BALANCING LOCAL RESOURCES AND GLOBAL GOALS IN MULTIPLY-CONSTRAINED DISTRIBUTED CONSTRAINT OPTIMIZATION

by

Emma Bowring

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2007

Acknowledgments

I would like to thank my adviser, Milind Tambe, and my committee members: Barbara Grosz, Anne Balsamo, Yolanda Gil and Stacy Marsella for their support and advice. I would also like to thank my co-authors: Rachel Greenstadt, Hyuckchul Jung, Gal Kaminka, Rajiv Maheswaran, Janusz Marecki, Ankit Modi, Jay Modi, Ranjit Nair, Praveen Paruchuri, Jonathan Pearce, David Pynadath, Paul Scerri, Nathan Schurr, Pradeep Varakantham and Makoto Yokoo. Finally, I would like to thank my friends and family for their invaluable support.

Contents

List Of Figures

Abstract

Distributed constraint optimization (DCOP) is a useful framework for cooperative multiagent coordination. DCOP focuses on optimizing a single team objective. However, in many domains, agents must satisfy constraints on resources consumed locally while optimizing the team goal. These resource constraints may need to be kept private or shared to improve efficiency. Extending DCOP to these domains raises two issues: algorithm design and sensitivity analysis. Algorithm design requires creating algorithms that trade off completeness, scalability, privacy and efficiency. Sensitivity analysis examines whether slightly increasing the available resources could yield a significantly better outcome.

This thesis defines the *multiply-constrained DCOP (MC-DCOP)* framework and provides complete and incomplete algorithms for solving MC-DCOP problems. Complete algorithms find the best allocation of scarce resources, while incomplete algorithms are more scalable. The algorithms use *mutually-intervening search*; they use local resource constraints to intervene in the search for the globally optimal solution. The algorithms use four key techniques: (i) transforming constraints to maintain privacy; (ii) dynamically setting upper bounds on resource consumption; (iii) identifying the extent to which the local graph structure allows agents to compute exact bounds on resource consumption; and (iv) using a virtual assignment to flag problems rendered unsatisfiable by their resource constraints. Proofs of correctness are presented for all algorithms. Finally, the complete and incomplete algorithms are used in conjunction with one another to perform distributed local reoptimization to address sensitivity analysis.

Experimental results demonstrated that MC-DCOP problems are most challenging when resources are scarce but sufficient. In problems where there are insufficient resources, the team goal is largely irrelevant. In problems with ample resources, the local resource constraints require little consideration. The incomplete algorithms were two orders of magnitude more efficient than the complete algorithm for the most challenging MC-DCOP problems and their runtime increased very little as the number of agents in the network increased. Finally, sensitivity analysis results indicated that local reoptimization is an effective way to identify resource constraints that are creating bottlenecks.

Taken together these new algorithms and examination of the problem of sensitivity analysis help extend the applicability of DCOP to more complex domains.

Chapter 1

Introduction

This thesis focuses on cooperative multiagent systems. Cooperative multiagent systems are networks of intelligent agents used to perform distributed computation. Intelligent agents may be broadly defined as automated software or hardware entities that are goal-oriented and situated in a complex, dynamic environment. The networks of cooperative agents are heterogeneous, and not all agents have direct communication links to one another. Additionally, information is distributed throughout the network either due to privacy concerns or to the impractically of centralizing. Yet, the agents need to coordinate their activities to accomplish their collective goals [29, 57, 35, 11, 38, 60, 37].

Various models have been proposed to handle cooperative multiagent coordination. These models differ in the complexity of the computation performed by individual agents and the complexity of the interactions among agents. Distributed Constraint Optimization (DCOP) [36, 33, 1, 48] is a cooperative multiagent coordination framework that has been applied to distributed meeting scheduling, distributed factory and staff scheduling, and sensor network domains [29, 35, 39, 23].

1.1 Problem Addressed

DCOP algorithms have been widely applied, but, DCOP, like many cooperative multiagent frameworks, assumes that problems can be expressed in terms of a single global goal to be optimized. However, many multiagent coordination problems cannot be fully expressed as a single team goal. Examples range from distributed meeting scheduling [35, 37] to distributed software development [10, 18, 23, 26], distributed task or role allocation [24, 57, 41, 39], and sensor networks [29].

One prototypical bounded optimization domain is distributed meeting scheduling. In distributed meeting scheduling problems, groups of researchers working in labs at several different locations need to set up meetings to facilitate work on a joint project. Each researcher has preferences over when and where the meetings he or she is involved in will occur. The global goal is to schedule the meetings in such a way as to maximize the sum of everyone's satisfaction with their schedules. This global goal can be effectively modeled using a cooperative multiagent framework like DCOP. However, given that the research groups are in different locations, there are costs associated with traveling to meetings and the money for these expenses is allocated from travel budgets maintained by group leaders in each research group. Now the problem becomes that of maximizing the sum of everyone's preferences, while ensuring that the travel costs accrued by researchers in each group do not exceed the travel budget for that group.

This combination of a global optimization goal and local resource constraints that must be satisfied is what characterizes *bounded optimization* domains. The resource constraints may be a feature of the problem (e.g. limited travel budget) over which the agent has no control, or they may be within the control of the agent (e.g. time devoted to the team goal). An agent may have many reasons for imposing a resource constraint on itself: individual interests, commitments to another team, or commitments to future goals of the team. Bounded optimization encompasses both internally and externally imposed local resource constraints. Additionally, these resource constraints may need to be kept private. Group leaders may be unwilling to share their budgetary information. Since these resource constraints cannot be expressed as part of the team goal and because specific constraints may need to be kept private during execution, the existing DCOP framework is unable to address this important class of problems. Finally, resource constraints may not be as absolute as the DCOP formalism assumes. In the distributed meeting scheduling domain, researchers may wish to know how alterations to the funds available for travel will impact the team's performance before committing to the alterations.

In extending DCOP to handle bounded optimization problems, this thesis tackles two specific problems: designing algorithms to solve bounded optimization problems and performing sensitivity analysis.

- *Algorithm Design:* Bounded optimization domains like distributed meeting scheduling require multiply-constrained DCOP algorithms that can optimize a global objective, while ensuring that resource limits are not exceeded. These algorithms must be able to keep these resource constraints private when required (e.g. travel budgets in distributed meeting scheduling [35]). Some bounded optimization problems require *complete (globally-optimal)* algorithms that can guarantee optimal usage of scarce resources. For example, if group leaders had relatively tight travel budgets, it might be worth expending the extra computation time to find the best way to allocate every last travel dollar. Other problems require extremely efficient *incomplete (locally-optimal)* algorithms that can handle large scale problems. For example, if the distributed meeting scheduling problem involved a large number of research groups and many meetings, finding the best schedule may be too time consuming. Both types of algorithms are needed.

- *Sensitivity Analysis:* Performing sensitivity analysis is especially useful in bounded optimization domains because these domains feature resource constraints that are treated as hard constraints. In real world engineering domains, constraints may not be as strict as the DCOP formalism assumes. In some domains, additional resources could be acquired if the benefits to the team sufficiently outweighed the costs [4, 3, 27]. For example, in the distributed meeting scheduling problem, group leaders might be willing to reallocate a small amount of money to supplement their travel budget, if it would allow the overall schedule to be significantly improved. However, before taking money away from another budget item, group leaders would want to know what effect their reallocation would have and whether they are the best agent to make that reallocation. It is, therefore, very useful to be able to perform sensitivity analysis.

In summary, the challenge is to balance local resource constraints and global objectives in a privacy aware manner, while using non-private information to maximize efficiency. In addition, techniques for analyzing problem solutions to see how to improve the balance between resource constraints and global goals are needed.

1.2 Challenges and Contributions

1.2.1 Algorithm Design and Implementation

This thesis defines a new problem formulation called mutiply-constrained DCOP (MC-DCOP), which captures problems involving global goals and local resource constraints. The first part of this thesis focuses on algorithm design, specifically, designing both complete and incomplete algorithms to solve MC-DCOP problems. Four main challenges had to be addressed in designing the MC-DCOP algorithms:

- *Search Complexity:* Agents' additional resource constraints add to the search complexity of bounded optimization problems. Since these additional constraints have been added to a problem without increasing the number of agents in the system, the primary challenge in algorithm design is to handle this added complexity without greatly increasing the runtime of the algorithm. This requires quickly pruning unproductive search paths.

- *Harnessing Existing Algorithms:* A design choice was made to harness existing DCOP algorithms, because significant progress has been made in maximizing their efficiency [36, 60, 63, 1, 48, 30]. However, existing algorithms are not designed to handle resource constraints that are possibly private and not part of the global goal. The challenge was to adapt these algorithms to the new bounded optimization problems while preserving their efficiency mechanisms.

- *Privacy/Efficiency:* The third challenge was how to exploit constraint revelation to gain efficiency when privacy was not required. The design choice was made to give fine-grained control over the privacy/efficiency tradeoff by allowing individual resource constraints to be kept private. This meant that the inclusion of a single private resource constraint did not necessitate using a completely private algorithm, however, it created the challenge of handling private and non-private resource constraints simultaneously within the same problem.

- *Unsatisfiability Detection:* The final challenge, that of detecting when the local resource constraints have rendered the problem unsatisfiable, is most relevant to incomplete algorithms. Complete algorithms systematically consider all possible assignments and thus can straightforwardly detect unsatisfiability. However, locally optimal algorithms, explore only part of the search space and so need to know when to stop searching if the problem is unsatisfiable.

One of the primary contributions of this thesis is that it presents both complete and incomplete multiply-constrained DCOP (MC-DCOP) algorithms to solve bounded optimization problems. These algorithms employ *mutually-intervening searches* to address the challenge of search complexity: while an agent immediately intervenes in the search for the team optimal if its local resource constraint is violated, opportunistic search for the global optimal solution obviates testing all partial solutions for resource constraint satisfaction.

The complete Multiply-Constrained Adopt (MCA) algorithm employs three techniques, which all contribute to the mutually-intervening search: constraint-graph transformation, dynamically-constraining search and local acyclicity. The first technique, *constraint-graph transformation*, is motivated by the challenge of harnessing existing algorithms. The key innovation is in efficiently employing virtual variables to enforce an agent's resource constraints within the singly-constrained DCOP algorithms. Each virtual variable represents a single resource constraint. These virtual variables asynchronously notify affected agents when an over-expenditure of resources occurs and preemptively prune search paths. For correctness and privacy preservation, the algorithm restructures the multiagent network and appropriately places the virtual variables in that network. The other two techniques address the privacy/efficiency challenge by communicating information about the available resources to agents affected by the resource constraints. In *dynamically-constraining search*, an agent reveals to its neighbors an upper-bound on the resources available for use by that neighboring agent. When optimizing the global objective function, the agent's neighbors only consider solutions that abide by the upper-bounds. Ignoring potential solutions that are overly resource expensive improves algorithmic efficiency. The final technique, *local acyclicity*, further improves efficiency in locally acyclic networks by allowing the communicated resource bounds to be tightened to exact bounds, which further prunes the search space. It is also shown that these exact bounds cannot be applied in areas of the multiagent network that are not locally acyclic.

The incomplete Multiply-Constrained Maximum Gain Message algorithms (MC-MGM-1 and MC-MGM-2) presented in this thesis also make use of *constraint-graph transformation* and *dynamically-constraining search* to meet the three challenges common between complete and incomplete algorithms. The main innovation was addressing the unsatisfiability detection challenge. For this, two design choices were available: search for the optimal satisfying assignment and detect during the search whether the problem is unsatisfiable or maintain an invariant that resource limitations are never exceeded and detect at the outset whether the problem is unsatisfiable. The latter approach fits better with the idea of mutually-intervening searches because it uses the resource limitations to prune the search space. However, in order to maintain the satisfaction invariant, an initial assignment needs to be found that meets this invariant. To meet this challenge a virtual starting assignment was added to the problem which met the satisfaction invariant. The starting assignment was designed so that the failure of any agent to move away from it before termination flagged the problem as unsatisfiable.

The complete bounded optimization DCOP algorithm presented in this thesis was built on top of Adopt, one of the most efficient complete DCOP algorithms [36]. However, the techniques outlined above could also be applied to other complete algorithms, e.g. OptAPO and SynchBB [33, 60]. The incomplete algorithms were built on top of the k-optimal Maximum Gain Message (MGM) algorithms. K-optimal algorithms are locally optimal algorithms where k indicates the locality of the optimal the algorithm reaches [45]. The techniques described in this thesis could also be employed in conjunction with other incomplete algorithms like the Distributed Stochastic Algorithm (DSA), the Distributed Breakout Algorithm (DBA) and the Low Communication Approximate DCOP Algorithm (LA-DCOP) [45, 14, 61, 44, 41]. The algorithms allow for fine-grained control over the privacy of constraints and the exploitation of the information inherent in individual resource constraints. Proofs of correctness are provided for all of the algorithms.

Experimental results for the complete and incomplete algorithms demonstrated that bounded optimization problems are most challenging when resource constraints are tight but satisfiable. In problems where there are insufficient resources, the team goal is largely irrelevant, making the problem a distributed constraint satisfaction problem. In problems with ample resources, the local resource constraints require little consideration, making the problem a normal DCOP. The incomplete algorithms were two orders of magnitude more efficient than the complete algorithm for the most challenging bounded optimization problems. The incomplete algorithms were also extremely scalable, with very little increase in the runtime as the number of agents in the network increased. The results also confirmed that the smaller the subgroup of agents that coordinated in the locally-optimal algorithms, the greater the efficiency of the algorithm, but the lower the quality of the solution obtained.

1.2.2 Sensitivity Analysis

While designing multiply-constrained DCOP algorithms is important for extending DCOP to more realistic and complex domains, real world domains also raise the question of sensitivity analysis. A simple approach to sensitivity analysis would be to run the complete multiply-constrained DCOP algorithm on all of the possible problem variants that result from introducing additional resources and compare the solutions. However, even singly-constrained DCOP is known to be an NP-hard problem, so running the complete algorithm an additional $O(2^n)$ times, where n is the number of agents in the network, would be very computationally expensive. Furthermore, the introduction of additional resource at a set of agents causes only a local perturbation to the problem. Thus, rerunning from scratch on each problem variant would require performing many duplicate computations. Therefore, the main challenge is to identify bottleneck resource constraints efficiently while taking advantage of calculations that have already been performed. The other main challenge is to do this identification in a distributed manner.

This thesis presents two different approaches to sensitivity analysis. The first approach, *link analysis*, takes the solution to the original problem and rapidly examines individual resource constraints in isolation to see whether a particular agent can use additional resources to improve its local contribution to the team effort. This approach is very efficient, but it ignores the rippling effect that the local perturbation will have on other team members. A more sophisticated approach, *local reoptimization*, harnesses the strengths of both the complete and incomplete algorithms. The complete algorithm provides the globally optimal solution to the problem as originally posed. Once seeded with the original global optimal, the incomplete algorithm uses its local optimization techniques to propagate the effects of the additional resources to other team members.

These two approaches to sensitivity analysis were compared experimentally and the results indicated that considering individual resource constraints in isolation is ineffective at solving sensitivity analysis. However, local reoptimization was found to be be very effective. A heuristic was also identified to further target the search for bottleneck nodes.

1.3 Guide to Thesis

This thesis is organized in the following way. Chapter 2 introduces the general DCOP framework and the new Multiply-Constrained DCOP problem that this thesis addresses. Chapter 3 focuses on complete algorithms, providing background, algorithm descriptions and experimental results. Chapter 4 examines incomplete algorithms, again providing background information, new bounded optimization algorithms and experimental results. Finally, Chapter 5 looks at the combined usage of the algorithms on sensitivity analysis problems. Chapter 6 presents related work and Chapter 7 concludes the thesis.

