of the values of this subset. In the context of personal assistant agents, we can equip each agent with an efficient DCOP algorithms(Modi *et al.* 2003; Maheswaran *et al.* 2004; Yokoo *et al.* 1998), and the agents can then coordinate their choice of values so that a global objective function is optimized.

One central challenge in such coordinations among personal assistants however is that users may not be satisfied with a single global objective function to capture the utility of the group of users. For instance, users may wish to coordinate their schedules so that they optimize their performance on a set of tasks, but each user may impose additional requirements on his/her own schedule — one user may prefer that his/her own schedule have tasks be closely packed together, while another may prefer that the tasks be scheduled far apart from each other. Users may also have preferences over slack times awarded to them in case of failures, based on their own assessment of their capabilities or the possibility of failures due to environmental factors. However, current DCOP algorithms insist on a single global utility function, which does not allow multiple criteria to be taken into account during negotiations. While in simple cases it is feasible to combine the multiple criteria into a single global criteria, in complex cases, such an integration into a single criteria may be difficult and may lead to undesirable results.

In this paper we make some initial progress on the problem of multi-criteria DCOP. While there exist single-agent techniques for multi-criteria optimization, e.g., Constrained Markov Decision Processes (CMDPs) (Altman 1999), little work has focused on distributed multi-criteria constraint optimization. This paper provides two novel DCOP algorithms that enable personal assistants to engage in such multi-criteria coordination. Our goal is to allow for two objective functions: (i) a cost function which must be globally minimized and (ii) individual objectives at each agent which must be held below a pre-specified threshold. The first algorithm that we present employs n-ary NOGOODS to achieve this. A key advantage of this techinique is that it can keep the individual constraint functions for each link private and so we named it private-g Multi-Criteria Adopt. One drawback of this technique is that it can extensively search solutions that obviously violate a variable's private constraint before abandoning them. In order to speed up execution we developed a second algorithm, public-g Multi-Criteria Adopt,

that would not explore search space that violated a variable's individual constraint. The primary challenge is in deciding where in the partial solution space to search for a better assignment of values to variables. Since improvement in one measure does not necessarily correlate with an improvement in the other, there are two potential directions of search that must be balanced. To meet this challenge, we interleave the criteria searches — rather than sequentially attempting to satisfy the criteria — and this improves efficiency. In particular, each agent attempts to satisfy its local objective threshold and minimize the global cost given this threshold. Thus, we restrict the exploration of the search for a globally optimal solution to only those values that respect these prespecified individual thresholds. Agents may search through these individual thresholds to obtain better global solutions. Simultaneously, if the actual cost of the global solution given current thresholds is seen to be higher than other potentially lower cost solutions, then the agents will abandon adjusting their local thresholds and opportunistically jump to a new part of the search space. Thus, global information can also enable agents to limit local searches.

These ideas are realized in the form of *private-g and public-g Multi-Criteria ADOPT*, built on top of ADOPT, one of the most efficient DCOP algorithms (Modi *et al.* 2003). However, the techniques we describe could be applied to other DCOP algorithms. We have implemented private-g and public-g Multi-criteria ADOPT and we present results from the implementations.

Background: DCOP and Adopt

We focus on the use of DCOPs for negotiations among personal assistant agents, and in this section, we first present an overview of DCOP. Second, since we build up on ADOPT(Modi *et al.* 2003), a highly efficient DCOP algorithm (indeed, the most efficient DCOP algorithm that does not allow any centralization of value assignments), we also provide a brief overview of ADOPT. A Distributed Constraint Optimization Problem (DCOP) consists of n variables $V = \{x_1, x_2, \ldots, x_n\}$ which are assigned to a set of agents. A variable's value is controlled by the agent to which it is assigned and 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 an objective function is minimized or maximized. The objective function is an aggregation, usually the sum, over a set of constraints and associated cost functions. The cost functions are defined as $f_{ij} : D_i \times D_j \rightarrow N \cup \infty$. The objective is to find A, 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 shown in Figure 1a, $F((task1, 8am), (task2, 11am), (task3, 12pm), (task4, 2pm))$ = 4 and $F((task1, 9am), (task2, 11am), (task3, 1pm), (task4, 2pm))$ = 0. In this example A* would be $(task1, 9am)$, $(task2, 11am)$, $(task3, 1pm)$, $(task4, 2pm)$.

The original Adopt algorithm starts by organizing agents into a Depth-First Search (DFS) tree in which constraints are allowed between an agent and any of its ancestors or descendents, but not between variables in separate sub-trees.

