The Multi Variable Multi Constrained Distributed Constraint Optimization Framework

Christopher Portway and Edmund H. Durfee
University of Michigan, Ann Arbor MI 48105, USA,
portway@umich.edu, durfee@umich.edu

Abstract. Service coordination in domains involving temporal constraints and duration uncertainty has previously been solved with a greedy algorithm that attempts to satisfy service requests one by one rather than reasoning over their combinations to deal with intractable problems. We propose preprocessing using a model that abstracts away duration uncertainty, so that the simplified problem can be solved faster and suggest which requests should be addressed first based on reward and satisfiability. We define a new generalization of MC-DCOP, called MV-MC-DCOP, and use this as our model. We present a Mixed Integer Linear Program, a Partially Centralized MV-MC-DCOP, and a fully decentralized MV-MC-DCOP approach to this preprocessing, and analyze the strengths and weaknesses of each approach.

1 Introduction

A challenge in developing joint policies for interacting agents, in applications such as teaming service-oriented agents together to accomplish a task, is that there are often more potential interactions than feasible interactions. That is, some service-providing agents might be unable to satisfy all requests. However, discovering which combinations of service requests are infeasible by attempting to generate detailed joint policies can be extremely time-consuming because agents might have to consider an exponential number of state trajectories that differ in small ways. To avoid such intractable search, suboptimal greedy service provisioning techniques have been resorted to[1].

Our work is investigating the use of distributed constraint optimization techniques as a preprocessing step that prunes the space of joint policies to search by ruling out (or in) candidate combinations of service requests. The idea is to abstract away details of stochastic features such as duration uncertainty, and to perform distributed constraint reasoning over the simplified problem. If the simplifications are done well, the solution(s) to the distributed constraint optimization problem can seed the joint policy search to allow much more rapid convergence.

We consider the MC-DCOP algorithm [2] as a logical starting point for our preprocessing model, due to its simplicity and separation of constraints into a maximizing function and a limiting function. For example, when the problem instance model described in the next section is simplified by abstracting away the probabilistic aspects of time to completion, we can use time to completion as the limiting function, while maximizing on the reward given to service-providing agents (providers) by service-requesting agents (requesters).
While the separation of a limiting function in constraints in MC-DCOP makes it appealing, it is still a little too simple for our purposes. We propose a variant of this approach that we call Multi-Variable MC-DCOP (MV-MC-DCOP), which is a more generalized framework that better models our domain. Specifically, the traditional assumption that each agent controls only a single variable is removed in MV-MC-DCOP, as described in detail below. This is done in order to more accurately express the cumulative costs of variables within a more realistic agent. This also creates two levels of satisfaction: variable and agent. In some cases, it is useful to consider an agent satisfied even if some of its internal variables remain unsatisfied. MV-MC-DCOP also represents this added issue with an additional minimum satisfaction (minSatisfied) parameter.

We will begin by describing the motivating domain and what assumptions and transformations we have made to generate the abstracted formulation we address. We will then illustrate the usefulness of the pre-processing step through an example in which the greedy search for policies will not be able to satisfy the last requesting agent, while pre-processing with MV-MC-DCOP can result in seeding the policy search in a way that results in higher global utility. The evolution from DCOP to MC-DCOP to the proposed MV-MC-DCOP is then presented. We continue by discussing three solvers for our new problem formulation: a Mixed Integer Linear Program (MILP), a Partially Centralized complete DCOP algorithm, and an incomplete algorithm based on modifications to the MC-MGM algorithm. Finally, we compare runtime and quality results of the three approaches and conclude with a discussion of the advantages of pre-processing.

2 Motivation

In our motivating domain of service coordination, there are conceptually 2 types of agents, providers and requesters. Each agent has internal tasks, which can depend on the prior completion of other tasks, either internal or external to the given agent. An agent is considered satisfied when any of a group of subsets of its internal tasks are completed. The task durations can be uncertain, and several task reward structures may exist.