Chapter 2

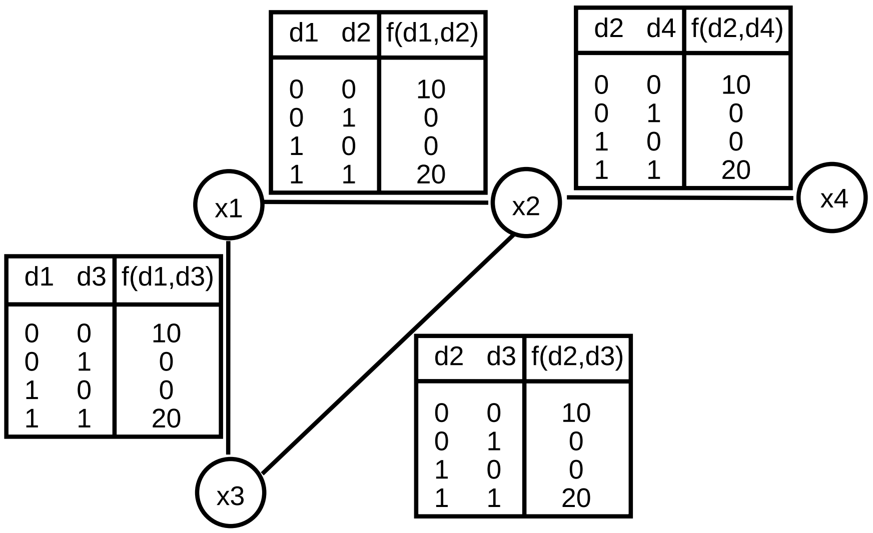
Problem Definition

This chapter begins by providing background on the DCOP formalism (Section 2.1), which is a framework for approaching multiagent coordination. DCOP is used in domains such as distributed task allocation, distributed meeting scheduling, and sensor networks. It allows for distributed control over actions and information, which makes it useful in domains where privacy is important or where computational or communication limitations prevent centralization. In Section 2.2, two domains are presented that motivated this thesis' attempt to increase the expressivity of the DCOP formalism. Finally, Section 2.3 presents the new multiply-constrained DCOP (MC-DCOP) formalism which provides the basis for this thesis work.

2.1 DCOP Definition

The field of cooperative multiagent systems studies how teams of intelligent agents can be harnessed to perform distributed coordinated computation. DCOP is a widely-used formalism for performing multiagent coordination. DCOP [36, 1, 48] grew out of the field of Constraint Optimization (COP) [53, 61], which was itself built upon work from the field of constraint satisfaction (CSP) [3, 63, 60].

A COP, like a CSP, is basically made up of a set of n variables or nodes, $\{x_1, x_2, \ldots, x_n\}$, and a set of constraints between the variables. Constraint reasoning problems are frequently represented diagrammatically, with the variables represented by ellipses and the constraints represented as links between subgroups (frequently pairs) of variables. An example of a graph 2-coloring problem is shown in Figure 2.1. Graph coloring problems are a standard benchmark domain in the field of constraint reasoning. In the example, there are four variables $\{x_1, x_2, x_3, x_4\}$ and four constraints, one between x_1 and x_2, another between x_1 and x_3, a third between x_2 and x_3, and the last between x_2 and x_4.



d1	d2	f(d1,d2)
0	0	10
0	1	0
1	0	0
1	1	20

d2	d4	f(d2,d4)
0	0	10
0	1	0
1	0	0
1	1	20

d1	d3	f(d1,d3)
0	0	10
0	1	0
1	0	0
1	1	20

d2	d3	f(d2,d3)
0	0	10
0	1	0
1	0	0
1	1	20

Figure 2.1: An example 2-coloring DCOP problem.

Variable x_i can take on any value from the discrete finite domain D_i. The notation $x_i \leftarrow d_i$ means that variable x_i has taken on the value d_i. The value generally represents an action the user or agent should perform as part of the larger team effort. For example, in a sensor network domain an agent would control a variable representing a particular sensor and the values the variable took on would represent when and where the sensor should sense. In the graph 2-coloring example in Figure 2.1, each of the four variables has two values in its domain {0,1}, which represent the two colors available for use.

In extending COP to DCOP, variables are assigned to agents, where agents are physically distinct entities. Each variable is owned by a single agent, which controls the value that the variable takes on. Agents may own one or more variables. There is no standard diagrammatic way of indicating which agents own which variables in a DCOP. The designation is usually omitted; its inclusion is only necessary when an algorithm makes a logical (rather than just implementational) distinction between communication sent between variables within an agent and communication between variables across agent boundaries. Most DCOP algorithms do not make a logical distinction between the two types of communication.

Constraints convey how the values taken on by different variables affect one another. Variables x_i and x_j are considered neighbors if they share a constraint, and only neighbors can directly communicate with one another. In CSP problems, constraints simply specify that certain combinations of values are legal or illegal. However, in the more expressive COP and DCOP formalisms, constraints specify how preferable different combinations of values are by having cost functions (f_{ij}) associated with the constraints. Diagrammatically the cost functions are shown as tables and the proximity of a table to a particular link, as well as the column titles in the table, convey which link the cost function is associated with. In the example in Figure 2.1, there are identical cost functions on each link. The cost functions do not have to be identical on all links, but, for simplicity, they are in this example. In the example, if x_1 took on the value 0 and x_2 took on the value 1, the cost, denoted f, on that particular link would be 0. In contrast, if both x_1 and x_2 took on the value 0, the f-cost on that link would be 10. This means that the assignment $\{x_1 \leftarrow 0, x_2 \leftarrow 1\}$ is preferable, due to its lower cost, to the assignment $\{x_1 \leftarrow 0, x_2 \leftarrow 0\}$. In terms of the 2-coloring problem, the coloring constraint is reflected in the cost function, which intuitively suggests that connected areas must select different colors, otherwise they will incur a high cost; however, if they must use the same color, color 0 is preferred to color 1.

The objective of a DCOP is to choose values for the variables such that the global sum over the set of constraint cost functions (f_{ij}) is minimized, i.e. find an assignment, A, s.t. F(A) is minimized:

$$F(A) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j) \quad where \;\; x_i \leftarrow d_i, x_j \leftarrow d_j \in A$$

In the example in Figure 2.1, $F(x_1 \leftarrow 0,\ x_2 \leftarrow 0,\ x_3 \leftarrow 0,\ x_4 \leftarrow 0) = 40$ because an f-cost of 10 is accrued on each link. The assignment $(x_1 \leftarrow 1,\ x_2 \leftarrow 0,\ x_3 \leftarrow 1,\ x_4 \leftarrow 0)$ has an f-cost of 30 because the link between x_1 and x_3 has a cost of 20 and the link between x_2 and x_4 has a cost of 10. The optimal assignment F(A*) would be $(x_1 \leftarrow 0,\ x_2 \leftarrow 0,\ x_3 \leftarrow 1,\ x_4 \leftarrow 1)$ which leads to an f-cost of 10 which is accrued from the link between x_1 and x_2.

So far, DCOP has been defined as a problem that involves minimizing a set of cost functions. However, DCOP can equivalently be defined as a problem of maximizing a set of utility functions. Due to the legacy of the existing DCOP algorithms used in this thesis (Adopt [36] and MGM [46]), different definitions of DCOP will be used in different parts of the thesis. The complete algorithms described in chapter 3 work as cost minimizers, whereas the incomplete algorithms described in chapter 4 are utility maximizers. Converting a problem from one of costs to one of rewards can be done by changing each constraint function from $f(d_i, d_j) = f_{ij}$ to $f(d_i, d_j) = B - f_{ij}$, where B is a large constant. This mapping becomes relevant in Chapter 5, where the work on sensitivity analysis involves bringing both complete and incomplete algorithms to bear on a problem.

2.2 Motivating Domains

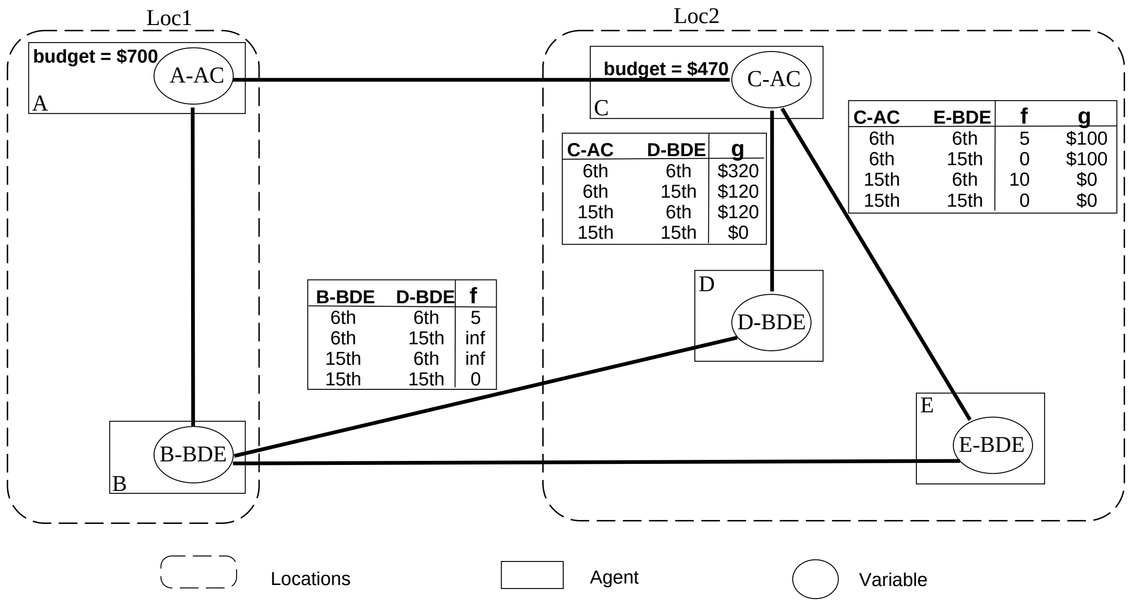
This section demonstrates the need to extend the expressivity of DCOP with examples from two domains: distributed meeting scheduling and distributed software development.

Distributed Meeting Scheduling: In distributed meeting scheduling problems, groups of researchers working in labs at several different locations need to meet to facilitate work on a joint project. Each researcher has preferences over when and where the meetings he or she is involved in will occur. However, given that the research groups are in different locations, there are costs associated with traveling to meetings. The money for these expenses is allocated from travel budgets maintained by group leaders in each research group. The problem being presented to the multiagent system is to maximize the global sum of everyone's satisfaction with the schedule, while ensuring that the travel costs accrued by researchers do not exceed the local travel budget for their research group.

In addition, while researchers are willing to share their scheduling preferences with each other, the budgetary information needs to be kept private.

The existing DCOP formalism is able to model meeting time preferences using f-cost functions, however, it cannot handle multiple types of constraints being defined on the same set of variables. Therefore, it cannot capture both the global and local aspects of the problem simultaneously. In addition, the travel budgets are private n-ary constraints, and DCOP cannot explicitly represent individual constraint privacy.

A simple distributed meeting scheduling example is shown in Figure 2.2. Researchers, A, B, C, D and E are divided between labs at Loc1 (A,B) and Loc2 (C,D,E). Each researcher has an agent (shown as a box) that acts on his/her behalf. Researchers A and C need to meet, and researchers B, D and E also need to meet. Each researcher's agent has a variable which represents the meeting they are to participate in (shown as an ellipse inside the box of the agent that owns the variable). The domain of each variable is a tuple of time-of-meeting and location. In the example in Figure 2.2, the variables are all assumed to have the same set of domain values and those domain values are the 6th of October at Loc1 (abbreviated in the figure as "6th") and the 15th of October at Loc2 (abbreviated as "15th"). Although not shown in the example, giving each agent a separate copy of the meeting variable allows an individual researcher to opt out of a meeting if his or her attendance is not mandatory. For example, the BDE meeting is represented with three variables B-BDE, D-BDE and E-BDE. If E's attendance were not mandatory, E-BDE (E's copy of the BDE meeting variable) could be set to a value indicating non-attendance, while B-BDE and D-BDE took on a value indicating the time and location the meeting was to be held.



C-AC	D-BDE	g
6th	6th	$320
6th	15th	$120
15th	6th	$120
15th	15th	$0

C-AC	E-BDE	f	g
6th	6th	5	$100
6th	15th	0	$100
15th	6th	10	$0
15th	15th	0	$0

B-BDE	D-BDE	f
6th	6th	5
6th	15th	inf
15th	6th	inf
15th	15th	0

Figure 2.2: Meeting Scheduling Example

The links between agents capture the scheduling preferences and the costs of travel. For readability not all of the link functions are shown in Figure 2.2. The f-cost from the DCOP framework can express agents' scheduling preferences. An example is shown on the link between agents B and D. If both agents pick the 6th October then the f-cost (representing their preference) is 5. The infinite cost for selecting different values represents the fact that choosing different times for the same meeting is an invalid assignment. The f-cost function on the link between agents C and E reflects a preference that the BDE meeting precede the AC meeting. An additional cost (marked g) reflects the cost of traveling to a meeting. The location of a particular meeting is indicated by the location portion of the domain value. If the meeting is being held at the researcher's current location, then no travel costs are incurred. For example, on the link between C-AC and D-BDE, when both meetings are scheduled for 15th of October at Loc2, the travel cost, marked g, is \$0. If travel is required, the researcher's group leader incurs the travel cost. The cost varies depending on the date of travel and whether all members are traveling on the same day and can share a cab to the airport. For example, on the link between C-AC and D-BDE, when at least one meeting variable has selected the value of 6th October at Loc 1, the travel cost is greater than \$0. In the example in Figure 2.2, A and C are group leaders and their travel budgets are \$700 for A and \$470 for C. The presence of a travel budget indicates they are the group leaders.

Distributed Software Development: Many software companies have campuses in different parts of the globe, and teams there must collaborate across both distances and time zones [10, 26]. Interdependent tasks within a project must be scheduled for their timely completion. In addition, to facilitate the handoff, a team liaison must videoconference with the team to which the code is being sent during the initial stages of development on a dependent piece of software. This liaison may have to stay after hours to videoconference in the new team's time zone. To avoid burnout, corporate policy may limit liaison overtime to a specific number of hours a week per liaison.

The limits on liaison overtime introduce an extra set of n-ary satisfaction constraints, but unlike the travel budgets from the meeting scheduling domain, the overtime limits are publicly known constraints. Once again, DCOP can capture the scheduling preference constraints or it could capture the satisfaction constraints (since they are non-private), however, it cannot express both sets of constraints simultaneously.

The variables a distributed software development domain would represent tasks to be completed and the values would represent times at which a task could be scheduled. The regular DCOP f-cost function could capture the temporal precedence constraints, e.g.

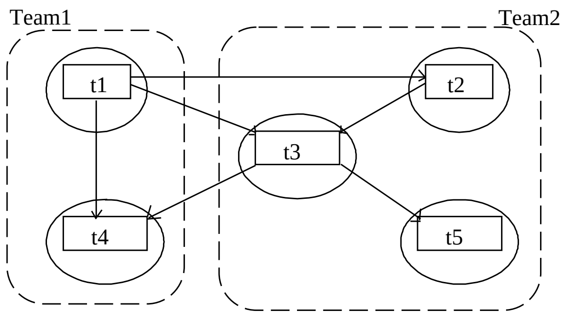


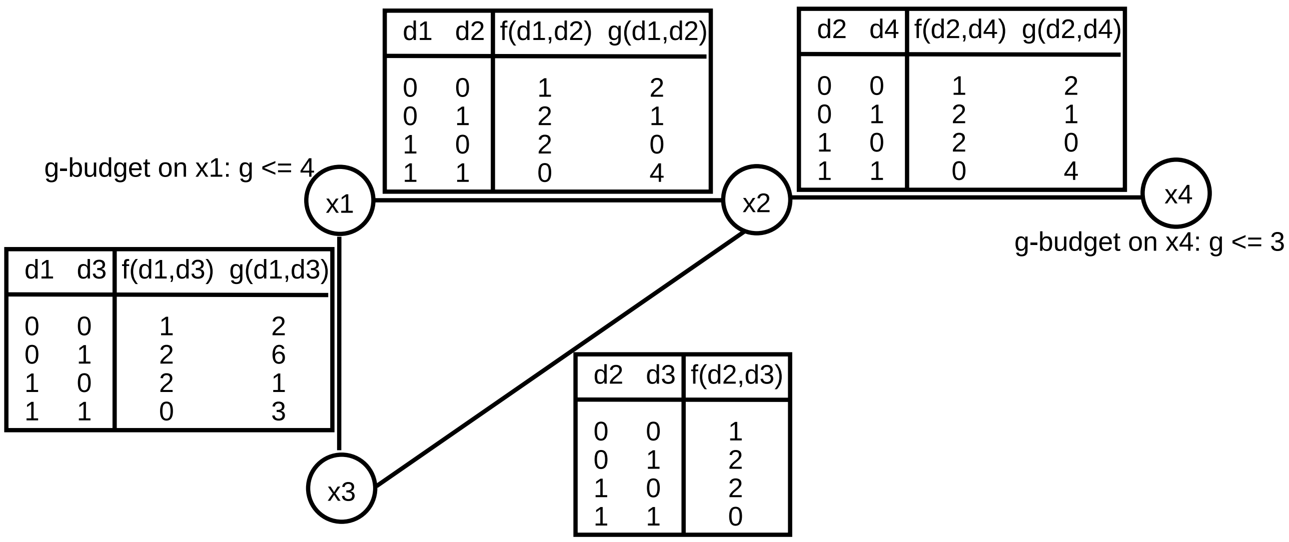
Figure 2.3: Distributed Software Development Example

task 1 must complete before task 2. The f-cost constraints could also capture preferences over the potential times, e.g. it is better to start task 1 early in the day. However, if task 2 builds on task 1, then the liaison from the team working on task 1 would accrue overtime hours if task 2 is scheduled outside of the normal work hours of the team working on task 1. The sum of the hours of overtime worked by each liaison needs to remain below the company threshold.