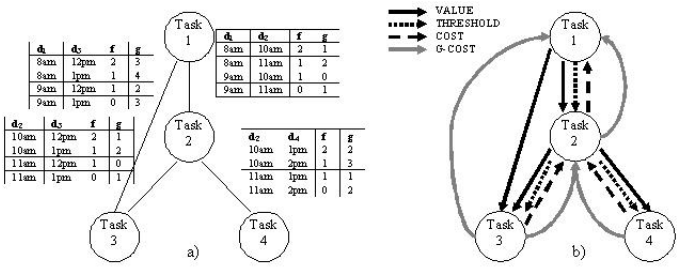

Figure 1: a) constraint graph b) MC-Adopt message passing

Communication in Adopt takes the form of 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 first starts, all nodes take on a random value and send out appropriate VALUE messages to get the flow of computation started. A THRESHOLD message is sent from parents to children and contains a backtrack threshold which is initially set to zero. The backtrack threshold indicates the maximum cost the child can accrue before suspending its line of search for another. A third type of message, the COST message, is sent from children to parents to indicate the cost of the sub-tree rooted at the child. These three forms of communication are shown in Figure 1 which is adapted from the diagram in (Modi *et al.* 2003). The pseudo-code for these message sending and receiving procedures is shown in the non-lettered lines (1) - (52) of Figure 2 and Figure 4.

Since communication in Adopt is asynchronous messages have a context attached to them, which is to say a list of the variable assignments in existence at the time that the message was sent, to allow the recipient to tell if the information is still relevant.

A sub-tree explores a line of search until the lower bound on cost accumulated at its root, which is defined as the lower bound cost of the sub-trees rooted at its children plus the cost of its constraints with its ancestors, surpasses the threshold assigned to it. When a node's lower bound surpasses its threshold, it will attempt to change its value to one that has a lower bound cost still under the threshold (all unexplored assignments begin with a lower bound of zero). If this is not possible an agent summarily raises its threshold and reports this to its parent. The parent then has to rebalance the thresholds of its other children. Similarly, an upper bound on cost is maintained and if a child discovers its threshold is greater than its upper bound it will unilaterally decrease its threshold.

The root's upper and lower bounds represent the upper and lower bounds on the global problem, so when they meet each other the optimal solution has been found and the algorithm terminates. This has been proven to be sound and complete. For more details see (Modi *et al.* 2003).

Problem Definition: Personal Assistants with Multiple Criteria

As illustrated in the previous section, current DCOP formulation and algorithms designed to solve problems within the formulation focus on a single objective function to minimize the cost function in a distributed manner. As illustrated in (Maheswaran et al 04), single-objective optimization can model problems such as optimizing allocation of tasks to multiple users, so that the user team obtains maximum utility from task performance. The result may be a schedule where for example a user *user1* may be required to do a task TASK1 at 8 AM in the morning, and another task TASK2 at 4 PM in the evening; whereas another user *user2* may be required to take on TASK1 at 9 AM (as soon as user1 finishes), and then a second task TASK2 at 10 AM (as soon as TASK1 finishes). However, users may prefer a schedule that in addition to optimizing the team performance, also addresses individual preferences. For instance, in the above scneario, users may prefer a schedule that either does not pack tasks together across users or within a single user's schedule by leaving more individual slack time — user2 may prefer at least half an hour between the end of one task and the start of the next task. Other preferences may include packing tasks closely together, e.g., user1 may prefer a schedule that does not schedule tasks only at the start and the end of the day (leaving a 7-8 hour gap in the middle of the day).

We define the multicriteria distributed optimization problem, building on the original DCOP, to address the above requirement. A Multi-Criteria Distributed Constraint Optimization Problem adds a second cost function $g_{ij} : D_i \times D_j \rightarrow N \cup \infty$ that is to be held below a threshold T.