Rewards can be internally or externally based. For a requester, its own reward could be per-task or a single reward for completing a satisficing set of tasks, but is always internally based. Similarly, a requester can elicit help from providers by assigning external reward to the desired individual or groups of tasks of providers. In addition, providers can likewise have internal per-task or satisficing rewards.

2.1 Restricting the Domain of Service Coordination Problems

In this paper, we make five simplifying assumptions and transformations of this domain to enable mapping of the complex service coordination problems into a simple formulation which can be solved quickly to seed the search space. Future work will attempt to relax some of these restrictions. In this paper these restrictions are:

1. Single-level requester-provider relationships
2. Simple satisficing task subset function
3. Simple static reward structures
4. No task duration uncertainty
5. No temporal ordering of task completion

We will look in this paper at problems representing a single-level requester-provider relationship - in other words there are no middlemen or sub-contractors. This assumption allows easy mapping of a subset of real-world problems to DCOP, and ultimately will be relaxed in future work.

We assume that all internal tasks are considered equal in terms of satisfying requesters, and therefore a single number, minSatisfied, can represent the number of tasks which must be satisfied for a given requester to be considered satisfied. When this is not the case, heuristic transformations can be performed. Providers are inherently satisfied as they are not requesting any service.

For our implementation, we ensure that agents will have trivial reward structures; specifically, requesters will always commit to receiving a response to each request from every applicable provider. In other words, requesters only ask for things they are committed to assign utility for. All utilities are symmetric between a requester and a provider, static, and known a priori. While utilities are per-task, they are only earned if the requester involved is satisfied. As with the single-level requester-provider relationship, this assumption allows an easier mapping to DCOP, and ultimately should be relaxed for greater accuracy.

Task duration uncertainty can be removed in several ways. A pessimistic or optimistic transformation could take the longest or shortest possible duration, respectively, some type of average could be calculated, or a uncertainty threshold could be used.

Temporal ordering is removed by simply reasoning over total time taken by chains of dependant tasks, without regard to order of execution. This can lead to finding infeasible solutions, but it is a key restriction which allows mapping to a simple MC-DCOP derived formulation.

Figure 1 shows a portion of an example graph in the unrestricted motivating domain, with two requesters and one provider visible.

![Figure 1. Partial View of a Service Coordination Problem in the Unrestricted Domain](image-url)
In order to represent this graph into our MV-MC-DCOP domain, we must do 3 transformations. First, restrictions 1 and 5 require that tasks V-2 and V-3 be merged, by setting the new super-task’s duration and utility to the sum of the individual utility and concatenated duration tables, respectively. Second, restriction 2 requires that the satisficing conditions of R3 be simplified. If we set the simplified R3’s minSatisfied = 2, then R3 is satisfied whenever our simplified R3 is satisfied. Third, restrictions 3 and 4 require a single duration for P4’s task V-1 be chosen, as with any other tasks with uncertain duration. There is opportunity here to use various heuristics, such selecting based on acceptable uncertainty. If we choose a pessimistic approach and only accept 100% certainty, then the duration of P4’s V-1 will be locked at 5, and the utility will become 10.

2.2 Example Within the Restricted Domain

Let us take a simple example of 3 agents needing surveillance images, with 3 surveillance satellites known between them. Thus there are 3 requesters and 3 providers, which we will label R1, R2, R3 and P4, P5, P6, respectively. In our example, the budget constraint will be length of time spent taking a picture. Better quality pictures require more time to zoom and focus. Each Provider can take two low quality images or one high quality image in the given time horizon. R1 requires at least 2 high quality images from any of the providers. R2 only needs 1 high quality image, but P4 is in a bad position, given R2’s requirements, so R2 can only obtain what it needs from P5 or P6. R3 needs 2 low quality images, from any P. If each image is worth 5 units of utility to R1, and 20 units of utility per image for both R2 and R3, then one optimal solution is for R2 to be provided by P5 while R3 is provided by P4 and P6, resulting in a global utility of 60 and two satisfied requesters.