The two examples above illustrate the need for multiply-constrained DCOPs to model such collaborations. These domains require agents to optimize an f-cost function and yet adhere to additional resource constraints, which may or may not be publicly known. The next section presents multiply-constrained DCOP, a new formalism with the expressivity to capture these domain problems.

2.3 Multiply-Constrained DCOP

To address the expressivity challenges of real world applications such as those described in the previous section, this thesis defines *Multiply-Constrained DCOP (MC-DCOP)*. MC-DCOP adds a new cost function g_{ij} on a subset of the links connected to node x_i and a g-budget G_i which the accumulated g-cost on local links must not exceed. Together the g-function and g-budget constitute a g-constraint. Figure 2.3 shows an example with g-constraints on x_1 and x_4. In the example, if x_1, x_2, x_3 each take on the value of 1 (leading to an optimal f-cost) then x_1's local g-cost is 7, which violates x_1's g-budget of 4. G-constraints may be either private or shared.



d1	d2	f(d1,d2)	g(d1,d2)
0	0	1	2
0	1	2	1
1	0	2	0
1	1	0	4

d2	d4	f(d2,d4)	g(d2,d4)
0	0	1	2
0	1	2	1
1	0	2	0
1	1	0	4

d1	d3	f(d1,d3)	g(d1,d3)
0	0	1	2
0	1	2	6
1	0	2	1
1	1	0	3

d2	d3	f(d2,d3)
0	0	1
0	1	2
1	0	2
1	1	0

Figure 2.4: Example Multiply-Constrained DCOP problem

G-cost functions cannot be merged with f-cost functions. Thus, each variable's value must meet the constraints of both f and g, and hence this is a *multiply-constrained DCOP*. It is straightforward to extend this framework to multiple g-constraints on a variable. In general, the g-constraint can be an arbitrary function on the values of x_i and its neighbors.[1] However, given that both of the motivating domains described in Section 2.2 involve additive g-constraint functions, the assumption will be made in this thesis that the function is additive. This makes the formalism slightly less general; however, the additive

[1]While for simplicity it is assumed that the set of f-neighbors and g-neighbors are the same, this is not necessary.

nature of the g-constraints parallels the additive nature of the f-constraints in DCOP. It is worth noting that the private multiply-constrained algorithms described in chapters 3 and 4 could handle non-additive g-constraints, but for simplicity of exposition, this thesis will maintain throughout the assumption that g-constraints are additive. Given the g-cost functions and g-budgets, the Multiply-Constrained DCOP (MC-DCOP) objective is defined as follows: find A s.t. F(A) is minimized, where:

$$F(A) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j) \quad where \;\; x_i \leftarrow d_i, x_j \leftarrow d_j \in A$$

and

$$\forall x_i \in V : \sum_{x_i, x_j \in neighbors(x_i)} g_{ij}(d_i, d_j) \leq G_i$$

Multiply-Constrained DCOP has three features absent in regular DCOP:

1. n-ary (g-cost) constraints
2. private g-constraints where both G_i and g_{ij} are kept private
3. the need to exploit the interaction between f- and g-constraints for pruning the search space

No existing DCOP algorithm addresses these three issues simultaneously. Specifically, none of the algorithms handle feature (3), because existing DCOP algorithms do not allow for multiple types of constraints to be defined on the same pair of variables, so there is no interaction for them to exploit. Adopt[36] can handle (2) because it does not centralize constraints, but it cannot handle (1) because it was designed to exclusively use binary constraints. OptAPO[33] and DPOP[48] can handle (1), but not (2) because they partially centralize constraints.

Now that the motivation for extending DCOP and the new formalism of MC-DCOP has been introduced the next two chapters will describe complete (Chapter 3) and incomplete (Chapter 4) algorithms designed to solve this problem.

Chapter 3

Complete MC-DCOP algorithms

This chapter focuses on the complete algorithm developed to solve the MC-DCOP problem defined in Chapter 2. Multiply-Constrained Adopt (MCA) is a novel algorithm that builds upon the highly efficient DCOP algorithm Adopt. As mentioned in the Introduction, there are three main challenges that had to be addressed in designing Multiply-Constrained Adopt. First, g-constraints add to DCOP search complexity so MCA must rapidly prune unproductive search paths. Second, Adopt [36, 1, 30] is not designed to handle resource constraints that are local, private or defined over n-ary domains. Third, MCA must exploit constraint revelation to gain efficiency when privacy is not required.

This chapter is organized as follows: Section 3.1 provides background on the Adopt algorithm, Section 3.2 describes the key techniques employed in Multiply-Constrained Adopt, Section 3.3 describes MCA in detail, Section 3.4 provides correctness and complexity results and Section 3.5 contains the experimental results.

3.1 Background: ADOPT

The Adopt algorithm [36], which provides the basis for the complete MC-DCOP algorithm developed in this thesis, is an asynchronous complete DCOP algorithm. Several mechanisms in Adopt lend themselves particularly well to creating multiply-constrained Adopt. Adopt uses an opportunistic search mechanism which can be harnessed for finding not only the best global solution but also the second best, the third best and so on, which is valuable when the best solution does not satisfy the g-constraints imposed on a problem. In addition, Adopt does not centralize multiple variables' constraints, which is important in the managing of private constraints. Adopt is also a polynomial space algorithm, which is helpful in domains where agents may have limited processing power.

In the course of computing the solution to a DCOP, variables communicate with their neighbors, variables with whom they share a constraint. In order to organize the communication between variables, Adopt sorts variables into a Depth-First Search (DFS) priority tree in which constraints are allowed between a variable and its ancestors or descendants, but not between variables in separate sub-trees. Heuristics are used to decide how to prioritize linked variables. The example in Figure 3.1 shows (a) the constraint graph initially given to Adopt and then (b) the DFS tree formed by Adopt. Each variable (except the root) must have a single parent node, but may have any number of children and neighbors. In Figure 3.1b variable x_1 is the root of the DFS tree as well as the parent of x_2. x_2 is the parent of x_3 and x_4. x_2 is a child of x_1 and x_3 is a neighbor (but not a child) of x_1.

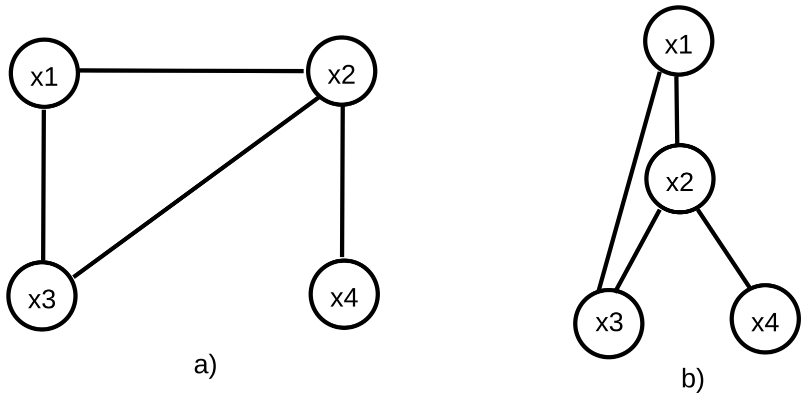


Figure 3.1: a) Original constraint graph b) DFS tree

In communicating between neighboring variables, Adopt employs two basic messages: VALUE and COST.[1] Assignments of values to variables are conveyed in VALUE messages that are sent to all neighbors lower in the DFS tree. For example, x_1 will send a VALUE messages to x_2 and x_3 indicating the value it has currently taken on. A COST message is sent from children to parents indicating the f-cost of the sub-tree rooted at the child. In the example in Figure 3.1, x_3 and x_4 will send their COST messages to x_2 and x_2 will send COST messages to x_1 that incorporate its own local cost and those costs reported by x_3 and x_4.

Adopt does not perform its computation synchronously, it is event driven and the events that trigger computation are the receipt of messages. The receipt of a VALUE or COST message prompts the sending of further VALUE and COST messages. Thus, in order to begin computation, Adopt needs to send out an initial set of messages. It does this by having variables randomly select a starting value (they cannot choose intelligently

[1]Adopt also uses THRESHOLD messages for improved efficiency. These messages are orthogonal to the concerns of this thesis and so are not discussed.

since they do not know the choices of their neighbors) and then send VALUE messages to their neighbors lower in the tree.

Upon receipt of a VALUE message, a variable, x_i, will examine all the values in its domain. For each value d_i, x_i will compute the f-cost incurred on all of its links with its ancestors given that x_i has taken on the value d_i and the ancestors have taken on the values communicated in their VALUE messages. If available, it will also add in the lower bounds on cost reported by each child given x_i's value of d_i. If lower bound cost information is not available it is assumed to be 0. The calculated sum represents a lower bound on the cost of taking on the value d_i in the current context. Variable x_i will select the value that minimizes the lower bound cost given the current context. If the lowest cost value is different to the value it had previously taken on, it will send VALUE messages to its descendants. It will also send a COST message to its parent containing the lower bound cost. Since communication is asynchronous, COST messages have a context attached to help the receiver determine information relevance. The context is a list of the variable assignments in existence at the time the COST message was sent and if it matches the variables assignments currently in existence, the cost information is still relevant.

Upon receipt of a COST message, a variable x_i first examines the message to see if the context matches the current one. If there is a match then the variable stores the lower bound reported by the child and uses the procedure described in the previous paragraph to select the lowest cost value given the information it currently possesses. After this computation it sends a COST message to its parent, and if it has changed its value assignment, it sends out VALUE messages.

A variable, x_i, maintains a lower bound on the cost of different assignments, which represents a lower bound on the f-cost of all assignment of values to the variables in the subtree rooted at x_i. A lower bound can be generated even before all children have sent information to x_i. In addition to the lower bound, variables maintain an upper bound on the cost of their subtree. The upper bound is the actual cost of the lowest cost complete assignment currently tried. Calculating an upper bound requires information for the current context from all variables in the subtree to have percolated up to node x_i. Until then it is assumed to be infinite. The root's upper and lower bounds represent the upper and lower bounds on the global problem; when they meet the optimal has been found and the algorithm terminates.

3.2 Basic Ideas

MCA uses *mutually-intervening search* to address these challenges. An agent immediately intervenes in the search for the optimal f-cost if its local g-constraint is violated. In addition, opportunistic search for the optimal f obviates testing all partial solutions for g-constraint satisfaction. The mutually intervening approach is realized through three techniques: constraint-graph transformation, dynamically constraining search, and local acyclicity.

3.2.1 Constraint-Graph Transformation

To exploit existing DCOP algorithms and maintain privacy, virtual variables are added to enforce n-ary g-constraints. Each virtual variable is responsible for the enforcement of a single g-constraint. A virtual variable collects information from all of the regular variables involved in its g-constraint and performs a centralized computation to see if the g-constraint is currently violated. If it detects a g-constraint violation, the virtual variable asynchronously preempts the current search path. Using extra variables to enforce n-ary satisfaction constraints with binary satisfaction constraints has appeared in the centralized CSP literature [55]. The use of this idea in MCA has 3 novel aspects.

- First, these virtual variables are embedded in a singly-constrained DCOP algorithm. This thesis proves the correctness of the resulting asynchronous multiply-constrained DCOP algorithm. To that end, Adopt's DFS tree is restructured and the virtual variables are appropriately placed in the tree.
- Second, since the constraint-graph restructuring can change local acyclicity properties of other variables, MCA's preprocessing identifies and preserves local acyclicity where feasible.
- Third, the local g-constraint is encapsulated in a virtual variable which is owned by the same agent as the original variable and placed as a leaf in the DFS tree. This allows the privacy of both the g-function and the g-budget to be protected because the only information required by or revealed to the other agents is that their current assignment is invalid.

3.2.2 Dynamically Constraining Search

To mitigate against the increased complexity of bounded optimization problems, it is important to exploit g-constraint revelation when allowed to gain efficiency in the search for the optimal solution. This is achieved by requiring descendant nodes to consider only assignments that will not violate their ancestors' g-constraints. Specifically, each descendant is passed a bound (termed *g-threshold*) specifying how large a g-cost it can pass up, limiting its effective domain. This g-threshold is an exact bound when the local constraint graph is acyclic and an upper bound otherwise. If a potential value assignment fails to satisfy an ancestor's g-threshold, variables will not explore this value for f-cost optimality. Additionally, the opportunistic search for an optimal f necessitates checking for g-constraint violations only for those value combinations that are of low f-cost. Thus, the searches dynamically constrain each other, leading to performance improvements.

3.2.3 Local Acyclicity (T-nodes)

The notion of local acyclicity is captured formally by the definition of T-nodes. A variable x_i is a T-node if all neighbors of x_i lower in the DFS tree are children of x_i. In Figure 2.1, x_1 is not a T-node because its lower priority neighbor x_3 is not also its child. However, x_2, x_3 and x_4 are all T-nodes. T-nodes enable the calculation of exact g-thresholds and elimination of virtual variables because their children respond independently to allocated g-thresholds.

MCA exploits the fact that g-constraints are local constraints to allow these three techniques to be applied simultaneously to different variables within the same problem. As a result privacy protecting techniques like constraint-graph transformation apply only at those variables requiring that their g-constraint be kept private. Thus, having a single private g-constraint in the problem does not preclude the other variables from using the more efficient dynamically constraining search and local acyclicity techniques.

3.3 MCA Algorithm Description

Figures 3.5 and 3.6 present pseudo-code for MCA. MCA uses three subalgorithms: (i) MCA Private (MCAP) is used when a g-constraint may not be revealed: it uses constraint-graph transformation, (ii) MCA Shared (MCAS) is used when a g-constraint is non-private, but the variable is not a T-node: it employs constraint-graph transformation and dynamically constraining search, and (iii) MCA Shared and Acyclic (MCASA) is utilized when a g-constraint is non-private and the variable is a T-node: it uses dynamically constraining search. Each sub-algorithm will be discussed separately, and then their interaction in the unified algorithm will be described.

3.3.1 MCAP

The MCAP subalgorithm is used when a variable's g-constraint must be kept private. Privacy is taken to mean that neither the g-constraints nor the variables involved are explicitly revealed to other agents. An illustrative snapshot of MCAP's execution is shown in Figure 3.2.

Since the g-constraints are not visible other agents, MCAP uses Adopt's normal opportunistic search mechanisms to search for an optimal f-cost solution. To handle the additional g-constraints, MCAP has a set of virtual variables. Each virtual variable is responsible for enforcing the g-constraint of a single variable. In the example in Figure 3.2 x_1' is a virtual variable that is responsible for enforcing the g-constraint of variable x_1. The virtual variable is controlled by the agent that owns the original variable (lines 3-5).

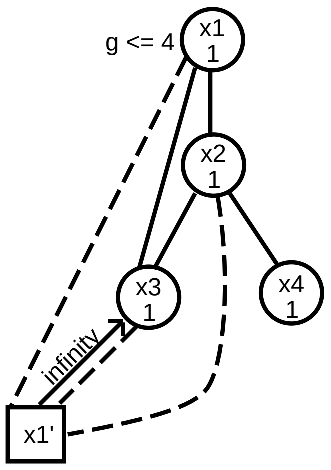


Figure 3.2: Snapshot of MCAP

A virtual variable receives VALUE messages from all of the variables in its particular g-constraint. It looks at the current full or partial assignment and tests whether its g-constraint is violated. If the current assignment violates the g-constraint, the virtual variable sends out a feedback signal, in the form of an infinite COST message (lines 60-63 in Figure 3.6). The feedback is sent to the lowest priority variable involved in the violated constraint, which in the example in Figure 3.2 would be x_3. Using the f-cost optimization mechanisms of Adopt, the feedback will propagate to the node(s) that must change their values to find the optimal satisfying solution. Specifically, if a variable (e.g. x_3) receives an infinite cost for all of its domain values, it will send a COST message to its parent (e.g. x_2) indicating an lower bound and upper bound of ∞ which will prompt the parent to change its value. As a result, the child's context is reset, allowing the child to try its domain values again.