A first pass multi-criteria DCOP defines g on the same inputs as f: Find A (= 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 $g_{ij}(d_i, d_j) \leq T$

This turns out to be trivially solvable by constructing:

$$f'_{ij}(d_i, d_j) = \{\infty \quad if g_{ij}(d_i, d_j) > T\}$$
$$= \{f_{ij}(d_i, d_j) \quad otherwise\}$$

A more complex multi-criteria DCOP defines g on a variable or more formally on the aggregation of the constraints impinging upon a particular variable: Find A (= 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 $\forall x_a \in V \sum_{x_b \in V} g_{ab}(d_a, d_b) \leq T$ where $x_a \leftarrow d_a, x_b \leftarrow d_b \in A$

This is the problem that is tackled in this paper by presenting a modified version of the DCOP solving algorithm Adopt. Our motivation for choosing to tackle this formulation was its ability to handle multicriteria problems of interest, e.g. in the task scheduling domain the team wants to maximize the number and utility of tasks scheduled while maintaining a minimum amount of flexibility per agent. This gives each agent the flexibility to recover from unforeseen delays without adversely affecting the rest of the team.

Multi-Criteria Adopt

Basic Idea

N-ary NOGOODS The key difficulty in our more complex multi-criteria DCOP problem is that the g-constraint is defined on a set of binary constraints, which can be expressed as an n-ary constraint. Since the second criteria must be satisfied, not optimized, we employ a techinique from DisCSP (Yokoo *et al.* 1998), namely NOGOODs. The algorithm searches for an optimal solution for f and when an assignment violates the g-constraint of a variable, a NOGOOD is added. Since the NOGOOD is enforced by the agent controlling the variable with the g-constraint, the constraint function is kept private.

Private-g Multi-Criteria Adopt Algorithm

In order to allow Adopt to handle multiple criteria, a preprocessing step was added as well as a mechanism for handling n-ary NOGOODs. The preprocessing modifies the tree structure and creates variables that will enforce the the n-ary NOGOODS. The n-ary NOGOOD variables each represent the g-constraint of a single variable and are owned by the owner of the variable they represent.

The preprocessing step modifies the tree structure to force all variables in a particular g-constraint to be in the same subtree (lines 0f-g in figure 2). It also creates an n-ary NOGOOD variable to enforce the constraint. The n-ary NOGOOD variables need to be placed lower in the tree structure than any of the variables in the g-constraint it enforces (lines 0a-e). The n-ary NOGOOD variables will thus only receive VALUE messages and will use these to determine whether the current assignment will violate the g-constraint being represented. If the constraint is violated, an infinite cost will be passed up, forcing the variables to try a different assignment, otherwise a 0 cost will be passed up (lines 52a-e).

In all other respects the algorithm behaves the same as the original Adopt algorithm.

Correctness of private-g

To prove correctness and completeness of private-g Multi-Criteria Adopt we need to prove 3 theorems: 1) LB and UB are correct, 2) Mulit-Criteria Adopt terminates, 3) the final threshold at a node is its cost. The proofs of the latter two items are identical to the originals in (Modi *et al.* 2003) and so they have been omitted. The proof that LB and UB are correct is an adaptation of the corresponding one in (Modi *et al.* 2003).

Property 1 $\forall x_i \in V$,
$OPT(x_i, CurrentContext)$ *def*
$min_{d \in D_i} \delta(d)$ + $\sum_{x_l \in Children} OPT(x_l, CurrentContext \cup (x_i, d))$
if $\sum_{x_j,x_k \in Neighbors} g_{jk}(d_j, d_k) \leq$ *g constraint*
∞ *otherwise*