A greedy solver, on the other hand, may have only earned a global utility of 30. If a greedy algorithm attempts to satisfy requesters in increasing label order, one of the greedy solutions would be R1 is provided by P4 and P5, netting a utility of 10, and R2 is then satisfied by P6, contributing an additional 20 utility units.

The goal of MV-MC-DCOP preprocessing is to guide the search for an efficient joint policy using a more accurate model, for example to avoid results such as the local optimum of the preceding paragraph.

3 Background

3.1 DCOP

Prior work [2] defines a DCOP as a set of variables (one per agent) \(\mathcal{N} := \{1, \ldots, n\}\) and a set of domains \(\mathcal{A} := \{A_1, \ldots, A_n\}\), where the \(i^{th}\) variable takes value \(a_i \in A_i\). We denote the joint action (or assignment) of a subgroup of agents \(S \subset \mathcal{N}\) by \(a_S := \times_{i \in S} a_i \in A_S\) where \(A_S := \times_{i \in S} A_i\) and the joint action of the multi-agent team by \(a_\mathcal{N} = [a_1 \cdots a_n]\). (Since DCOP approaches usually assume each agent controls a single variable, we will use the terms “agent” and “variable” interchangeably until the distinction is needed.)
Valued constraints exist on various subsets $S \subset N$ of DCOP variables. A constraint on $S$ is expressed as a reward function $R(a_S)$. This function represents the reward to the team generated by the constraint on $S$ when the agents take assignment $a_S$. We refer to these subsets $S$ as “constraints” and the functions $R(\cdot)$ as “reward functions”. A DCOP can be depicted graphically with each node representing an agent and each edge representing a constraint (or hyper-edge in the case of constraints on more than two agents). Given that DCOPs are motivated by agents’ privacy and communication overheads, non-neighboring agents in DCOPs do not communicate with (or directly reveal their values to) each other. The solution quality for a particular complete assignment $a_N$, $R(a_N)$, is the sum of the rewards for that assignment from all constraints (captured in the set denoted by $\theta$) in the DCOP: $R(a_N) = \sum_{S \in \theta} R(a_S)$.

### 3.2 MC-DCOP

The DCOP formulation has previously been extended to cover Multiply Constrained DCOP (MC-DCOP) problems, where each constraint now has a cost function in addition to the existing reward function. The sum of the cost function must be lower than the given agent’s Budget value.

More formally, we define a new secondary function on constraints on $S$. A constraint on $S$ is now represented by two functions, $R(a_S)$ and $C(a_S)$. The function $C(a_S)$ represents the cost under the assignment $a_S$, which is incurred by each agent owning a variable $i \in S$. We define a function $J(i)$ which returns a set consisting of all joint assignments $a_S$ which involve variable $i$. Note that, by setting some or all $C(a_S) = 0$, individual agents or entire graphs revert to the standard DCOP form.

The agent (and variable) $i$ has a satisfied budget when $\sum_{a_S \in J(i)} C(a_S) \leq B_i$ where $B_i$ is the budget of the agent $i$.

### 3.3 MV-MC-DCOP

There are 2 changes from MC-DCOP to Multiple Variable MC-DCOP (MV-MC-DCOP). The first is the removal of the assumption that each agent controls one variable, allowing a Budget to span across multiple variables within an agent.

More formally, let $I_g \in N$ be the set of variables controlled by agent $g$. We redefine the conditions for a satisfied budget by summation over all variables $i \in I_g$ controlled by agent $g$. Note that, by setting some or all agents to representing a single variable, individual agents or entire graphs revert to MC-DCOP form, and by additionally setting $C(a_S) = 0$, they revert to the standard DCOP form.

Then the agent $g$ has a satisfied budget when $\sum_{i \in I_g} \sum_{a_S \in J(i)} C(a_S) \leq B_g$.