Since feedback must be able to percolate up to all the nodes in a particular g-constraint, Adopt's DFS tree must be restructured to put all the variables in a particular g-constraint in the same subtree (lines 6-8). MCAP preprocessing then creates the virtual variables and places them as leaves, lower in the DFS tree than the variables in the g-constraint they enforce (lines 10-11). Being a leaf allows the virtual variables to receive VALUE messages from all of the variables in the g-constraint without having to send any itself. The preprocessing of the DFS tree is revisited in Section 3.3.4.

MCAP protects privacy in several ways. Since the g-constraint is encapsulated in the virtual variable, no other agent explicitly needs to know either the g-budget or the g-functions. Other variables just search for the optimal f-cost solution. Additionally, since the feedback sent by the virtual variables is a regular COST message and agents do not know the global graph structure, the fact that a variable is a virtual variable can be obfuscated. The receipt of an infinite COST message from a child can mean one of three indistinguishable things:

- the child is a virtual variable
- the child is a regular variable passing up an infinite cost from a virtual variable lower in the DFS tree
- the child is a regular variable passing up an infinite cost incurred due to the f-cost constraints

In some versions of DCOP, agents are assumed to own only one variable. If this were the case, the lowest priority neighbor in a particular g-constraint would be able to infer that its child was a virtual variable because it would receive VALUE messages and COST messages from two variables owned by the same agent. None of the other agents in a g-constraint receive COST messages from the virtual variable, so it is only the lowest priority variable that could make this inference. However, the general definition of DCOP allows an agent to own multiple regular variables, thus in general, the fact that a variable is connected to multiple variables owned by the same agent does not reveal whether the variables are real or virtual.

3.3.2 MCAS

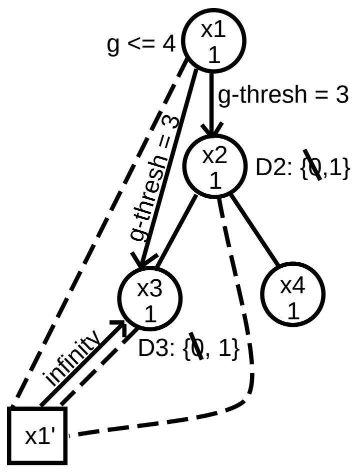


Figure 3.3: Snapshot of MCAS

MCAP's partial search of unsatisfying assignments slows the algorithm. When a g-constraint does not have to be kept private, the g-function of a link is revealed to both the variable with the g-constraint and the variable on the other end of the link. For example, in Figure 3.3, x_1 has a g-constraint which is made up of the g-budget (which is 4) and the g-cost functions on the $x_1 - x_2$ and $x_1 - x_3$ links. Since the g-constraint is not private, variable x_2 will be aware of the g-cost function on the $x_1 - x_2$ link and x_3 will be aware of the g-cost function on the $x_1 - x_3$ link. The g-cost function indicates what g-cost is incurred by each of the pairs of value choices the variables can take on.

MCAS and MCASA, shown in Figures 3.3 and 3.4 respectively, exploit g-function revelation by having nodes send their descendants g-thresholds (line 52). These g-thresholds indicate that the descendant may take on no value with a g-cost greater than the g-threshold. The snapshots from Figures 3.3 and 3.4 show the g-thresholds being passed down from x_1 to x_2 and x_3.

If the variable is not a T-node, which is the case in MCAS, the g-threshold for a child (x_l) is an upper bound: the total g-budget minus the minimum g-cost that could be consumed by each of the other links:

$$G_i - \sum_{x_j \in Neighbors(x_i) \neq x_l} min\{g_{ij}(d_i, d_j)\}$$

In Figure 3.3, the total g-budget at x_1 is 4. If the minimum g that could be consumed on the $x_1 - x_3$ link were 1, then the g-threshold sent to x_2 would be 3. Given this bound, a node can prune values from its domain that do not satisfy the g-threshold (e.g. 0 is pruned from the domain of x_2). This pruning of the domain reduces the search space and leads to speedups over MCAP.

3.3.3 MCASA

MCASA is applied when a variable's g-constraint is not required to be kept private and the variable is a T-Node. For T-nodes, it is possible to calculate an exact bound which enables more values to be pruned from the domain and further speed-ups to be achieved. For example, x_1 in Figure 3.4 is a T-node, and it calculates exact bounds, which means that it sends a tighter g-threshold (0 instead of 3) to x_3.

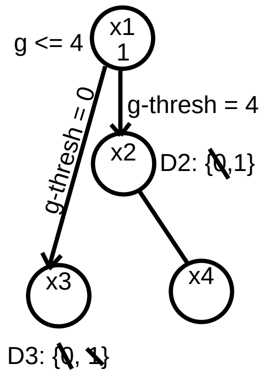


Figure 3.4: Snapshot of MCASA

Calculating the exact bounds requires a node to maintain a mapping from potential g-thresholds to lower bounds on f-costs (GFmap) for each of its children. For example, in Figure 3.4, x_1 would maintain a GFmap for x_2 and a GFmap for x_3. The GFmap for x_2 would have entries for g-budgets of $0 \ldots 4$ and the entry for 0 would be the lower bound on cost that x_2 would report if given a g-threshold of 0 in the current context. GFmap is dynamically initialized from the link g-functions (line 17-18; line 50) and then updated as COST messages arrive (line 38).

A T-node uses the g-threshold to f-cost mappings to calculate how to split its remaining g-budget among its children. The remaining g-budget is its total g-budget minus the g-cost consumed on each of the links with its ancestors. The optimal split minimizes the sum of the f-costs that will be reported by the variable's children, as estimated in the GFmap. Each time the GFmap is updated the optimal split is recalculated. The split is calculated by the calcOptimalSplit function, which is implemented using a straightforward dynamic program.

Since x_i's lower bound (lb) and upper bound (ub) for the current context now depend upon the way x_i splits its g-budget among its children, x_i's current split is now part of the context for lb and ub. Variable x_i stores the g-budget split as part of its context and when x_i's children send COST messages, they include the g-threshold that was in effect when they sent the message. The lb and ub are reset to 0 and ∞ respectively whenever x_i changes its split (line 39) as well as after normal context changes (line 28). If a node can't find a satisfactory split of its remaining g-budget, it will send up a cost of ∞ to its parent which will cause its ancestors to switch their values (line 45).

3.3.4 Combining Techniques

The subalgorithms may be applied simultaneously to different nodes within the same problem. The g-constraints are local constraints, so the only challenging situations occur when a parent and a child are using different techniques. The algorithm handles these situations as follows:

- Parent = MCAS/MCASA, Child = MCAP: The child can restrict its domain based upon the g-threshold of its parent, and the parent will automatically change its value if an infinite COST message percolates up from the child's virtual variable.

- Parent = MCAP, Child = MCAS/MCASA: The child will automatically adjust its value if the parent's virtual variable passes up an infinite cost and the child's technique makes no difference to the parent.

One key issue is whether the tree transformations performed as part of the preprocessing for MCAP and MCAS can turn a T-node somewhere else in the tree into a non-T-node

```
% An initial DFS tree is assumed to have been built in preprocessing.
% Convention: $x_i$ is self, $x_j$ is a higher priority neighbor
% and $x_k$ and $x_l$ are lower priority neighbors.
Preprocessing
(1)   for each $x_i$ from highest priority to lowest
(2)     if $Tnode_i == false$ or $private_i == true$
(3)       $x'_i$ is a new virtual variable
(4)       $Neighbors(x'_i) \leftarrow Neighbors(x_i) \cup x_i$
(5)       $Neighbors(x_i) \leftarrow Neighbors(x_i) \cup x'_i$
(6)       forall $x_k \in Children(x_i)$
(7)         if $x_k$ is not a neighbor of $x_l \in Children(x_i)$
(8)         $Neighbors(x_k) \leftarrow Neighbors(x_k) \cup x_l$
(9)       rebuildDFStree($x_1 \ldots x_n$)
(10)  forall virtual variables $x'_i$,
(11)    $parent(x'_i) \leftarrow$ lowest priority Neighbor of $x'_i$

Initialize
(12)  $CurrentContext \leftarrow \{\}$
(13)  initialize structures to store lb and ub from children
(14)  $d_i \leftarrow d$ that minimizes $LB(d)$
(15)  if $private_i == false$ and $Tnode_i == true$
(16)    forall $x_l \in Children$
(17)      for $gt \leftarrow 0 \ldots G_i$
(18)        $GFmap(x_l, gt) \leftarrow min\ f(d_i, d_l)$ s.t. $g(d_i, d_l) \leq gt$
(19)  if $Tnode_i == true$ and $private_i == false$
(20)    calcOptimalSplit
(21)  else if $private_i == false$
(22)    calcUpperBound
(23)  backTrack

when received (VALUE, $x_j$, $d_j$, $gThresh_j$)
(24)  if $private_j == false$
(25)    add ($x_j$,$d_j$,$gThresh_j$) to $CurrentContext$
(26)  else
(27)    add ($x_j$,$d_j$) to $CurrentContext$
(28)  reset lb and ub if $CurrentContext$ has changed
(29)  if $private_i == false$ and $Tnode_i == true$
(31)    if $CurrentContext$ has changed
(30)      forall $x_l \in Children$
(32)        $GFmap(x_l, gt) \leftarrow min\ f(d_l, d_i)$ s.t. $g(d_l, d_i) \leq gt$
(33)    calcOptimalSplit
(34)  backTrack;
```

Figure 3.5: Multiply-Constrained Adopt Pseudo-code, part 1

```
when received (COST, $x_k$, context, lb, ub)
(35)  update $CurrentContext$
(36)  if $context$ compatible with $CurrentContext$
        and $Tnode_i == true$ and $private_i == false$
(37)    $GFmap(x_l, gThresh_{il}) \leftarrow lb$ s.t.
          $(x_i, d_i, gThresh_{il})$ is part of $context$ from $x_k$
(38)    calcOptimalSplit
(39)    if any $gThresh_{il}$ has changed, reset lb,ub
(40)    else store lb,ub
(41)  else if $context$ compatible with $CurrentContext$ and $gThresh_{il}$
(42)    store lb and ub
(43)  backTrack

procedure backTrack
(44)  if $x_i$ not a virtual variable
(45)    if no $d$ satisfies $gThresh_j$ $LB, UB \leftarrow \infty$
(46)    else if $LB(d_i) > LB(d)$ for some $d$
(47)      $d_i \leftarrow d$ that minimizes $LB(d)$ and satisfies $gThresh_j$
(48)      if $Tnode_i == true$ and $private_i == false$
(49)        $GFmap(x_l, gt) \leftarrow min\ f(d_i, d_l)$ s.t. $g(d_i, d_l) \leq gt$
(50)        calcOptimalSplit; reset lb and ub
(51)      if $private_i == false$
(52)        SEND (VALUE, ($x_i$, $d_i$, $gThresh(x_k)$))
            to each lower priority neighbor $x_k$
(53)      else SEND (VALUE, ($x_i$, $d_i$))
            to each lower priority neighbor $x_k$
(54)    if $LB == UB$:
(55)      if TERMINATE received from parent or $x_i$ is root:
(56)        SEND TERMINATE to each child
(57)        Terminate execution;
(58)    SEND (COST, $x_i$, $CurrentContext$, $LB$, $UB$)
(59)  else % else a virtual variable
(60)    if $g(d_i, CurrentContext) > gBudget(x_i)$
(61)        SEND (COST, $x_i$, $CurrentContext$, $\infty$, $\infty$) to parent
(62)    else
(63)      SEND (COST, $x_i$, $CurrentContext$, 0, 0) to parent
```

Figure 3.6: Multiply-Constrained Adopt Pseudo-code, part 2

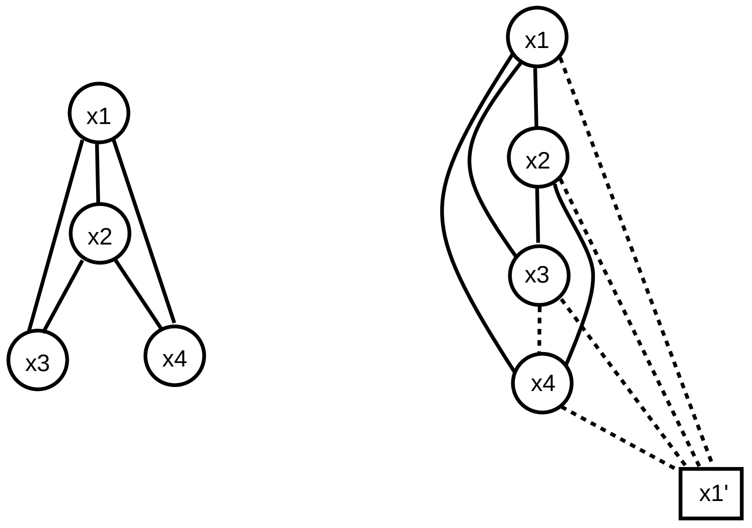


Figure 3.7: a) original constraint graph b) after adding x_1'

because it would interfere with MCA's ability to use MCASA independently of MCAS and MCAP. There are two steps in the preprocessing that cause links to be added: (i) adding empty links between x_i's children to force them into the same subtree (lines 6-8) and (ii) adding a virtual variable and empty links to each of the variables in the g-constraint it represents (lines 10-11). If a link is added between two children of a T-node, then despite having no f-function or g-function it can turn a T-node into a non-T-node. For instance, in Figure 3.7a, x_1 has a g-constraint with x_2, x_3 and x_4. During preprocessing, a link must be added between x_3 and x_4, because the tree building algorithm will attempt to place non-neighbors in separate subtrees to increase parallelism. Additionally, one variable must be chosen (arbitrarily) to be the parent of the other (x_3 is made parent of x_4). x_4's COST messages no longer flow directly to x_2 but are sent via x_3 where they are combined with x_3's local costs. This combination prevents x_2 from calculating an exact bound and renders it no longer a T-node.

In contrast, in Figure 3.2 the addition of virtual variables and their accompanying links, does not change a T-node to a non-T-node. While x_2 is originally a T-node, the addition of the virtual variable x'_1 creates a lower priority neighbor for x_2 that is not a child. However, in this case, x'_1 is a virtual variable, i.e. it has no f-functions or g-functions on any of its links and it has no domain of its own. Since x'_1 need be given no g-threshold, the fact that it is not one of x_2's children does not prevent x_2 from being a T-node.

Given these two cases, preprocessing (lines 1-11) starts by walking through the tree in priority order adding in the empty links between children where necessary and adjusting the priorities accordingly. Once a node, x_i, has been determined to be a T-node no links added between its lower priority non-neighbors will change x_i into a non-T-node, so one sweep of the tree is sufficient to correctly determine which nodes are T-nodes. After this sweep the virtual variables themselves can be added as leaves without causing any of the previously determined T-nodes to lose that property.

3.4 Correctness and Complexity of MCA

In this section the proofs for each subalgorithm are handled separately for the sake of clarity. As previously described, the interaction of the techniques in the combined algorithm does not change their properties. In the following proofs a context is the set of variable assignments upon which a piece of information is predicated.

Proposition 1 *For each node x_i for the current context, MCAP finds the assignment whose f-cost, local cost (δ) plus the sum of each of x_i's children's (x_l's) costs, is minimized while satisfying the g-constraint:*

$OPT(x_i, context) \overset{def}{=}$

$min_{d \in D_i}[\delta(d) + \sum_{x_l} OPT(x_l, context \cup (x_i, d))]$

$if\ \forall x_i \in V : \sum_{x_i, x_j \in neighbors(x_i)} g_{ij}(d_i, d_j) \leq G_i$

$\infty\ otherwise$

$where\ \delta(d) \overset{def}{=} \sum_{x_j \in ancestors} f_{ij}(d_i, d_j)$

Proof: The proof starts from the correctness of the original Adopt algorithm [36]. At every node x_i Adopt will find:

$OPT'(x_i, context) \overset{def}{=}$

$min_{d \in D_i}[\delta'(d) + \sum_{x_l} OPT'(x_l, context \cup (x_i, d))]$

where $\delta'(d) \overset{def}{=} \sum_{x_j \in ancestors} f_{ij}(d_i, d_j)$

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

Part (1) uses the fact that the virtual variables introduce an infinite f-cost into the subtree containing the violated constraint. This infinite f-cost enters lower in the priority tree than any variable in the constraint which allows the normal flow of COST messages to eventually carry it to all of the involved nodes. Since any assignment that does not violate the g-constraint will have a finite f-cost, it follows from the correctness of Adopt that by choosing the assignment that minimizes f, variables in MCAP will never take on a final assignment that violates a g-constraint unless the g-constraint is unsatisfiable.