Proposition 1 $\forall x_i \in V$, $LB \leq OPT(x_i, CurrentContext) \leq UB$

```
Preprocessing
(0a)  forall x_i with a g-constraint
(0b)     x'_i is a new n-ary variable
(0c)     x'_i linked to x_i
(0d)     forall x_k ∈ Neighbours(x_i)
(0e)        x'_i linked to x_i
(0f)        forall x_l ∈ Neighbours(x_i) not linked to x_k
               and not equal to x_k
(0g)           x_k linked to x_l
(0h)  buildTree(x_1 ... x_n)
(0i)  forall x'_i
(0j)     parent(x'_i) ← lowest priority Neighbour of x'_i

Initialize
(1)   threshold ← 0; CurrentContext ← {}
(2)   forall d ∈ D_i, x_l ∈ Children do
(3)      lb(d, x_l) ← 0; t(d, x_l) ← 0
(4)      ub(d, x_l) ← Inf; context(d, x_l) ← {}; enddo
(5)   d_i ← d that minimizes LB(d)
(6)   backTrack

when received (THRESHOLD, t, context)
(7)   if context compatible with CurrentContext:
(8)      threshold ← t
(9)      maintainThresholdInvariant
(10)     backTrack; endif

when received (TERMINATE, context)
(11)  record TERMINATE received from parent
(12)  CurrentContext ← context
(13)  backTrack

when received (VALUE, (x_j,d_j))
(14)  if TERMINATE not received from parent:
(15)     add (x_j,d_j) to CurrentContext
(16)     forall d ∈ D_i, x_l ∈ Children do
(17)        if context(d, x_l) incompatible with CurrentContext:
(18)           lb(d, x_l) ← 0; t(d, x_l) ← 0
(19)           ub(d, x_l) ← Inf; context(d, x_l) ← {}; endif; enddo
(20)     maintainThresholdInvariant
(21)     backTrack; endif

when received (COST, x_k, context, lb, ub)
(22)  d ← value of x_i in context
(23)  remove (x_i,d) from context
(24)  if TERMINATE not received from parent:
(25)     forall (x_j,d_j) ∈ context and x_j is not my neighbor do
(26)        add (x_j,d_j) to CurrentContext;enddo
(27)     forall d' ∈ D_i, x_l ∈ Children do
(28)        if context(d', x_l) incompatible with CurrentContext:
(29)           lb(d', x_l) ← 0; t(d', x_l) ← 0
(30)           ub(d', x_l) ← Inf; context(d', x_l) ← {};
               endif;enddo;endif
(31)  if context compatible with CurrentContext:
(32)     lb(d, x_k) ← lb
(33)     ub(d, x_k) ← ub
(34)     context(d, x_k) ← context
(35)     maintainChildThresholdInvariant
(36)     maintainThresholdInvariant; endif
(37)  backTrack

procedure backTrack
(37a) if x_i not an n-ary variable
(38)     if threshold == UB:
(39)        d_i ← d that minimizes UB(d)
(40)     else if LB(d_i) > threshold:
(41)        d_i ← d that minimizes LB(d)endif
(42)     SEND (VALUE, (x_i, d_i))
(43)        to each lower priority neighbor x_j
(44)     maintainAllocationInvariant
(45)     if threshold == UB:
(46)        if TERMINATE received from parent
(47)        or x_i is root:
(48)           SEND (TERMINATE,
(49)           CurrentContext ∪ {(x_i, d_i)})
(50)                                  to each child
(51)           Terminate execution; endif;endif
(52)     SEND (COST, x_i, CurrentContext, LB, UB)
            to parent
(52a) else
(52b)    if  Σ_{x_k ∈ Neighbours(x_i)} g(x_i, x_k) > gConstraint(x_i)
(52c)       SEND (COST, x_i, CurrentContext, ∞, ∞)
(52d)    else
(52e)       SEND (COST, x_i, CurrentContext, 0, 0)
```

Figure 2: Private-g Pseudo-code

Proof: by induction on variable priorities.

*B*ase Case: x_i is a leaf. This means that x_i is either an n-ary variable, or a variable not involved in any g-constraints. The latter case is the same as a leaf in the original Adopt algorithm. In the former case, LB and UB are defined to be 0 if the constraint is not violated and ∞ if it is. Thus $LB \leq OPT(x_i, CurrentContext) \leq UB$ for leaves.

*I*nductive Hypothesis: $\forall x_i \in V$ of depth k or greater, $LB \leq OPT(x_i, CurrentContext) \leq UB$

*I*nductive Step: x_i is a variable of depth k - 1. Since all children x_l have depth k or greater, $lb(d, x_l) \leq OPT(x_l, CurrentContext \cup (x_i, d_i)) \leq ub(d, x_l)$. Substituting into the definition of LB and UB we get:

$LB = min_{d \in D_i} \delta(d) + \sum_{x_l \in Children} lb(d, x_l) \leq$

$min_{d \in D_i} \delta(d) + \sum_{x_l \in Children} OPT(x_l, CurrentContext \cup (x_i, d)) \leq$

$UB = min_{d \in D_i} \delta(d) + \sum_{x_l \in Children} ub(d, x_l)$

and if $x'_i s$ g constraint is unsatisfied then one of its children will report the $lb(d, x_l) = \infty = ub(d, x_l)$ from $x'_i s$ n-ary node. ■