The second change is the addition of an integer constant representing the minimum number of satisfied internal variables for the given agent to be considered satisfied overall. We label this $\text{minSatisfied}$. When $\text{minSatisfied} = 1$, then an OR relation is created; when $\text{minSatisfied} = |I_g|$, there is an AND relation. Other intermediate values may also be useful, such as $\text{minSatisfied} = 2$ when a “second opinion” for redundancy is required. The satisfaction function has a boolean value, representing satisfied or not: $\text{Satisfied}(g) = (\sum_{i \in I_g} \sum_{a_S \in J(i)} \text{isValid}(a_S) \geq \text{minSatisfied})$. Thus, the previous
global reward function, \( R(a) \) now becomes a function of local Satisfied reward functions, such that the global reward is the total reward gained by each variable within each satisfied agent. More formally, 
\[
R(a) = \sum_{g \in G} \sum_{i \in I_g} \sum_{a_S \in J(i)} \text{Satisfied}(g) \times R(a_S).
\]

4 Mapping Service Coordination

We can represent our specific service coordination domain within the MC-MV-DCOP framework, with a few specific patterns which will be exploited by the solvers described later in this paper.

While each agent/node can be considered the same from the framework standpoint, we represent the requester-provider relationship notationally by lettering as well as number nodes. In general Requesters will have infinite budgets and a non-zero minSatisfied value, while Providers will have a finite budget and a zero minSatisfied value.

In Figure 2, you can see an example graph with three requesters, R1, R2, and R3, along with three providers, P4, P5, and P6, and tables for the C and R functions. In our example, the C and R function tables are the same for all constraint edges. In both tables, the value taken by the Provider is on the y-axis, and the Requester on the x-axis. Assuming each Provider has a budget of 5, and each Requester a minSatisfied of 2, only one requester can be satisfied since each variable/task in a provider takes the entire budget, and each requester requires two tasks from a provider due to minSatisfied.

![Service Coordination framed in MV-MC-DCOP](image)

5 Mixed Integer Linear Programming - Centralized Solution

We have formulated a Mixed Integer Linear Program which performs a simplex search of a graph in the MV-MC-DCOP framework to find the globally optimal solution. We show in the Results section that this would not scale well to large problems, but can serve as a baseline to compare heuristics on graphs of small to medium size.
In this MILP, the sets $G$ and $I_g$ are used as before, and the set $V$ is used as the set of possible value assignments to a single internal variable, in contrast to $a_S$ which is a joint assignment between the subset of variables $S$. Similarly $V\text{Valid}$ is the set of $V$ which count towards satisfying an agent. In the MILP, Joint is a matrix that confirms whether the joint assignment exists or not, while Single is the same for a single variable. Also, Satisfied represents whether an individual agent is satisfied.

Constraints 1 and 2 in this MILP represent the constraints needed to solve a standard DCOP, if we were to remove the index $i$ and replace $\text{SatJoint}$ in the maximization function with Joint. We use matrices of binary variables to represent the single assignments, with each variable indexed by agent, internal variable, and assigned value, respectively. Thus constraint 1 simply ensures only one assigned value is marked as true for a given variable. Since we model DCOP with binary constraints, MILP constraint 2 allows the MILP to reference pairs of assignments in the matrix Joint.

The addition of MILP constraint 3 allows the MILP to solve MC-DCOP graphs, under the same changes to the maximization function. Constraint 3 ensures the sum of the cost of all internal variables of each agent is within that agent’s budget.

Thus, MILP constraints 4 through 7 are the ones new or altered for MV-MC-DCOP. Constraint 4 ensures the boolean array Satisfied is filled with a representation of whether the indexed agent has enough internal variables assigned to values other than the dummy starting value. Constraints 5 through 7 ensure the matrix $\text{SatJoint}$ holds a value of 1 if its indices are a chosen joint assignment in a satisfied agent, and 0 otherwise.

Note that the formulation of constraint 4 causes the MILP solver to produce rounding errors if any agent has over 198 times as many variables assigned as needed to be satisfied. This is due to integer rounding imprecision in CPLEX and AMPL, the MILP solving suite we are using. For our purposes, this is an acceptable limitation, but should be addressed if this MILP is used as the end product.