Part (2) follows directly from the correctness of Adopt because the virtual variables report a zero cost if all constraints are satisfied. As a result Adopt's normal mechanisms ensure it will find the minimum f-cost solution. ■

Proving that MCAS is correct requires a minor addition to the MCAP proof from Proposition 1.

Proposition 2 *If the g-constraint for each node x_i is:*

$$\sum_{x_j \in Neighbors(x_i)} g_{ij}(d_i, d_j) < G_i$$

then no satisfying solution can contain on link l_{il} a g-cost greater than:

$$G_i - \sum_{x_j \in Neighbors(x_i) \neq x_l} min\{g_{ij}(d_i, d_j)\}$$

Proof: Each link consumes a certain minimum g-cost, and MCAS only subtracts the sum of the absolute minimum costs on all links. ■

For MCASA, if the g-thresholds are assigned optimally, then given the correctness of the original Adopt algorithm, MCASA is correct.

Proposition 3 *For each T-node x_i, MCASA always terminates with an optimal division of the g-budget given d_i and the current context.*

Proof by Contradiction: Assume $x_k \in Children(x_i)$ and that there are no non-T-nodes as descendants of x_i. Additionally, assume MCASA terminates with g-thresholds

g'_{ik} ($\forall x_k$) which are not optimal. Thus there exists another set of g-thresholds (g^*_{ik}) such that:

$$\delta(d_i) + \sum_{x_k} min\ lb(d_i, d_k*) < \delta(d_i) + \sum_{x_k} min\ lb(d_i, d'_k)$$

$$where\ d_k* \in \{D_k \mid g_{ik}(d_i, d_k) \leq g_{ik}*\}$$

$$where\ d'_k \in \{D_k \mid g_{ik}(d_i, d_k) \leq g'_{ik}\}$$

Since local cost $\delta(d_i)$ is constant for all g-thresholds, it can be ignored. To have been selected, g'_{ik} must have seemed optimal based on the current information at some point. To distinguish these two states of knowledge let the following terms be defined:

- $f_{actual}(g_{ik}, x_k)$ is the f-cost (specifically $min\{lb(d_i, x_k)\}$ where $d_k \in \{D_k | g_{ik}(d_i, d_k) \leq g_{ik}\}$) when all costs have percolated up from all descendants
- $f_{current}(g_{ik}, x_k)$ is the current lower bound on f-cost

When g'_{ik} is selected:

1. $\sum_{x_k} f_{actual}(g'_{ik}, x_k) > \sum_{x_k} f_{actual}(g^*_{ik}, x_k)$
2. $\sum_{x_k} f_{current}(g'_{ik}, x_k) < \sum_{x_k} f_{current}(g^*_{ik}, x_k)$
3. $f_{current}(g_{ik}, x_k) \leq f_{actual}(g_{ik}, x_k)$ for any g_{ik}
4. $\sum_{x_k} f_{current}(g_{ik}, x_k) \leq \sum_{x_k} f_{actual}(g_{ik}, x_k)$

Since there are a finite number of nodes in the tree below x_i, before termination $\sum_{x_k} f_{current}(g'_{ik}, x_k) = \sum_{x_k} f_{actual}(g'_{ik}, x_k)$ will become true. At this point:

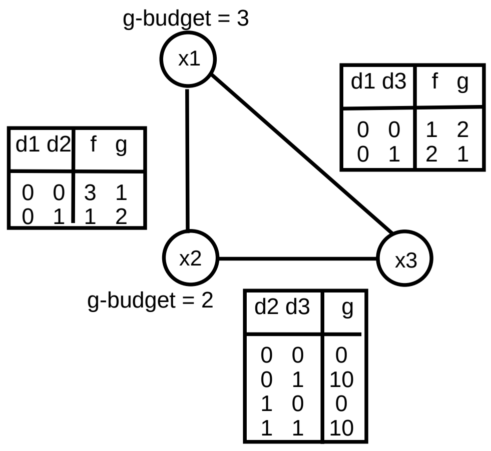
$$\sum_{x_k} f_{current}(g'_{ik}, x_k) \quad = \quad \sum_{x_k} f_{actual}(g'_{ik}, x_k)$$

$$\sum_{x_k} f_{current}(g'_{ik}, x_k) \quad > \quad \sum_{x_k} f_{actual}(g^*_{ik}, x_k) \qquad \text{by (1)}$$

$$\sum_{x_k} f_{current}(g'_{ik}, x_k) \quad > \quad \sum_{x_k} f_{current}(g^*_{ik}, x_k) \qquad \text{by (4)}$$

Thus, based upon current information ($f_{current}$) MCASA will switch from g'_{ik} to $g_{ik}*$ because it has a lower associated f-cost. This contradicts the assumption that MCASA will terminate with g-thresholds g'_{ik}. ■

If x_i is not a T-node, then MCASA is not guaranteed to find 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, as the counter-example in Figure 3.8 shows.



d1	d2	f	g
0	0	3	1
0	1	1	2

d1	d3	f	g
0	0	1	2
0	1	2	1

d2	d3	g
0	0	0
0	1	10
1	0	0
1	1	10

Figure 3.8: MCASA fails on non-T-nodes

In Figure 3.8, x_1 is a non-T-node since x_3 is a child of x_2 not x_1. If it is assumed that x_1 has a g-budget of 3 and only one value in its domain, then it must choose how to split its g between its two children. Based upon the functions on the links, it will choose to give a g-threshold of 2 to x_2 and 1 to x_3, leading to a predicted f-cost of $2 + 1 = 3$. This effectively removes the value 0 from x_3's domain and causes the link between x_2 and x_3 to incur a g-cost of 10, which in turn leads x_2's g-constraint to be unsatisfiable for any g-threshold it could receive from x_1 (x_2 tries out both values under given threshold of 2). Since increasing x_2's g-threshold does not stop x_2's g-constraint from being violated[2], x_1 infers that the problem is unsatisfiable. It is at this point that condition (3) from the previous proof has been violated since x_1 estimates $f_{current}(gt, x_2) = \infty$ whereas $f_{actual}(gt, x_2) = 4$ where $gt \in \{1, 2, 3\}$. This counter example is based on having just a g-function on the $x_2 - x_3$ link, but, similar examples can be constructed using just an f-function or both an f- and a g-function on the link. Modifications to MCASA to make these cases feasible is an issue for future work. ■

The final issue is that of MCA's runtime and space complexity. The original DCOP problem is known to be NP-hard. The algorithm for solving the MC-DCOP problem adds at most n additional nodes, so the runtime complexity class has not worsened. 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 up to n virtual variables have been added, so the space complexity for MCAP and MCAS is $(|D_i|+1)n$. In MCASA there are no virtual variables, but each node stores

[2]Variable x_1 will not try lowering x_2's g-threshold because logically a lower g-threshold should only worsen the situation.

a g-to-f mapping for each of its children. The addition of the mapping causes the space complexity to be $|D_i|\,n + G_i n$.

3.5 Experimental Results for MCA

This section presents five sets of experiments. The first compares the performance of MCAP, MCAS and MCASA on four settings that were motivated by the distributed meeting scheduling and distributed software development domains described in Section 2.2. Setting 1 comprises 20-node problems, with 3 values per node, an average link density of 1.9 and maximum link density of 4. It has 100% T-nodes and both the f- and g-costs were randomly chosen from a uniform distribution varying from 0 to 10. Setting 2 is similar to setting 1, except that the graph is 85% T-node (which increases the average link density to 2.2) to allow for comparison of the impact of T-nodes. Settings 3 and 4 are similar to settings 1 and 2 respectively, except that they are 10-node problems. Fifteen sets of constraint functions were created for each of the domain settings, so each data-point in the graphs in this section is an average over 15 problem instances.

While the four settings were motivated by examples from the meeting scheduling and software development domains in Section 2.2, three modifications were made to draw out interesting features of the MCA algorithm.

- In order to run each of the MCA subalgorithms in isolation on a single problem, the 100% T-Node settings were created. However, a 100% T-Node structure would be more realistic in domains with a strictly hierarchical setting. In the meeting scheduling and software development domains, the 85% T-Node settings provide a realistic group structure.

- MCA can only be simulated on a single machine for problems up to 30 variables in size. Thus, in order to test the g-constraint handling of the MCA algorithm on greater numbers of variables, g-constraints were added to all variables, not just the group leaders or team liaisons.
- The f- and g-costs were randomly assigned from a uniform distribution rather than tailored to specific scheduling preferences like "afternoons are better than mornings" to prevent the accidental introduction of bias into the constraint functions.

While these modifications mean that the settings do not represent actual instances of meeting scheduling or software development, they are reflective of these types of problems. Additionally, the experimental results obtained using them are no better than the performance of MCA on actual domain problems because having a larger number of g-constraints and having randomized preferences rather than ones with patterns makes the problem instances more difficult for MCA to solve.

To highlight the tradeoff between the subalgorithms, Figure 3.9 shows the performance of each subalgorithm when applied to all the nodes in a problem i.e. either MCAP is applied to all the nodes or MCAS or MCASA. Figure 3.9a shows the average run-times of MCAP, MCAS, MCASA in settings 1 and 3. The x-axis shows the g-budget applied to all variables and ranges from 0, which is unsatisfiable, to 40, which is effectively a singly-constrained problem. The y-axis shows runtime, which is measured in cycles where one cycle consists of all agents receiving incoming messages, performing local processing and

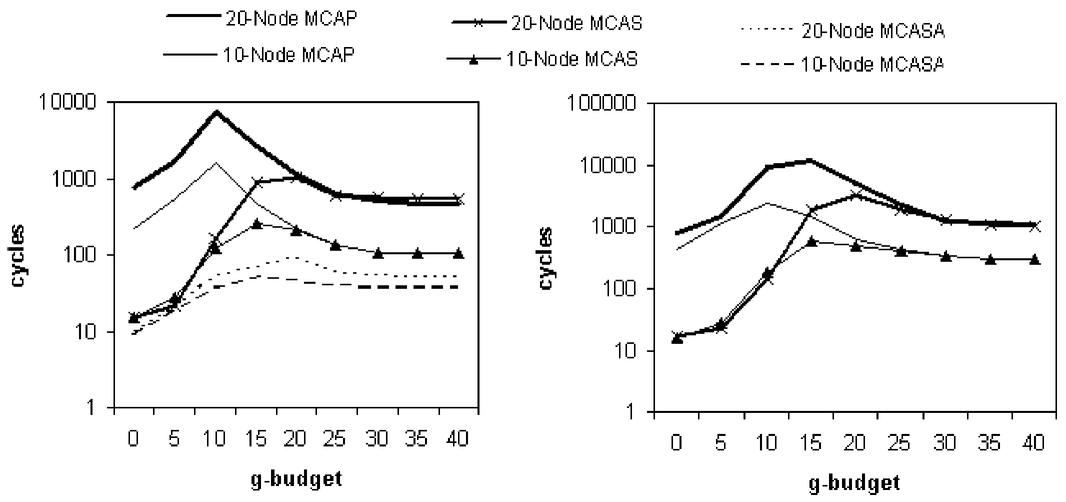
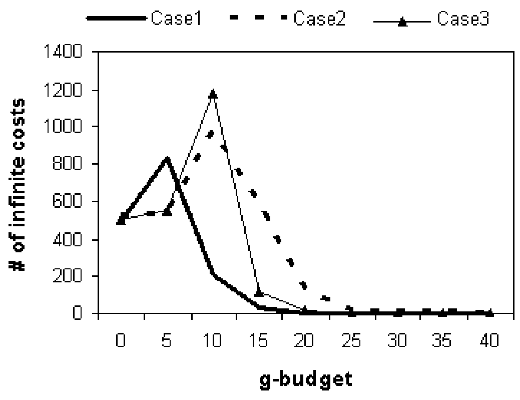


Figure 3.9: g-budget vs. run-time for a) 100% T-node problems b) 85% T-node problems

sending outgoing messages[36]. The y-axis is logarithmically scaled. The y-axes are not identical in the two graphs.

The graphs show that MCAP has the poorest performance. This result is caused by its preserving the privacy of g-constraints. 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 3.9b demonstrates similar results for setting 2 and setting 4. Only MCAP and MCAS results are shown given the switch to 85% T-node problems in these settings. MCASA cannot be applied to all the nodes in these settings. The switch from 100% T-node to 85% T-nodes causes a significant increase in run-time.

For all of the subalgorithms, the runtime curves in Figure 3.9 have a distinct inverted U-shape: the run-times are lowest at high g-budgets, which correspond to no resource



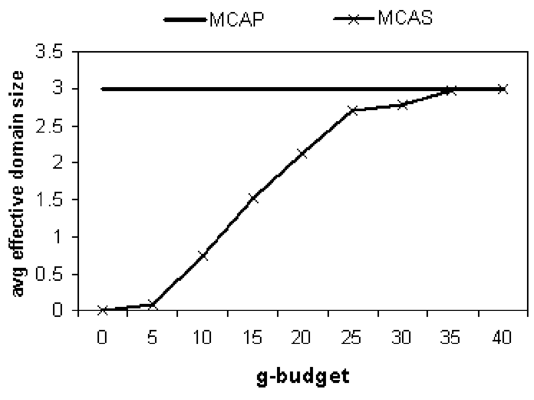


Figure 3.10: a) g-budget vs. number of infinite cost messages b) g-budget vs. number of values per domain

constraints, and at low g-budgets, which correspond to tight resource constraints. This suggests that bounded optimization problems are most challenging when the resources are sufficient but not plentiful. The data shown in Figure 3.10a suggest an explanation. This figure plots the g-budget on the x-axis and the total number of infinite cost messages received by any variable from a virtual variable. The figure shows results from three representative cases from setting 3 when running MCAP. The same hump can be seen appearing at a g-budget of 10 and diminishing to almost 0 at a g-budget of 20. This shape is consistent with the fact that the maximum g-cost on a link is 10 and the average link density is 1.9 (there are still one or two infinite cost messages all the way up to a g-budget of 35 because the maximum link density is 4). The larger number of infinite cost messages in the mid-g range indicates that it takes longer to discover unsatisfiability of solutions, leading to longer run-times and the hill shape.

In all settings, for low g-budgets, the MCAS algorithm outperforms MCAP. However, for high g-budgets, there is no difference in performance. The narrowing of the performance difference is based on the fact that MCAS prunes fewer domain values when the g-budget is high. In Figure 3.10b, the g-budget is plotted on the x-axis and the average number of values remaining in the domain of a variable running MCAS on the y-axis. The numbers are plotted as an average over all nodes over all 15 problem instances of setting 3. For comparison, results are also provided for MCAP, which performs no pruning and hence is a flat line. This figure shows that when g-budgets are tight, MCAS provides significant pruning, but when g-budgets are loose MCAS upper-bounds provide no pruning (in comparison with MCAP) and thus there is no efficiency loss due to privacy. The domain sizes converge at a g-budget of 25, which is also where the runtimes converge in Figure 3.9.

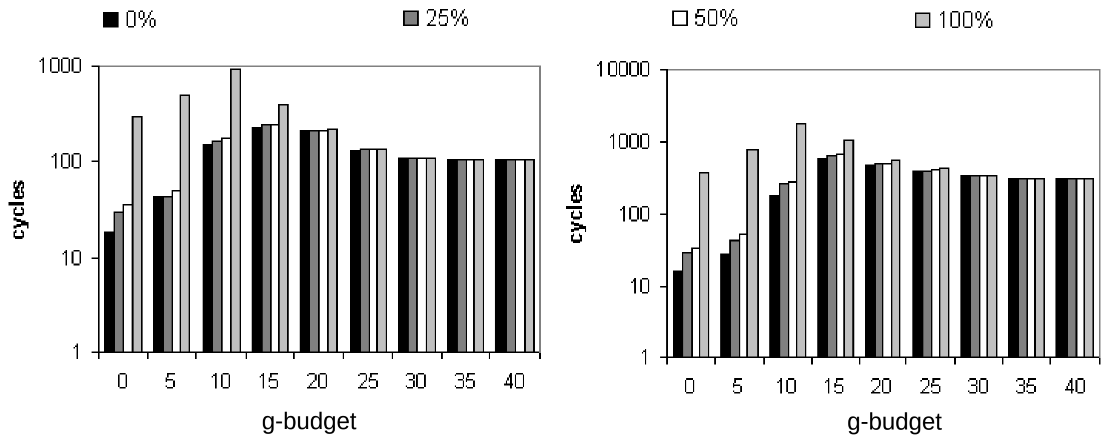


Figure 3.11: g-budget vs. runtime, varying percentages of private constraints: a) 100% T-node and b) 85% T-node problems

In real domains, there may be situations in which only some of the agents are concerned about privacy. For example, in meeting scheduling domains, people may trust different individuals or organizations differently with their budgetary information. However, to prevent bias, the experiments in Figure 3.11 randomly assign g-constraints to be private. These experiments demonstrate the benefits of the per-node application of the different subalgorithms: MCAP, MCAS and MCASA. Here, the examples from settings 1 and 2 from Figure 3.9a and b were taken and 0%, 25%, 50% and 100% of the nodes were randomly assigned to have private g-constraints while the remaining were assumed to be non-private. In the 25% and 50% cases, all three of the subalgorithms (MCASA, MCAS and MCAP) were executing simultaneously. At 0% private only MCAS and MCASA were executing and at 100% private only MCAP was executing. The results are shown in Figure 3.11a and b. The x-axis again shows the g-budget 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 constraint is private increases, the runtime increases for smaller g-budgets. Interestingly, the increase in run-time is not proportional to the increase in the number of private g-constraints; there is a significant jump in run-time when all nodes have private g-constraints. However, as in Figure 3.9, when the g-budget on each variable is loose, the runtimes converge because no pruning takes place.