Speeding Up Multi-Criteria Adopt

Two drawbacks to the private-g approach to multi-criteria Adopt are its exploration of obviously unsatisfying assignments and the fact that its preprocessing introduces more chain-structure into the DFS tree, increasing the tree depth. Both of these cause the algorithm to run more slowly. If the g-function of an individual link were to be made public to the nodes connected to that link the runtime could

be improved. We choose to only explicitly (i.e. in non-amalgamated form) reveal the g-function information of a single link to those vertices connected to it, which is the same level of privacy loss tolerated for the f-function. With this additional information we can develop an algorithm (public-g) that will improve performance for tree-structured problems. However, graph-structured problems cannot be optimally solved without revealing additional information. An example of the problem applying public-g to a graph can be seen in Figure 3. A g-constraint of 1 is imposed on x_0 while x_1 and x_2 have a g-constraint of 2. If x_0 wants to decide how much g to allocate to its link with x_1 and its link with x_2 it considers the tables on each of these links. Assuming that all variables initially choose to take on 0, x_1 will report (after removal of double counts) that given a g of 0 or greater it will report an accumulated cost of 0. Based on the fact that x_1 has currently taken on 0, x_2 will report to x_0 that given a g of 0, it can report an accumulated cost of 4 (summing from both links) and given a g of 1 or more it can report a cost of 3. As a result, x_0 would assign a g of 0 to x_1 and a g of 1 to x_2. This would lead to the following assignment: $d_0 \leftarrow 0$ $d_1 \leftarrow 0$ $d_2 \leftarrow 0$. However, the optimal split is in fact for x_0 to assign a g of 1 to x_1 and 0 to x_2 allowing for the following assignment: $d_0 \leftarrow 0$ $d_1 \leftarrow 1$ $d_2 \leftarrow 1$. The former assignment results in a total cost of 3, whereas the optimal results in a total cost of 2. The reason that x_0 cannot discover the optimal split is that the function on the link between x_1 and x_2 is hidden to it. Without having this information explicitly revealed, x_0 cannot compute the optimal split. Since this kind of cycle cannot occur in trees, the public-g algorithm can be applied to trees but not to more general graphs.

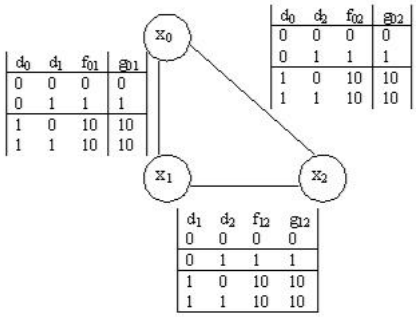

Figure 3: An example of public-g's failure on non-tree structures

Basic Idea

Interleaved Search The key difficulty in multi-criteria optimization is that improvement in one measure does not necessarily correspond with improvement in the other. This means that there are two directions in which an opportunistic search can proceed, so how does the algorithm choose which one to pursue? We tackle this problem by creating a threshold to force descendent nodes to only consider assignments that will not violate the g constraint of their ancestors. This is done by passing them each a personalized threshold specifying how large a g cost they can pass up to their ancestor. If a suitable division of g cannot be found then the algorithm will not try to optimize f since no solution would satisfy the g constraint. Thus we are using the g function to prune the search space in f of the algorithm.

Public-g Multi-Criteria Adopt Algorithm

In order to modify Adopt to more efficiently solve multi-criteria problems we applied the idea of thresholding to the g function. There are now two thresholds to be considered at each node. The first is the one that guides the search for an optimal f and has been described above. The second one is the g-threshold which specifies the maximum g cost allowed to be reported by a child to its ancestor. This requires a modification to the VALUE message to send g-thresholds and the creation of a new message, the G-COST message, for responding.

VALUE messages now have two additional fields: g-threshold and g-query (lines 15a,19e-19f in Figure 4). The parent sends all children a g-query with its full g-constraint value minus the g used on its links with its own ancestors (line 4e,19f) in order to collect information on how to best split the remaining g-threshold among its children. The parent then calculates the optimal split of its g among its children and fills in the g-threshold field with this value (lines 53-60). The g-threshold indicates that the agent receiving the message may take on no value whose g is greater than the g-threshold.

A G-COST message is sent from a child to the ancestor that sent it a g-query containing a table indicating for each potential g-threshold between 0 and the value in the g-query what the lowest f cost it could report would be (lines 61-67). This takes into account that values may have been eliminated from its domain because there is no way for it to divide its g-threshold among its children and satisfy them all (lines 53-60). Whenever a value is eliminated from a node's domain it sends a new G-COST message to its parent if it can no longer meet its g-threshold. Note that the removal of an element from the domain can only cause the f cost for that domain to increase because the reported f cost is supposed to be the minimum that maintains g below the potential g-threshold. This new G-COST message causes the parent to re-evaluate the best distribution of the g-thresholds (lines 37f-37k).