---

**Algorithm 1 MV-MC-DCOP Mixed Integer Linear Program**

max( $\sum_{g \in G} \sum_{v \in V} \text{Single}[g, i, v] \cdot \text{SatJoint}[g, i, 1, v, g, 2, i, 2, v, 2] + \text{R}[g, i, 1, v, 1, g, 2, i, 2, v, 2]$ )

subject to:

1. $\forall g \in G, \forall i \in I_g, \forall v \in V, \text{Single}[g, i, v] = 1$
2. $\forall g \in G, \forall v \in V, \forall i \in I_g, \forall v \in I_g, \forall v \in V, \forall v \in V$
   - $\text{Single}[g, i, 1, v, 1] + \text{Single}[g, 2, i, 2, v, 2] - 1 \leq \text{Joint}[g, i, 1, v, 1, g, 2, i, 2, v, 2] \leq \forall (g, i, v) \in \{(g, i, 1, v, 1), (g, i, 2, v, 2)\}$ $\text{Single}[g, i, v]$
3. $\forall g \in G, \forall v \in V, g \in G, \forall g \in G, \forall g \in G, \forall g \in G$
   - $\text{SatJoint}[g, i, 1, v, 1, g, 2, i, 2, v, 2] \cdot \text{Connected}[g, i, 1, g, 2, i, 2, v, 2] \cdot \text{Connected}[g, i, 1, g, 2, i, 2, v, 2] \cdot \text{Connected}[g, i, 1, g, 2, i, 2, v, 2] \cdot \text{Connected}[g, i, 1, g, 2, i, 2, v, 2]$
4. $\forall g \in G, \forall v \in V, g \in G, \forall g \in G, \forall g \in G, \forall g \in G$
   - $\text{Satisfied}[g] \leq \left( \sum_{(g, i, v) \in V \text{Val}} \text{SatJoint}[g, i, v] \cdot \text{SatJoint}[g, i, v] \right) / (\text{minSat}[g] + 1)$
5. $\forall g \in G, \forall v \in V, g \in G, \forall g \in G, \forall g \in G, \forall g \in G$
   - $\text{SatJoint}[g, i, 1, v, 1, g, 2, i, 2, v, 2] \leq \forall g \in \{g, 1, g, 2\} \text{Satisfied}[g]$
6. $\forall g \in G, \forall v \in V, g \in G, \forall g \in G, \forall g \in G, \forall g \in G$
   - $\text{Joint}[g, i, 1, v, 1, g, 2, i, 2, v, 2] \leq \forall g \in \{g, 1, g, 2\} \text{Satisfied}[g]$
7. $\forall g \in G, \forall v \in V, g \in G, \forall g \in G, \forall g \in G, \forall g \in G$
   - $\text{SatJoint}[g, i, 1, v, 1, g, 2, i, 2, v, 2] + \text{Satisfied}[g, 1] + \text{Satisfied}[g, 2] - 2$
6 MV-MC-DCOP Partially Centralized Protocol

We will be searching for the complete solution starting from a centralized DCOP approach which in the worst case enumerates all assignments after individual agent budget-based pruning, but may terminate earlier depending on thresholds. Other pruning and enhancements may be possible when a threshold is used, due to graph structure and domain knowledge. A new node labeled C is introduced as the centralized controlling agent. This agent controls the assignments of the other nodes, but learns only the total utilities of P nodes, not the utilities of each internal variable, nor internal budget allocations.

6.1 Partially Centralized Complete Protocol

Figure 3 shows an example graph structure in the Partially Centralized method. The C node makes all the decisions, while R and P nodes react based on internal constraints. Algorithm 2 describes the high level procedure.