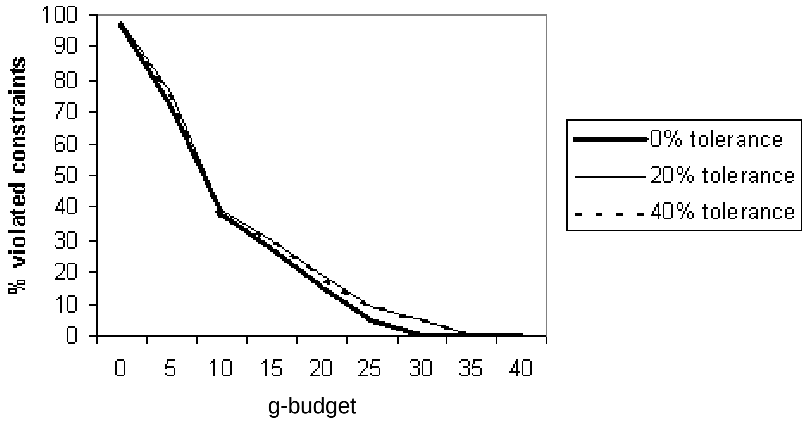


Figure 3.12: g-constraint violation by Adopt

One question that arises is whether singly-constrained algorithms could automatically satisfy the g-constraint simply by loosening their tolerances on optimal solutions. This was examined by measuring the degree to which the g-constraint is violated by Adopt when ignoring g but specifying a tolerance on f. Figure 3.12 shows the results with each data point representing an average over 5 randomly generated runs. The x-axis gives the g-budget applied to each variable and again varies from 0 to 40. The y-axis is the percentage of nodes whose g-constraint was not satisfied by the solution obtained by Adopt. The tolerance is a percentage of the difference between the optimal and maximum cost solutions. The key result is that even with a large 40% tolerance on f, large numbers of g-constraints are violated with small g-budgets. Thus, running singly-constrained algorithms with tolerances is inadequate in addressing multiply-constrained DCOPs.

This chapter has presented a complete algorithm (MCA) to solve the MC-DCOP problem defined in chapter 2. MCA builds upon Adopt and tailors its performance to the specific privacy/efficiency requirements of each agent. A detailed description of the algorithm, proofs of completeness, and experimental results were presented. The next chapter emphasizes the other side of the completeness/scalability tradeoff by presenting locally optimal algorithms for solving MC-DCOP problems.

Chapter 4

Incomplete MC-DCOP algorithms

While complete algorithms like those given in Chapter **??** have the advantage of finding the globally optimal apportionment of scarce resources, their completeness limits their efficiency and scalability. In some domains finding the global optimal is more important than finishing rapidly, while in others the priorities are reversed. To address domains where scalability and efficiency are of primary importance, this chapter describes a new set of incomplete algorithms for solving bounded optimization DCOPs. Two of the three primary techniques used in the complete algorithms also play a role in the incomplete algorithms: constraint graph transformation and dynamically constraining search.

Section 4.1 describes k-optimality, which is a way of classifying locally optimal algorithms. It also describes the MGM algorithms, which provide the basis for the locally optimal bounded optimization DCOP algorithms developed as part of this thesis [44]. Sections 4.3 and 4.4 describe the algorithms developed in this thesis: Multiply-Constrained

MGM-1 (MC-MGM-1) and Multiply-Constrained MGM-2 (MC-MGM-2), which are collectively referred to as the MC-MGM algorithms. Section 4.5 presents proofs of termination and multiply-constrained k-optimality. Finally, section 4.6 presents experimental results for the incomplete, bounded optimization algorithms.

One important caveat is that complete algorithms like Adopt have traditionally defined the DCOP problem as one of minimizing a global cost function. However, incomplete algorithms like MGM have traditionally defined DCOP in terms of maximizing a global utility function. Thus, Chapter **??** discussed problems of minimizing cost, whereas this chapter will frame the problem as one of maximizing utility.

4.1 Background:MGM

K-optimality is a way of describing how local an optimal solution an algorithm is designed to reach [46, 44]. This section will describe k-optimality and how it can be applied in bounded optimization domains. It will also describe a 1-optimal algorithm (MGM-1) and a 2-optimal algorithm (MGM-2) that are the basis for the multiply-constrained incomplete algorithms described in section 4.2 [32, 44].

4.1.1 K-optimality

K-optimality is a useful feature of an incomplete algorithm, because prior research has established two types of theoretical guarantees on the quality of the final solution of k-optimal DCOP algorithms [46, 44]. The first type fixes an upper bound on the number of k-optima that can occur in a problem. The second type establishes a lower bound on the quality of the k-optimum as a percentage of the quality the globally optimal solution.

These results allow users to estimate the benefits of using a particular level of k-optimum on a particular problem because the efficiency/performance tradeoff has been rigorously quantified. Recent research has extended the theoretical results of k-optimality to cover DCOPs with n-ary constraints similar to those used in MC-DCOP [44].

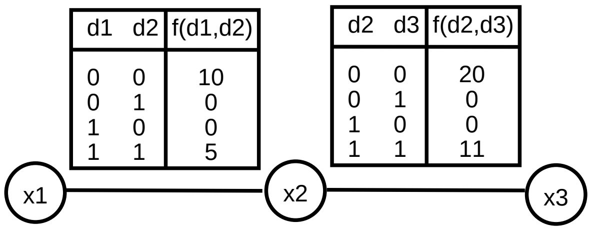


Figure 4.1: K-Optimality example.

A k-optimal solution is one where no group of k or fewer variables could make a coordinated value change and improve the quality of the overall solution. A 1-optimal solution is one where no individual variable could change its value and improve the quality of the team solution. Figure 4.1 gives an example that illustrates k-optimality. In this example, x_1, x_2 and x_3 are variables belonging to different agents. The f-reward functions are shown on the two links. In this example, assignments {0,0,0} and {1,1,1} are 1-optima. The assignment {0,0,0} is 1-optimal because given this solution, variable x_2 has no motivation to switch values from 0 to 1 because x_2's local reward would diminish from 30 to 0. Similarly, x_1 and x_3 have no motivation to switch from 0 to 1 because their local utility would decrease from 10 to 0 and from 20 to 0 respectively. In a 2-optimal solution, by contrast, no pair of agents are motivated to change their values. Thus in Figure 4.1 only {0,0,0} is a 2-optimal solution, because it would be profitable for variables x_2 and

x_3 to make a coordinated move away from the assignment {1,1,1}. An algorithm which is designed to reach a 1-optimum is a 1-optimal algorithm.

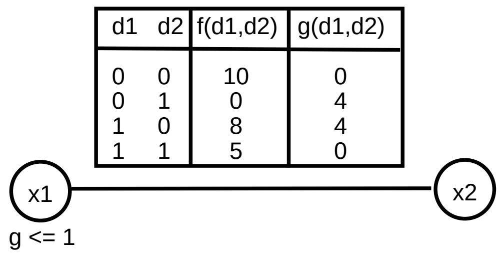
The definition of k-optimality has to be modified when dealing with bounded optimization domains, because the added resource constraints may prevent a set of variables from switching their values to an assignment which would improve the quality of the global solution. This thesis defines a modification to k-optimality called mc-k-optimality to address this issue.

- An *mc-k-optimum* is a solution where no group of k or fewer variables can change their values without either a) failing to improve the overall solution quality or b) having at least one g-constraint violation in the resulting assignment.

The example in Figure 4.2 demonstrates the change. In this example, x_1 and x_2 are variables belonging to different agents and x_1 has a g-budget of 1. The f-reward and g-cost functions are shown on the link between x_1 and x_2. While {1,1} would not be considered a 1-optimum under the traditional definition of 1-optimality, it is an mc-1-optimum, because x_2 cannot change its value to the more profitable assignment of 0 due to x_1's g-constraint.

As with regular k-optimality, all mc-k+1-optima are also mc-k-optima, however, not all mc-k-optima are mc-k+1-optima. This subset relation is inherent in the definition of an mc-k+1-optima, which states that no group of k+1 *or fewer* variables can improve the solution.

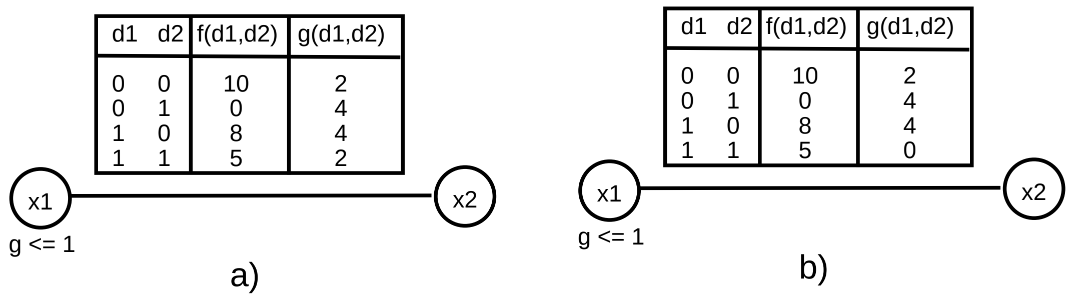
An mc-k-optimal solution may be an unsatisfying solution in the sense that some g-constraints are violated. This can occur for 2 reasons. First, the problem may be a *globally*



d1	d2	f(d1,d2)	g(d1,d2)
0	0	10	0
0	1	0	4
1	0	8	4
1	1	5	0

Figure 4.2: Modified K-Optimality example.

unsatisfiable problem, which is a problem where there does not exist an assignment such that all g-constraints are satisfied. The problem in Figure 4.3a is globally unsatisfiable. None of the four possible assignments satisfies x_1's g-constraint. The second source of unsatisfying mc-k-optima is when the k-optimal algorithm cannot reach a satisfying solution by making local moves. The problem in Figure 4.3b is not globally unsatisfiable since the assignment {1,1} satisfies x_1's g-constraint. However, in a k-optimal algorithm, the assignments able to be explored are limited by the starting assignment. In hill-climbing algorithms like MGM and Multiply-Constrained MGM, the starting assignments are chosen stochastically. If an mc-1-optimal algorithm were to start out with an initial assignment of {0,0}, there would be no way for a single variable to change its value and reach a satisfying assignment. This would still be considered terminating at an mc-1-optimum because if no group of 1 or fewer variables can change values and reach a satisfying solution. Thus {0,0} would be mc-1-optimal even though it contained a g-constraint violation. If an mc-1-optimal algorithm were to start out in any of the other three assignments, it would be able to find the satisfying mc-1-optimal assignment {1,1}.



d1	d2	f(d1,d2)	g(d1,d2)
0	0	10	2
0	1	0	4
1	0	8	4
1	1	5	2

d1	d2	f(d1,d2)	g(d1,d2)
0	0	10	2
0	1	0	4
1	0	8	4
1	1	5	0

Figure 4.3: Unsatisfying mc-k-optima.

4.1.2 MGM-1

A 1-optimal algorithm only considers unilateral actions by agents in a given context. This thesis builds a 1-optimal algorithm based on the MGM-1 (Maximum Gain Message-1) Algorithm [32, 44] which is a modification of DBA (Distributed Breakout Algorithm) [61]. The other 1-optimal DCOP algorithms that exist will be discussed in Chapter 6.

In MGM-1, variables begin by taking on an initial randomly selected assignment. Then execution continues in rounds. A *round* is defined as the duration for the system to move from one value assignment to the next. A round could involve multiple messaging phases. Every time a messaging phase occurs, it is counted as one *cycle*. During a round of MGM-1, each agent broadcasts a gain message to all its neighbors that represents the maximum change in its local utility if it is allowed to act under the current context. An agent is then allowed to act if its gain message is larger than all the gain messages it receives from its neighbors (ties can be broken through variable ordering or another method). For example, if the variables in the example in Figure 4.1 had initially selected the assignment {0,1,0} then x_1 would send a gain message to x_2 indicating it could switch values from 0 to 1 and achieve a gain of 5. x_3 would also send a gain message to x_2 indicating it could change its value and achieve a gain of 11. x_2 would send a proposal message to both x_1 and x_3 indicating it could switch values and achieve a gain of 30. Since x_2 has the highest gain, it will switch its value to 0 and the other two variables will remain at their current assignment. Execution continues until no further proposals are made. MGM-1 requires two cycles per round [32, 44].

4.1.3 MGM-2

With a 1-optimal algorithm, the evolution of the assignments will terminate at a 1-optimum. One method to improve the solution quality is for agents to coordinate actions with their neighbors. This allows the algorithm to break out of some local optima. This section introduces the 2-optimal algorithm MGM-2 (Maximum Gain Message-2) [44].

As with MGM-1, agents initially take on a random assignment and then begin executing rounds of the MGM-2 algorithm. MGM-2 uses randomization to decide which subset of agents are allowed to make *offers*. Each agent is randomly assigned to be an *offerer* or a *receiver*. Each offerer will choose a neighbor at random and send it an offer message which consists of all coordinated moves between the offerer and receiver that will yield a gain in local utility to the offerer under the current context. The offer message will contain both the suggested values for each player and the offerer's local utility gain for each value pair. For example, suppose the agents in the example in Figure 4.1 had currently taken on the assignment $\{x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0\}$ and x_1 had been assigned to be an offerer, while x_2 and x_3 had been made receivers. Agent x_1 could send a proposal to x_2 that they change values from $\{x_1 \leftarrow 1, x_2 \leftarrow 1\}$ to $\{x_1 \leftarrow 0, x_2 \leftarrow 0\}$ for a gain of 5 from x_1's perspective. Given that x_3 is not a neighbor of any offerers, it will not receive an offer in this round.

After it gets an offer, each receiver calculates the overall utility gain for each value pair in the offer message by adding the offerer's local utility gain to it's own utility change under the new context and subtracting the difference in the link between the two so it is not counted twice: $\sum_{y \in Neighbors(x_i)} Gain_{iy} + \sum_{z \in Neighbors(x_j)} Gain_{jz} - Gain_{ij}$. In the example in Figure 4.1, Agent x_2 would receive the offer and calculate that their combined gain from this move would be 25. If the maximum overall gain is positive, the receiver will send an *accept* message to the offerer and both the offerer and receiver are considered to be committed. Otherwise, it sends a *reject* message to the offerer, and neither agent is committed.

Uncommitted agents choose their best local utility gain for a unilateral move and send a proposal message. Uncommitted agents follow the same procedure as in MGM-1, where they modify their value if their gain message was larger than all the gain messages they received. Committed agents send the global gain for their coordinated move. Committed agents send their partners a *confirm* message if all the gain messages they received from their neighbors were less than the gain for the coordinated move they plan to make. They send a *deconfirm* message, otherwise. A committed agent will only modify its value if it receives a go message from its partner. MGM-2 requires five cycles (value, offer, accept/reject, gain, confirm/deconfirm) per round in contrast to MGM-1's 2 cycles per round.

4.2 Multiply Constrained MGM

As mentioned in the Introduction, there are four main challenges that must be addressed in designing locally optimal multiply-constrained DCOP algorithms: search complexity, harnessing existing algorithms, privacy/efficiency, and unsatisfiability detection. The unsatisfiability detection challenge is more relevant to incomplete than complete algorithms because incomplete algorithms do not systematically consider all possible assignments and thus cannot easily detect unsatisfiability. There are two approaches to this challenge: a) require a valid initial starting point and maintain a satisfaction invariant or b) detect when the search has covered all assignments without any being found that are satisfying. This thesis uses approach a), but this still leaves the challenge of detecting when it will be impossible to find a valid start point. The incomplete Multiply-Constrained DCOP algorithms presented in this chapter address the unsatisfiability detection challenge with a carefully defined dummy value which is added to variables' domains so that they can easily find a valid start point and also flag a local constraint violation that remains even at termination.