Since G-COST messages get reported up all ancestor links it is important to prevent double counting from affecting the split. Thus, each agent sends its name up to its parent in the G-COST message, if it has links with other ancestor nodes (lines 4f-4g) and it also passes along any variable names that it recieves in G-COST messages from its children (37b). When a variable receives a subtree list from its child x_k, it checks whether it is linked to any of the variables x_l in the list and if so stores that x_l is in the subtree of x_k (line 37b) and so when calculating the optimal split it subtracts the f's reported by x_l from those reported by x_k, thus eliminating the double counting (line 70).

Since n-ary NOGOODS are not required for the execution of public-g, the preprocessing steps are no longer required, which helps to create additional speedup because preprocessing tends to create more chain-like trees, which accord-

ing to Maheswaran et al (2004) causes the Adopt algorithm to run more slowly.

Correctness of public-g

Due to asynchrony in Adopt's execution it may happen that a node x_i must change its previous G-COST table to reflect changes in its domain. However, when the current contexts of the nodes in a subtree rooted at x_i are fixed, a condition which Jay Modi proved eventually occurs (Modi *et al.* 2003), the G-COST table can no longer change and so the division of g-Threshold at termination is guaranteed to be optimal.

Proposition 2 *If CurrentContext is fixed for node x_i and all of the nodes in the subtree rooted at x_i then x_i will not change its G-COST table at any point in the future.*

Proof. G-COST entry is $f(d_i)$ such that $d_i \in D_i$ and $\forall$ other $d_j \in D_i f(d_i) \leq f(d_j)$ and $g(d_i) \leq gThreshold$ Otherwise the entry is ∞. Node x_i will change an entry in its G-COST table iff $D_i \leftarrow D_i - d_i$ $D_i \leftarrow D_i - d_i$ iff there does not exist a g-Threshold distribution $g_1 \ldots g_k$ such that $\forall x_l \in children(x_i) x_l$ reports an f-cost $< \infty$.

Base Case: x_i is a leaf. Since x_i has no children it trivially has a g-Threshold distribution that satisfies all its children. Therefore its domain D_i is unchanging and this implies it will never change its G-COST table.

Inductive Hypothesis: If x_i is a node at depth k in the DFS tree and if CurrentContext is fixed for x_i and all of its descendents then x_i will never change its G-COST table.

We need to show that if the inductive hypothesis is true for all x_i of depth $\geq k$, it is true for all x_j of depth k - 1.

Since all descendents of x_j have a depth $\geq k$, then they will never change their G-COST tables by the inductive hypothesis. This implies that no further values can be eliminated from the domain of x_j. Therefore, the domain of x_j is fixed, and x_j will never change its G-COST table. ■

A node will try to find the optimal distribution of G-Threshold that causes all of its children to report f costs of less than ∞. If no such distribution exists then there is no solution to the multi-criteria DCOP with the ancestor node at its current value and so it will eliminate that value from its domain. However, if a distribution does exist then the node will select the distribution that minimizes the f costs reported by its children.

Experimental Results

Figure 5 demonstrates the effect of a varying g-constraint on the f-cost of the solution obtained with the private-g algorithm. The g-constraint was applied to each variable in the problem and the graph represents 10 runs of both tree and graph structured networks (5 of each). The looser the g-constraint, the lower the f-cost of the final solution. The case where the g-constraint is 40 is analagous to a cingle criterion optimization since the largest g-value on a single link was 10 and the greatest link density was 4.