At the beginning of each complete run of the algorithm, each P-node does a preprocessing step of finding all possible allocations of internal variables that are within the agent’s gbudget. This number, which indexes internal variable joint assignments, is sent to node C by each node P.
In each round of the algorithm, node C will send an assignment index to each corresponding P node, thus trying a new cross product of assignments in the P nodes, and R nodes will change their internal variables’ values to the corresponding variables’ values in the P nodes. Each round, the P nodes each send their total utility to node C. Internally the R nodes count the number of variables in $I_g$ which have left the starting invalid dummy value for the corresponding P node, and subtract this number from minSatisfied resulting in the satisfaction-number(SN). As previously mentioned, any agent with SN $\leq 0$ is satisfied.

### 6.2 Partially Centralized Threshold Protocols

The simplest method to guarantee finding the global optimum is to iterate through the cartesian product of all assignments node C can make from P nodes, as in Algorithm 2 above. Another option is to use heuristic guided search, which should make the anytime property of this approach more valuable. One choice for this is shown in Algorithm 3. This approach can be terminated early internally from thresholds or externally at a time limit. We have only implemented the naive approach at this point, and leave the implementation and testing of these methods as future work.

There are two intuitive thresholds which can be set, along with a third hybrid of the first two.

1. satisfaction of all, or a certain percentage, of R nodes
2. satisfactory numeric utility
3. satisfaction of a percentage of R nodes, with a satisfactory numeric utility

The first threshold leads to a logical heuristic in which after the initial assignment, only a subset of the providers with non-zero SN values will be asked to reassign their internal variables. This requires adding new information to the messages sent by the R nodes to the C node in the first round to establish which providers it is connected to. The simplest implementation of this is that the C node will randomly reassign some number of corresponding P nodes, totaling less than $|SN|$ for each negative SN. This could cause more movement than necessary, since the reassignment of a single P node can remove and add provision for multiple R nodes. This also does nothing to protect R nodes already at the ideal state of SN = 0, but it is likely to be an improvement over blind search.

This initial heuristic could be adjusted in several ways. For one, the C agent could send a change request message to the P nodes, and to which the P nodes will reply
Algorithm 3 PartCent-MV-MC-DCOP (exploration, providerChange) [heuristic]

1: GetValidAssignments(allProviders)
2: currentGlobalAssignment = RandomUntriedCrossProductOfProviderAssignments()
3: for provider in Providers do
4: AssignValue(provider, currentGlobalAssignment)
5: currentGlobalUtility = receiveUtilities(allProviders)
6: bestGlobalAssignment = currentGlobalAssignment
7: TriedAssignments.add(currentGlobalAssignment)
8: for requester in requesters do
9: possibleProviders[requester] = receiveListOfProviders(requester)
10: satisfied[requester] = receiveSatisfiedStatus(requester)
11: while thresholdNotMet do
12: if RandomProbability < exploration then
13: currentGlobalAssignment = RandomUntriedCrossProductOfProviderAssignments()
14: else
15: for requester in requesters do
16: if satisfied[requester] = NotSatisfied then
17: ProviderCandidatesForChange.add(possibleProviders[requester])
18: for provider in ProviderCandidatesForChange do
19: if RandomProbability < providerChange then
20: AskForBestChange(provider)
21: receiveNextAssignment(provider)
22: currentGlobalAssignment = internalViewOfEachProviderAssignment
23: if (currentGlobalAssignment ∈ TriedAssignments) then
24: findUntriedAssignment(currentGlobalAssignment, ProviderCandidatesForChange)
25: TriedAssignments.add(currentGlobalAssignment)
26: if currentGlobalUtility > largestGlobalUtility then
27: bestGlobalAssignment = currentGlobalAssignment
28: currentGlobalUtility = receiveUtilities(allProviders)
29: if currentGlobalUtility > largestGlobalUtility then
30: bestGlobalAssignment = currentGlobalAssignment

with an assignment based on their consideration of the best next move in terms of local utility. The C node would then send a nogood message to a subset of the changing P nodes if the resulting joint assignment across all P nodes as viewed by the C node has already been attempted.