4.3 MC-MGM-1

MC-MGM-1 may be thought of as containing two separate subalgorithms that can operate simultaneously on the same problem. The first is the shared MC-MGM-1 subalgorithm that makes g-constraint information available to neighboring variables. The second

```
MC-MGM-1 ($allNeighbors$, $currentValue$)
(1)  if !$private$
(2)     SendValueMessages($allNeighbors$, $currentValue$, $available\text{-}g\text{-}budget$)
(3)  else
(4)     SendValueMessages($allNeighbors$, $currentValue$)
(5)  $currentContext$ = GetValueMessages($allNeighbors$)
(6)  for all $newValue$ in EffectiveDomain($currentContext$)
(7)     [$gain$, $newValue$] = BestUnilateralGain($currentContext$)
(8)  if $gain > 0$
(9)     SendGainMessage($allNeighbors$,$gain$, $newValue$)
(10) $neighborGains$ = ReceiveGainMessages($allNeighbors$, $NeighborValues$)
(11) if GConstraintViolated($newContext$) and !$virtual$
(12)    $n$ = SelectNeighborsToBlock()
(13)    SendBlockMessages($n$)
(14) else if GConstraintViolated($newContext$) and $virtual$
(15)    $n$ = SelectNeighborsToBlock()
(16)    SendNogoodMessages($n$)
(17) if ReceivedNogoodMessage()
(18)    $NoGoods = NoGoods$ + [$newValue$, $newContext$]
(19) else if $gain >$ max($neighborGains$) and !ReceivedBlockMessage()
(20)    $currentValue = newValue$
```

Figure 4.4: Multiply-Constrained MGM-1 Pseudo-code

is the private subalgorithm which encapsulates g-constraint information in virtual variables. The pseudo-code for the combined MC-MGM-1 is shown in Figure 4.4 and will be explained in the next two subsections.

4.3.1 Shared MC-MGM-1

MC-MGM-1 maintains an invariant that at no point during execution does the current assignment violate any agent's g-constraint. All moves are calculated to go from one assignment where this invariant holds to another assignment where it holds. However, the tricky part is finding the first assignment where this holds. In order to address this issue as well as to provide a mechanism for determining when a problem is unsatisfiable, MC-MGM-1 performs an initialization step when it first begins where it adds a dummy value to each variable's domain. This dummy value is used as the starting value for each variable. The dummy value, d', is defined to have the following constraint function on all links:

- $f(d', d') = c$
- $g(d', d') = 0$
- $f(d', d_i) = f(d_i, d') = k$
- $g(d', d_i) = g(d_i, d') = 0$
- c and k are constants such that $c < k < 0$
- d_i is a regular domain value

This definition ensures that all variables can start at an assignment that spends 0 g, which is a satisfying solution. However, since the quality in terms of f is by definition lower than any real assignment, MC-MGM-1 will attempt to find a solution that does not involve dummy values. The reason that c must be smaller than k is that since MC-MGM-1 only allows one variable to move at a time, it must be profitable for nodes to move from a dummy value to a real value even if their neighbors are all still set to their dummy values. Otherwise the initial assignment becomes an mc-1-optimum and termination occurs immediately. If MC-MGM-1 fails to find a valid assignment containing no dummy values then the problem is 1-optimally unsatisfiable, since MC-MGM-1 will always favor real domain values over the dummy one. So the dummy values also allow for easy detection of 1-optimal unsatisfiability.

After initialization, each variable repeatedly runs rounds of the pseudo-code shown in Algorithm 4.4. Note that each round involves multiple cycles of communication and execution. The first thing the variables do is send value messages to their neighbors informing them of their current value as well as how much g-budget is currently available for use by that particular neighbor. The available-g sent to node x_j equals total g minus the g currently consumed by all of x_i's other neighbors (line 2 in Algorithm 4.4). This is similar to the available-g sent out during MCAS and, like with MCAS, this can lead to overspending if multiple variables try to use the full available-g.

After receiving the value messages, each node calculates its effective domain (line 6). This means removing from consideration any values that given the current context would violate either the variable's own g-constraint or any of its neighbors' available-g's. The node then considers all of the values in the effective domain and picks the one that would allow it to gain the largest increase in local f, if selected (lines 6-7). It then sends a gain message to all of its neighbors proposing the move and listing the gain that it would achieve (lines 8-9).

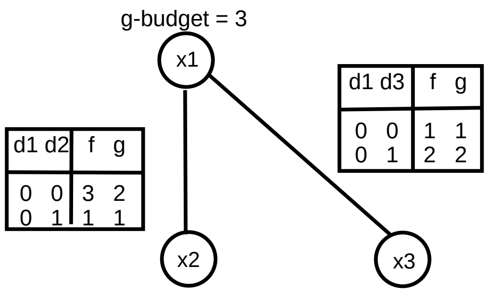


Figure 4.5: Blocking situation

Upon receipt of gain messages, two things occur. First the variable looks to see if any of its neighbors can achieve a better gain and if so rescinds its intention to move (line

21). Second, each variable checks to see whether the combined expenditure from all of its neighbors' proposed moves violates its g-constraint (line 11).

An example of this situation is shown in Figure 4.5. Variable x_1 has taken on the value 0 and receives two move proposals from its neighbors x_2 and x_3. The neighbors are assumed to currently have taken on the dummy value, thus each sees an available-g of 3. They make the following move proposals, neither of which individually violate the available-g: x_2 proposes taking on 0 and x_3 proposes taking on 1. However, if both moves are made, x_1's g-constraint will be violated. If a node detects this situation, then it will send a blocking message to a subset of the offending neighbors (lines 12-13). For example, x_1 may choose to send a blocking message to x_3. (The heuristics for selecting neighbors will be discussed in section 4.3.3.) If a variable receives a block message, then it will not move in the current round. The blocks are temporary and thus an agent is free to consider moving in the next round. Those agents with the highest local gain in the current round who don't receive blocking messages will move. In the example from Figure 4.5, x_3 will not change from the dummy value, but x_2 will go ahead and take on the value 0. The available-g will then be updated and x_3 will be informed in the next round that it can only propose moves that spend no more than 1 unit of g. These rounds repeat until all agents have ceased to propose moves.

4.3.2 Private MC-MGM-1

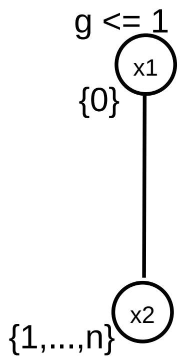
The private MC-MGM-1 subalgorithm works like the shared version with two modifications. First, variables cannot explicitly take into account their private neighbors' g-constraints when proposing moves and second, a private variable's g-constraint is encapsulated in a virtual variable. Note that the virtual variable's functionality could be incorporated into the original variable itself. The reason for this is that unlike in MCA, there is no priority ordering among neighbors, so the original variable will receive all the necessary messages to act as its own virtual variable. However, for simplicity of explanation, this thesis will treat the virtual variable as if it is a separate entity from the original variable.

Initialization in Private MC-MGM-1 is the same as that of shared MC-MGM-1 except that for each variable, x_i, that has a private g-constraint, a virtual variable, x_i' is created and connected both to x_i and all of x_i's neighbors. The virtual variable x_i' has no domain and its links to other variables have no f- or g-cost functions on them, they are purely used for communication.

All of the non-virtual variables select their most profitable potential value, just like in shared MC-MGM-1. However, in this case, the EffectiveDomain() function will not weed out values that would overspend a neighbor's g-budget (lines 4 and 6), if that neighbor has a private g-constraint. One other difference is that EffectiveDomain() will consult a list called Nogoods, which lists values for which a Nogood message was received and the context during which it was received. If the current context is the same, then the value from the Nogood list will be eliminated from the effective domain (line 6). These Nogood messages are received from virtual variables and unlike the regular blocking messages of shared MC-MGM-1, they are maintained throughout execution of the algorithm, thus making them a permanent block (lines 18-19). They are made permanent because agents have no other way to know what value assignments to avoid because their neighbors' g-constraints are private.

The non-virtual variables send out their proposed moves to all of their neighbors (lines 11-12). At this point the virtual variables use the information from the proposal messages to evaluate whether the g-constraint of the variable they represent will be violated. If it will, then they send Nogood messages to some subset of the variables involved in the g-constraint violation (lines 14-17).

If a Nogood message is received by a node then it is added to the Nogoods list along with the appropriate context (lines 18-19). If not, then nodes make their moves just as if they were running shared MC-MGM-1.



d1	d2	f	g
0	1	n	n
0	2	n-1	n-1
...	...	...	...
0	n	1	1

Figure 4.6: Shared vs. Private MC-MGM

Private MC-MGM-1 works like shared MC-MGM-1 in many respects except that it does not consider its neighbors' g-constraints a priori and its messages are more strongly tied to the current context. As mentioned in Chapter **??** the current context is the current assignment of values to neighboring variables. It is possible to demonstrate in a simple example the contrast between the two algorithms and the reason for shared MC-MGM-1's greater efficiency. In Figure 4.6 a simple two variable example is shown. x_1 has a single value in its domain (for simplicity the dummy value has been omitted), while x_2 has n values in its domain. The constraint function for the single link is shown to the right. Assuming that x_1 has made the initial move from it's dummy value to the value 0, when x_2 comes to move, if x_1's g-constraint is shared then x_2 will prune all values from its domain except n and when it proposes moving to the value n, x_1 will not block the move and execution will terminate immediately after moving. In contrast, if x_1's g-constraint is private then x_2 will propose moving to value 1, which will be blocked by a Nogood message, then in the next round x_2 will propose moving to value 2, and so on through n rounds of execution before it finally tries the value n and terminates.

4.3.3 Heuristics in MC-MGM

Under certain circumstances a variable x_i may receive proposed moves from multiple neighbors such that while no individual move violates x_i's g-constraint, the combined set of moves violates x_i's g-constraint. In this case x_i must send one or more blocking messages. The number of blocking messages sent is the minimum number that will prevent x_i's g-constraint from being violated, which will be at worst one fewer than the number of x_i's neighbors proposing moves (since each individual move is legal). Picking the optimal neighbor to block only using local information is impossible, but there are several possible heuristics for selecting which neighbor(s) to block. Heuristics can be deterministic or stochastic. Deterministic heuristics have the advantage of not ignoring local information in deciding who to block. However, local information may not indicate the globally optimal choice, and so stochastic heuristics have the advantage, when run multiple times, of being able to eventually find the optimal set of neighbors to block. Additionally, there are two ways to handle a blocking message: 1) x_j maintains its old value and chooses not to make its proposed move (monotonic) 2) x_j resets itself to a value that consumes less g (non-monotonic). Monotonic heuristics are guaranteed to terminate, but non-monotonic heuristics provide more options for breaking out of a local optimum. For this thesis, four different heuristics were selected as representative examples of possible heuristics. While examples can be created that cause particular problems for an individual heuristic, experimental results (see Figure 4.11) demonstrate that on randomly generated examples, the different heuristics produce similar quality results. The four heuristics used are as follows:

- *Monotonic*: x_i selects one or more random neighbors and sends blocking messages. The blocking messages are interpreted by each neighbor x_j to mean that x_j should refrain from changing its value in the current round. The advantage of this heuristic is that it maintains the property of monotonicity which was a property of the original MGM algorithms. The global utility never decreases during execution which allows proof of termination to be guaranteed. However, experimentally, this heuristic is the worst performing of the heuristics on random examples.

- *Random Reset*: x_i selects one or more random neighbors and sends blocking messages which are interpreted by each neighbor x_j to mean that it should reset its value to the dummy value. This heuristic, when run multiple times, allows MC-MGM to eventually send its blocking message(s) to the optimal neighbor(s). The disadvantages are that monotonicity cannot be guaranteed and that random reset will not consider changing its own value to prevent the violation.

- *Self*: x_i sends no blocking messages but instead resets itself to the dummy value. In this case, one fewer cycle of communication is required. However, monotonicity is not guaranteed. This is also a deterministic heuristic.

- *Biggest Spender*: x_i selects the neighbor x_j that is using the greatest amount of x_i's g-budget and sends a blocking message which is interpreted by x_j to mean that it should reset its value to the dummy value. This heuristic attempts to avoid the pitfall of random neighbor selection by choosing a neighbor whose reseting will free up the greatest amount of g. However, it has the disadvantages of a deterministic heuristic.

4.4 MC-MGM-2

MC-MGM-2 is also divided into two subalgorithms which can be run simultaneously on different parts of the same problem: Shared MC-MGM-2 and Private MC-MGM-2. The pseudo-code appears in Figure 4.7.

4.4.1 Shared MC-MGM-2

Multiply-Constrained MGM-2 operates much like MC-MGM-1 with the exception that in addition to making individual moves, it can propose joint moves between pairs of agents. At the beginning of the round agents are stochastically designated to be either offerers or receivers, this designation helps reduce redundant computation where multiple agents propose the same move (lines 6-7 in Figure 4.7). If a node, x_i, is an offerer, it will randomly select a neighbor, x_j and search for the best joint move that does not violate

```
MC-MGM-2 ($allNeighbors$, $currentValue$)
(1)  if !$private$
(2)     SendValueMessages($allNeighbors$, $currentValue$, $available\text{-}g\text{-}budget$)
(3)  else
(4)     SendValueMessages($allNeighbors$, $currentValue$)
(5)  $currentContext$ = GetValueMessages($allNeighbors$); $committed = false$
(6)  if Random(0,1) < $offererThreshold$
(7)     $committed = true$ ; $partner$ = RandomNeighbor($allNeighbors$)
(8)     for all $newValue$ in EffectiveDomain($currentContext$)
(9)        SendOfferMsg($partner$, bestCoordinatedMove($partner$))
(10) for all $newValue$ in EffectiveDomain($currentContext$)
(11)    [$gain$, $newValue$] = BestUnilateralGain($currentContext$)
(12) $offers$ = ReceiveOffers($allNeighbors$); $offerReplySet = \cup\ offers.neighbor$
(13) if !$committed$
(14)    $bestOffer$ = FindBestOffer($offers$)
(15)    if $bestOffer.gain > gain$ and $newValue$ in EffectiveDomain($currentContext$)
(16)       $offerReplySet = offerReplySet \setminus \{bestOffer.neighbor\}$
(17)       $committed = true$; $partner = bestOffer.neighbor$
(18)       $newValue = bestOffer.myNewValue$; $gain = bestOffer.gain$
(19)       SendOfferReplyMsg($partner$, $commit$, $bestOffer.partnerNewValue$, $gain$)
(20)    for all $neighbor \in offerReplySet$
(21)       SendOfferReplyMsg($neighbor$, $noCommit$)
(22) if $committed$
(23)    $reply$ = ReceiveOfferReplyMsg($partner$)
(24)    if $reply$ = commit
(25)       $newValue = reply.myNewValue$; $gain = reply.gain$
(26)    else
(27)       $committed = false$
(28) SendGainMsg($allNeighbors$, $gain$)
(29) $neighborGains$ = ReceiveGainMsgs($allNeighbors$); $changeValue$ = no
(30) if GConstraintViolated($newContext$) and !$virtual$
(31)    $n$ = SelectNeighborsToBlock()
(32)    SendBlockMessages($n$)
(33) else if GConstraintViolated($newContext$) and $virtual$
(34)    $n$ = SelectNeighborsToBlock()
(35)    SendNogoodMessages($n$)
(36) if ReceivedNogoodMessage()
(37)    $NoGoods = NoGoods + [newValue, newContext]$
(38) else if $committed$
(39)    if $gain >$ max($neighborGains$)
(40)       SendConfirmMsg($partner$, go)
(41)    else
(42)       SendConfirmMessege($partner$, noGo)
(43)    $confirmed$ = ReceiveConfirmMsg($partner$)
(44)    if $confirmed$
(45)       $changeValue$ = yes
(46) else if  $gain >$ max($neighborGains$)
(47)    $changeValue$ = yes
(48) if $changeValue$ = yes
(49)    $currentValue = newValue$
```

Figure 4.7: Multiply-Constrained MGM-2 Pseudo-code

x_i's g-constraint or the available-g of any of x_i's neighbors including x_j (line 8). Variable x_i will then send a proposal message to x_j (line 9). As with regular MGM-2, x_j will not accept a proposal if it is itself an offerer (line 13). However, those receivers that receive a proposal for a joint move will check to see if it violates their g-constraint of the available-g of any of their neighbors. If the proposed move is legal and has the highest local gain, then the node will send an acceptance message (line 19).