Figure 6 demonstrates the effect of a varying g-constraint on the runtime of private-g. The runtime is measured in cycles and was generated by simulating the execution of the algorithm on a single processor. The data points represent

```
Initialize
(1)   threshold ← 0; CurrentContext ← {}
(2)   forall d ∈ D_i, x_l ∈ Children do
(3)     lb(d, x_l) ← 0; t(d, x_l) ← 0
(4)     ub(d, x_l) ← Inf; context(d, x_l) ← {}; enddo
(4a)  forall x_l ∈ Children do
(4b)    gThresh(x_l) ← 0
(4c)    gContext(x_l) ← {}
(4d)    gReportedBy(x_l) ← null; enddo
(4e)  availableG ← T − Σ_{x_i ∈ Ancestors} maxG(x_i)
(4f)  if x_i has a non-parent ancestor link
(4g)    subtree ← x_i; endif
(5)   d_i ← d that minimizes LB(d)
(6)   backTrack

when received (THRESHOLD, t, context)
See figure 2

when received (TERMINATE, context)
See figure 2

when received (VALUE, (x_j,d_j,g_query,g_thresh))
(14)  if TERMINATE not received from parent:
(15a)   add (x_j,d_j,g_thresh) to CurrentContext
(16)    forall d ∈ D_i, x_l ∈ Children do
(17)      if context(d, x_l) incompatible with CurrentContext:
(18)        lb(d, x_l) ← 0; t(d, x_l) ← 0
(19)        ub(d, x_l) ← Inf; context(d, x_l) ← {}; endif; enddo
(19a)   forall x_l ∈ Children do
(19b)     if gContext(x_l) incompatible with CurrentContext:
(19c)       gThresh(x_l) ← 0;
(19d)       gContext(x_l) ← {}; endif; enddo
(19e)   gTbl(x_j) ← calcTbl(g_query)
(19f)   availableG ← T − Σ_{x_i ∈ Ancestors} maxG(x_i)
(20)    maintainThresholdInvariant
(21)    backTrack; endif

when received (COST, x_k, context, lb, ub)
See figure 2

when received (G-COST, x_k, context, tbl, sub)
(37a) d ← value of x_i in context
(37b) merge sub into subtree
(37c)   forall x_l ∈ children do
(37d)     if x_l ∈ sub
(37e)       gReportedBy(x_l) ← x_k; endif; enddo
(37f) if TERMINATE not received from parent:
(37g) if context compatible with CurrentContext:
(37h)   gInfo(x_k) ← tbl
(37i)   gContext(x_k) ← context
(37j)   calcOptimalSplit; endif; endif;
(37k) backTrack
```

```
procedure backTrack
(38)  if threshold == UB:
(39a)    d_i ← d that minimizes UB(d) and satisfies gMax
(40)  else if LB(d_i) > threshold:
(41a)    d_i ← d that minimizes LB(d) and satisfies gMax;endif
(42a) SEND (VALUE, (x_i, d_i, availableG, gThresh(x_j)))
(43)     to each lower priority neighbor x_j
(44)  maintainAllocationInvariant
(45)  if threshold == UB:
(46)     if TERMINATE received from parent
(47a)    or x_i is root and ∀x_l ∈ childrengReplied(x_l) is true:
(48)        SEND (TERMINATE,
(49)        CurrentContext ∪ {(x_i, d_i)})
(50)           to each child
(51)        Terminate execution; endif;endif
(52)  SEND (G-COST, x_i, CurrentContext, LB, UB)
         to parent
(52a) SEND (G-COST, x_i, CurrentContext, gTbl(x_k), {})
         to all ancestors x_k except parent
(52b) SEND (G-COST, x_i, CurrentContext, gTbl(parent), subtree)
         to parent

procedure calcOptimalSplit
(53)  remove double counts
(54)  f_best ← infty
(55)  for g_0 ← 0 to availableG
            ⋮
(56)     for g_{k-1} ← 0 to availableG − g_{k-2} − ... − g_0
(57)        g_k ← availableG − g_{k-1} − ... − g_0
(58)        if gInfo(x_0)[g_0] + ... + gInfo(x_k)[g_k] < f_best
(59)           gThresh(x_0) ← g_0 ... gThresh(x_k) ← g_k
(60)           f_best ← gInfo(x_0)[g_0] + ... + gInfo(x_k)[g_k]

procedure calcTbl (gQuery)
(61)  sort d_i ∈ D_i by increasing LB(d_i)
(62)  for x = gQuery to 0 do
(63)     while d_1 doesn't satisfy a g-threshold of x remove d_1
(64)     if d_i not null
(65)        gTbl[x] ← LB(d_1)
(66)     else
(67)        gTbl[x] ← infty
```