7 MV-MC-MGM Incomplete Local Search Protocol

In the preceding approaches to solving MV-MC-DCOP, we have been able to guarantee optimal solutions, or an arbitrary lower bound as a parameter, given that a viable solution exists. In this section we describe how we have also extended the existing MC-MGM algorithm to solve MV-MC-DCOPs. We call this new algorithm MV-MC-MGM, which is a local search algorithm for which there are not yet any provable lower bounds (besides 0) on the solution quality. This local search technique is also explored in hopes it can scale beyond what the previous algorithms can handle.
Algorithm 4 describes our implementation of MV-MC-MGM, which overall takes a decomposition approach (see [3] for tradeoffs in approaches), although it assumes shared knowledge of the overall agent budget, which is assigned to individual variables and is treated as a part of currentContext.

**Algorithm 4 MV-MC-MGM-1 (allNeighbors, currentValue)**

1: SendValueMessages(allNeighbors, currentValue, available-budget)
2: currentContext = GetValueMessages(allNeighbors)
3: for var in InternalVariables do
4:   for newValue(var) in EffectiveDomain(currentContext) do
5:     [gain(var),newValue(var)] = BestUnilateralGain(currentContext)
6:     if gain(var) > 0 OR (curSatisfied < minSatisfied AND unsatisfied(var)) then
7:       SendGainMessage(allNeighbors, gain(var), newValue(var))
8: neighborGains = receiveGainMessages(allNeighbors, NeighborValues)
9: if BudgetViolated(newContext) then
10:   n = SelectNeighborsToBlock()
11:   SendBlockMessages(n)
12: for var in InternalVariables do
13:   if gain(var) > max(neighborGains(var)) and !receivedBlockMessage() then
14:     currentValue(var) = newValue(var)

In more general terms, in MV-MC-MGM each variable calculates its best move given the values taken by neighboring variables and the budget allocated by the agent it belongs to, and broadcasts the gain it would receive for this move. If none of its neighbors would gain more utility, then that variable moves. The MC-MGM algorithm was chosen as the basis of our incomplete approach because it has nice properties such as monotonically increasing global utility, and good lower bounds on DCOP and a subclass of MC-DCOP graphs. Thus it will likely be possible to prove useful lower bounds on MV-MC-MGM in future work.

8 Experimental Results

We randomly generated 6 sets of MV-MC-DCOP instances based on 4 factors: increasing the number of agents, the number of internal variables per agent, the link density, and the per agent budget. All of the instances had purely external constraints, other than the implied constraints from a budget, and a domain of 2, representing the most simplified abstraction of our motivating domain. The results were as expected: the MV-MC-MGM algorithm scales roughly linearly in computation time while the MILP formulation scales at a reasonably slow exponential rate, over increasing number of agents and internal variables. In initial testing we found the Partially Centralized DCOP approach grew too quickly in our testing sets to be feasible. When the total number of internal variables exceeded 50, the naive Partially Centralized had already hit our 30min runtime limit, on the same instances that the MILP and MV-MC-MGM approaches were running two magnitudes below that limit. Therefore the Partially Centralized approach is missing from the graphed results, as it was not competitive in its naive form.
Fig. 4. MV-MC-MGM & MV-MC-DCOP MILP, with randomly generated minSatisfied

Fig. 5. MV-MC-MGM & MV-MC-DCOP MILP, with randomly generated minSatisfied

Fig. 6. MV-MC-MGM & MV-MC-DCOP MILP, with randomly generated minSatisfied
Fig. 7. MV-MC-MGM & MV-MC-DCOP MILP, with randomly generated minSatisfied

Fig. 8. MV-MC-MGM & MV-MC-DCOP MILP, with minSatisfied = 1

Fig. 9. MV-MC-MGM & MV-MC-DCOP MILP, with minSatisfied = 1
Figures 4 through 7 show the runtime and global utility across 4 different sets of MV-MC-DCOP problem generator parameterizations, each setting averaged over 10 generated problem instances. In Figures 4 and 5, we see that MV-MC-MGM scales linearly and the MILP grows exponentially, while MV-MGM earns no less than 75% of optimal utility, and as much as 95%. We see in Figures 6 and 7 that increasing constraint density or budget did not cause more than one magnitude of change in runtime. In Figure 7 we see that increasing constraint density has a similar effect on both algorithms on our sample instances in terms of runtime.

While Figures 4 through 7 have random minSatisfied values, Figures 8 and 9 use minSatisfied = 1 for all agents. This simplifies the problem greatly as can be seen by MV-MC-MGM reaching closer to optimal solutions, and lower run time for the MILP compared with similar instances used for Figures 4 and 5. While MV-MC-MGM has increased runtime when compared to Figures 4 and 5, it still scales linearly and dominates the MILP approach in all but the smallest cases.

Please note that MV-MC-MGM is run in a single-threaded simulation, so the decrease in computation time is likely even more dramatic in a true distributed environment due to inherent parallelization. Even disregarding this, the runtime dominates the MILP formulation in nearly all cases. As plain DCOP and MC-DCOP can easily scale to thousands of agents [4], it is not surprising that there is little variance in MV-MC-MGM runtime across the tests in Figures 4 through 9 which were chosen to be small due to the runtime of the complete approaches.

9 Future Work

Although as expected the naive Partially Centralized approach did not scale well on our test sets, the threshold method (which we did not have adequate time to implement and test) may prove promising. Additionally, the pruning on Partially Centralized methods should perform better on graphs where the agents have dense internal constraints, whereas our test sets had none. The MC-MV-MGM algorithm could also be improved, such as by adding breakout mechanisms. It is less clear how to improve upon the MILP, and it is less desirable as it produces significant privacy loss as well as longer runtime.

We present here our early work on algorithms ultimately intended as a pre-processing step. In addition to determining the most appropriate algorithm, the most appropriate methods of simplifying the world model should be explored. In our motivating domain, the formulations usually order the time constraints, have time durations which are probabilistic, and requesting agents split their total reward among the providers who have provided them and therefore do not assign a priori per-request rewards. Analysis should be performed on the appropriate balance between simplification for a low runtime and accuracy to produce more viable suggestions for relevant real world domains. This balance will also depend on whether the more exact solver calls the pre-processing once or interleaves it in execution.

Along similar lines to the determinization of time to completion, the service provider hierarchy has been reduced to implementing disjoint providers and requesters in the MV-MC-DCOP framework. While nothing in MV-MC-DCOP requires that an agent
cannot be both a provider and requester, the obvious method of applying the minSatisfied field encourages this separation, and the presented algorithms rely on it.

While we present here a slower complete centralized solution and a faster incomplete distributed solution, extending existing complete DCOP algorithms should be investigated. In particular, [5] provides a complete MC-DCOP algorithm based on Adopt [6], thus providing a logical starting point.

10 Conclusion

DCOP is a major paradigm within cooperative multiagent systems for coordination, scheduling, and task allocation in domains such as networks of sensors, unmanned air vehicles, or software personal assistants. Yet for some domains it is too simplistic to produce an optimal solution in terms of a more complete model. We would suggest that in such settings the simplicity and computational efficiency could be an asset in seeding the space for a slower, more accurate model. In this paper we present a new generalization of DCOP, MV-MC-DCOP, and present preliminary results comparing complete and incomplete implementations within the motivating domain of service coordination.

Although complete algorithms exist for DCOP, they tend to run at least one order of magnitude, and often more, slower than incomplete algorithms. Since our motivation is to ultimately provide a preprocessing step, which may be called iteratively, we believe the speed optimality trade-off of incomplete algorithms is a better fit for our domain. Furthermore, we are already relying on a relaxed model which will not always result in a feasible solution in the full scale model, which provides yet another reason to prefer speed over optimality.

References

3. David A. Burke and Kenneth N. Brown: A Comparison of Approaches to Handling Complex Local Problems in DCOP. Distributed Constraint Satisfaction workshop, Riva del Garda, Italy (2006) 27–33