Figure 4: Public-g Pseudo-code

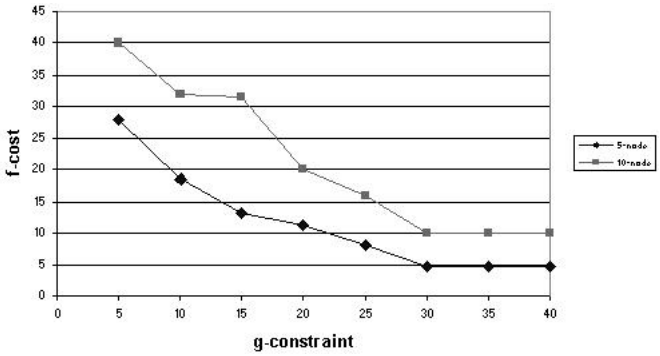

Figure 5: Measuring the global cost f as a function of local thresholds g

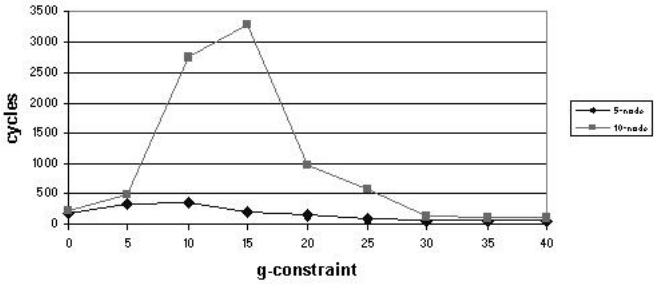

Figure 6: Measuring run time in cycles as a function of the threshold g.

10 runs of both tree and graph structured networks. The runtime is lowest when the search is either unconstrained due to a large g-constraint or so constrained that little search space needs be explored.

Figure 7 shows the relative runtimes of the private-g and public-g algorithms on a set of tree structured graphs. The graph is lograrithmically scaled and demonstrates that public-g decreases the runtime by a factor of 5 on average for 5-node trees and 27 for 10-node trees. The quality of solution is unaffected.

Related Work

Previous work in collaborative multicriteria negotiation (Moraitis and Tsoukias 1996)(Fallah *et al.* 2000) has focused on applications such as distributed planning, but this work did not benefit from the recent research that formalized DCOP representations and developed efficient algorithms for it. The key difference between our work and this previous work is that we build on these efficient algorithms, in particular, ADOPT, leading to a more efficient MC-ADOPT algorithm.

Conclusion

In many applications, personal assistant agents must negotiate over multiple criteria, where they must not only attempt to optimize a global team objective, but they must simultaneously meet additional user-specified criteria. In previous work, distributed constraint optimization (DCOP) has

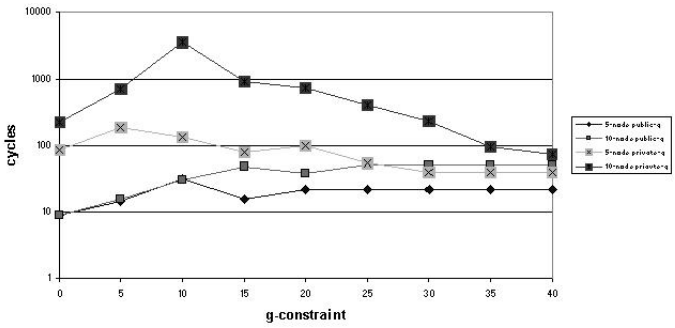

Figure 7: Comparing the runtime in cycles as a function of g

emerged as a key technology for multiple collaborative assistants to negotiate with each other, as it enables such negotiations while protecting user privacy. Unfortunately, previous work in DCOP only focuses on negotiation in service of optimizing a single global team objective. This paper provides 2 novel DCOP algorithms that enable personal assistants to engage in multi-criteria negotiation. TThe first algorithm employs n-ary NOGOODS and maintains the individual criterion private. The second interleaves the searches for the multiple criteria — rather than sequentially attempting to satisfy the criteria — and improves efficiency. These ideas are realized in the form of *public-g and private-g Multi-criteria ADOPT*, built on top of ADOPT, one of the most efficient DCOP algorithms. We present our detailed algorithm, as well as some initial experimental results in personal assistant domains.

Acknowledgement

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

E. Altman. *Constrained Markov Decision Process*. Chapman and Hall, 1999.

A. El Fallah, P. Moratis, and A. Tsoukias. An aggregation-disaggregation approach for automated negotiation in multi-agent systems. In *Proceedings of the MAMA2000 conference*, 2000.

R. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking dcop to the real world : Efficient complete solutions for distributed event scheduling. In *Proceedings of the Third International Joint Conference on Agents and Multiagent Systems (AAMAS'04)*, 2004.

P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Agents and Multiagent Systems (AAMAS'03)*, 2003.

P. Moraitis and A. Tsoukias. A multicriteria approach for distributed planning and negotiation in multiagent systems. In *International Conference on Multiagent Systems*, 1996.

P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real world. *Journa of Artificial Intelligence Research*, 17:171–228, 2002.

SRI. *Calo.sri.com*. San Jose, 2004.

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