4.4.2 Private MC-MGM-2

Private MC-MGM-2 also uses virtual variables to encapsulate private g-constraints. The only difference between the private versions of MC-MGM-1 and MC-MGM-2 is that when blocking a joint move the virtual variable in MC-MGM-2 sends a nogood message to both nodes planning to move (lines 35-36).

4.5 Proofs

This section presents proofs that when using the monotonic heuristic, MC-MGM-1 is monotonic and terminates. Additionally, when MC-MGM-1 terminates it has found an mc-1-optimal solution. Similarly this section gives proofs that when using the monotonic heuristic, MC-MGM-2 is monotonic and terminates at an mc-2-optimal solution. Many heuristics are available for use within MC-MGM. The proofs in this section depend on the use of the monotonic heuristic from Section 4.3.3, because monotonicity is important in proving termination. The other three heuristics described earlier were experimentally found always to terminate, but they have not been proven to terminate.

The following definitions and assumptions are used in the proofs that follow:

- Variables are denoted as $x_i \in X$, where X denotes the set of all variables; values of variables are denoted as $d_i \in D_i$ where D_i is the finite domain of the variable x_i.

- $d^{(n)}$ refers to the assignment of values to variables in the DCOP due to MC-MGM at the beginning of the n-th cycle of the algorithm. The global utility of this assignment is denoted $U(d^{(n)})$.

- $L(d_i; d_{-i})$ refers to the local utility of agent i, given the current context denoted as d_{-i}. $L(d_i; d_{-i}) = \sum_{x_j \in Neighbors(x_i)} f_{ij}(d_i, d_j)$.

- $Gain_i$ is the change in local utility of x_i due to a unilateral change in its value from d_i to d'_i, given fixed context d_{-i}, i.e. the difference between $L(d_i; d_{-i})$, and $L(d'_i; d_{-i})$.

- $Gain_{ij}$ is the change in local utility of x_i and x_j due to the 2-coordinated change in values from d_i to d'_i and d_j to d'_j. This equals $L(d'_i; d_{-i}) + L(d'_j; d_{-j}) + f(d_i, d_j) - f(d'_i, d'_j)$.

- In computing $Gain_i$, x_i only considers its *EffectiveDomain*, i.e. values that do not violate its own g-constraint, or the available-g of those neighbors whose g-constraints are public. Additionally, calculating EffectiveDomain removes from consideration those values that have received nogood messages for the current context. Thus, references to Gain below, refer to gain over x_i's EffectiveDomain.

- Once an agent in MC-MGM takes on a value from its real domain, it will never go back to its dummy starting value, because the gain will be negative.

- MC-MGM maintains as an invariant that no agents' g-constraint is violated at the end of any round of execution.

Proposition 4 *In MC-MGM-1, the global utility $U(d^{(n)})$ never decreases.*

Proof:

There are two separate cases to handle: *non-blocking* and *blocking.* In the non-blocking case, no block messages or nogood messages are issued. In the blocking case at least one blocking or nogood message is issued.

In the non-blocking case, if x_i intends to modify its value in round r, then:

- $Gain_i > Gain_j, \forall x_j \in Neighbors(x_i)$
- $Gain_i > 0$

Since x_i's neighbors would have received x_i's message proposing $Gain_i$, they will not modify their values in round r. Thus, no two neighboring variables will change values simultaneously. So, when x_i changes its value, $Gain_i$ will be realized. Since x_i's gain is the sum of utilities on each link connected to x_i, x_i's gain implies that the global utility $U(d^{(r)})$ increases. If multiple variables change values simultaneously, they are guaranteed to be non-neighbors, and thus, each of their gains will add to $U(d^{(r)})$.

In the blocking case, a blocking or nogood message within a round r will cause a variable x_i which had intended to change its value not to make the change. Such a block would cause x_i to realize a gain of 0, which does not cause a decrease in $U(d^{(r)})$. The message does not affect any other variables.

■

Proposition 5 *In MC-MGM-2, the global utility $U(d^{(n)})$ never decreases.*

Proof:

Variables in MC-MGM-2 can either make individual moves or coordinated moves. The individual moves are equivalent to moves in MC-MGM-1 and have been proven not to decrease $U(d^{(r)})$. This proof will examine the effects of the coordinated moves in both blocking and non-blocking cases.

In the non-blocking case, if x_i and x_j are committed to making a coordinated value change in round r, then

- $Gain_{ij} > Gain_{kl}, \forall x_k \in \{Neighbors(x_i) \cup Neighbors(x_j) - x_i - x_j\}, x_l \in Neighbors(x_k)$
- $Gain_{ij} > Gain_k, \forall x_k \in \{Neighbors(x_i) \cup Neighbors(x_j)\}$
- $Gain_{ij} > 0$

Since x_i and x_j's neighbors would have received x_i and x_j's messages proposing $Gain_{ij}$, they will not modify their values in round r. Thus, no two neighboring variables will change values simultaneously unless part of a coordinated move. Thus when the coordinated value change is made, $Gain_{ij}$ will be realized. Since the coordinated gain is amassed from the sum of utilities on each link connected to x_i or x_j, the coordinated local gain implies that the global utility $U(d^{(r)})$ increases. If other variables change values during round r, they are guaranteed not to be neighbors of either x_i or x_j, and thus, each of their gains will add to $U(d^{(r)})$.

In the blocking case, a blocking or nogood message within a round r will cause variables x_i and x_j which had intended to make a coordinated value change not to make the change. Such a block would cause the pair of variables to realize a gain of 0, which does not cause a decrease in $U(d^{(r)})$. The message does not affect any other variables.

■

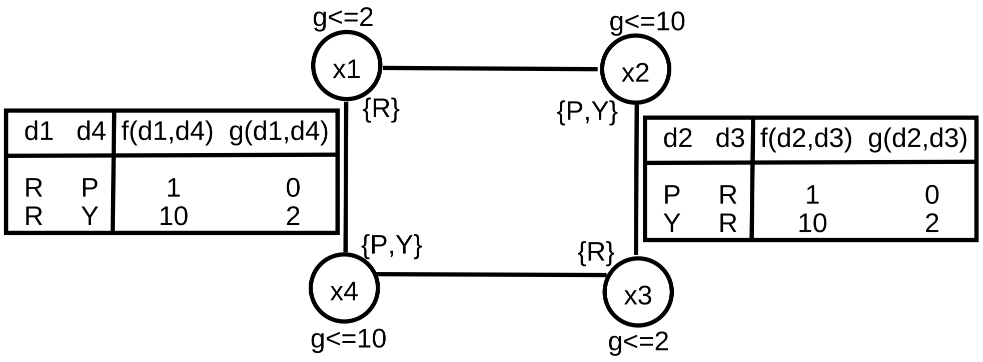


Figure 4.8: A Deadlock example for MC-MGM-1

Given that MC-MGM sends out blocking messages, there is a possibility of entering deadlock. A cycle of blocking messages could block all variables from changing values, even though they had not yet reached an mc-k-optimum. Figure 4.8 shows an MC-DCOP where it is possible to enter deadlock. There are four agents, with domains as shown. The f rewards and g costs are shown in the tables. The DCOP is initialized with all agents taking on a dummy value of 0. x_1 and x_3 switch from 0 to value R. At this point, x_2 and x_4 propose switching to Y, which gives them each a gain of 20, and Y is under the available-g of 2 (for both x_1 and x_3). Since x_1 and x_3's budgets would be violated if both x_2 and x_4 switched to the value Y, they must send blocking messages. If x_1 randomly selected x_2 to block and x_3 randomly selected x_4 neither x_2 or x_4 could change values, and this would create a deadlock situation.

In the monotonic heuristic, the agents being blocked are randomly selected. As a result, remaining in deadlock indefinitely is impossible. Suppose agents can enter deadlock with probability p, where $p \in [0, 1)$. In Figure 4.8, $p = 0.5$ because there are four ways the two blocking messages could be sent out and 2 result in deadlock. Since agents randomly select who to block each round, there is a probability of $1 - p$ of escaping deadlock in every round. After N rounds of execution, the probability of remaining in deadlock is p^N. Since execution continues until there are no longer any proposal messages being sent, N approaches ∞ and p^N approaches 0. Furthermore, once one variable is allowed to change its value, the budgets available at the remaining variables change and the old deadlock is resolved. For example in Figure 4.8, if x_2 is allowed to change its value to Y, then the available-g at x_1 and x_3 changes to 0, and x_4 will propose taking the value P in the next round.

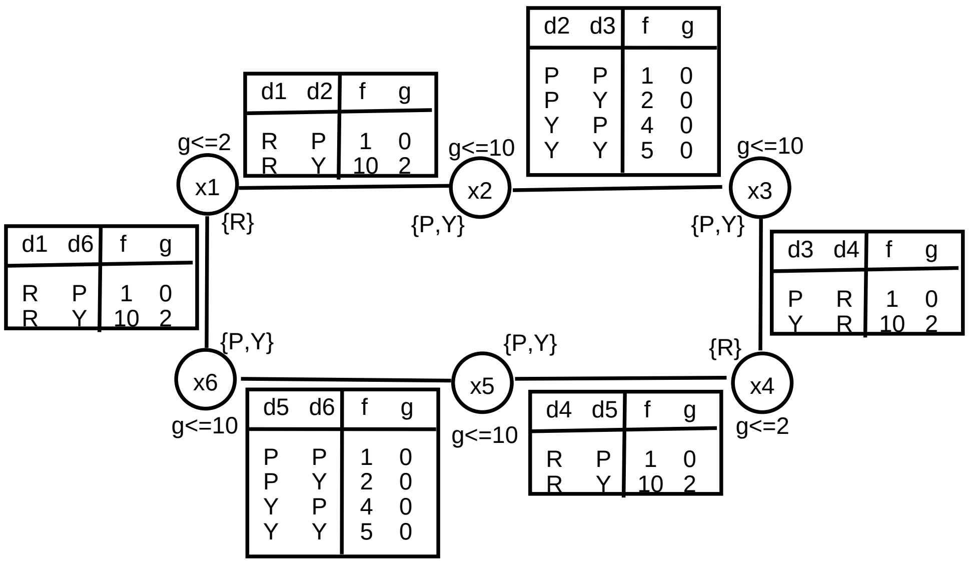


Figure 4.9: A Deadlock example for MC-MGM-2

A corresponding example of deadlock for MC-MGM-2 is shown in Figure 4.9. In this example, it is assumed that x_1 and x_4 initially switched from the dummy value to R. Now x_2 and x_3 are proposing making a coordinated move from their dummy values to $\{Y \leftarrow x_2, Y \leftarrow x_3\}$ and x_5 and x_6 want to make a coordinated move from their dummy values to $\{Y \leftarrow x_5, Y \leftarrow x_6\}$. However, if both coordinated moves are made, x_1 and x_4's g-constraints will be violated. If x_1 randomly chose to block x_2 and x_3 while x_4 chose to block x_5 and x_6, then deadlock would occur in MC-MGM-2. However, as with the previous example, remaining in deadlock is dependent on x_1 and x_4 continuing to select different pairs to block. Once again agents can enter deadlock with probability p, and in Figure 4.9, $p = 0.5$. However, after N rounds of execution, the probability of remaining in deadlock is p^N and as N approaches ∞, p^N approaches 0.

Proposition 6 *Given that deadlocks are resolved using randomization, MC-MGM-1 will terminate at an mc-1-optimal solution.*

Proof:

In Proposition 4, it was shown that MC-MGM-1 will lead to a monotonically increasing global utility $U(d^{(n)})$. Since $U(d^{(n)})$ cannot be higher than the finite globally optimal solution, MC-MGM-1 cannot keep increasing $U(d^{(n)})$ forever. Thus, assuming it eventually resolves any deadlocks it enters, MC-MGM-1 will terminate.

Termination in MC-MGM-1 occurs when no variable x_i is able to propose a move from d_i to d_i' given d_{-i} where $Gain_i > 0$ and no g-constraints are violated after the move. This situation is the definition of an mc-1-optimal, so when MC-MGM-1 terminates, the agents have reached an mc-1-optimal.

■

Proposition 7 *Given that deadlocks are resolved using randomization, MC-MGM-2 will terminate at an mc-2-optimal solution.*

Proof:

In Proposition 5, it was shown that MC-MGM-2 will lead to a monotonically increasing global utility $U(d^{(n)})$. Since $U(d^{(n)})$ cannot be higher than the finite globally optimal solution, MC-MGM-2 cannot keep increasing $U(d^{(n)})$ forever. Thus, assuming it eventually resolves any deadlocks it enters, MC-MGM-2 will terminate.

Termination in MC-MGM-2 occurs when no variable x_i is able to propose a move from d_i to d_i' where $Gain_i > 0$ and no pair of variables x_i and x_j can propose a move from $\{d_i, d_j\}$ to $\{d_i', d_j'\}$ where $Gain_{ij}$ without there being at least one g-constraint violation in the assignment reached after the move. This situation is the definition of an mc-2-optimal, so when MC-MGM-2 terminates, the agents have reached an mc-2-optimal.

■

4.6 Experimental Results for MC-MGM

This section presents five sets of experiments. The testcase suite described in Chapter 3 is employed. As in chapter 3, setting 1 comprises 20-node problems, with 3 values per node, an average link density of 1.9 and maximum link density of 4. It has 100% T-nodes (although this does not impact the algorithm usage) and both the f- and g-costs are randomly chosen from a uniform distribution varying from 0 to 10. Setting 2 is similar to setting 1, except that the graph is 85% T-node (which increases the average link density to 2.2). Setting 3 (setting 4) is similar to setting 1 (setting 2), except that it has only 10 nodes. Once again there are 15 sets of constraint functions for each domain setting. In contrast with the complete MCA algorithm, MC-MGM involves stochastic elements, specifically from three sources:

- when potential moves have identical gains, which is true at initialization, the variable that moves is chosen stochastically.
- when variables in MC-MGM-2 are selecting whether to be offerers and who to make an offer to, the decisions are made stochastically.
- when variables are sending blocking or nogood messages, 3 of the 4 heuristics use randomization.

Due to these sources of randomness, each testcase was run 100 times. The results from the 100 runs are averaged together in the graphs presented below.

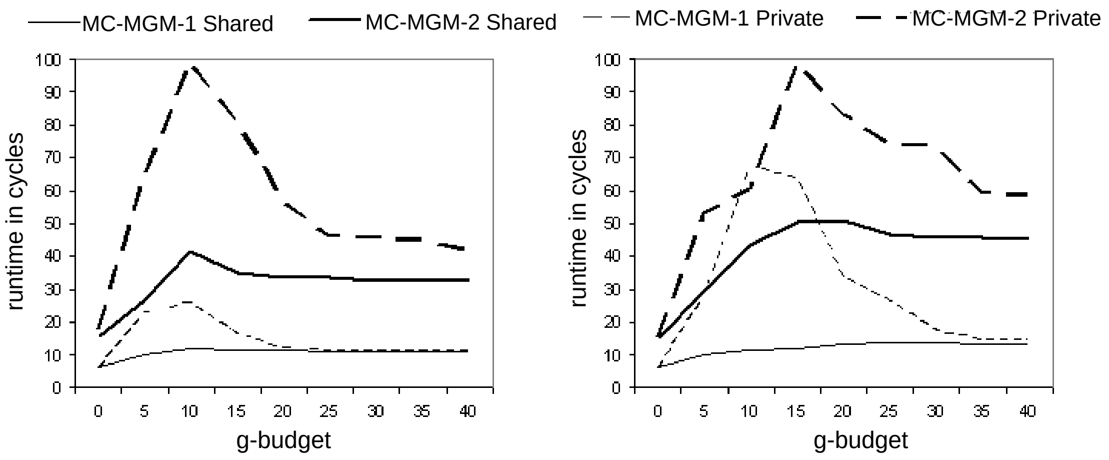


Figure 4.10: g-budget vs. runtime on a) 10-node and b) 20-node cases

This first set of experiments demonstrate the runtime savings gained by using the incomplete MC-MGM algorithms. The Random Reset heuristic described in Section 4.3.3 was used with both algorithms for the results presented in this graph because, as is demonstrated in the next Figure, it slightly outperforms the other heuristics that were tested. The examples from settings 2 and 4 from Figure 3.9 were taken and MC-MGM-1 and MC-MGM-2 (both shared and private versions) were run on the problems. The results are shown in Figure 4.10a and b. The x-axis shows the g-budget applied to each node and the y-axis measures the runtime in cycles. Each data point is an average over 100 runs of each of the 15 instances of the problem.

We can see that whereas the runtime for MCA was on the order of hundreds to thousands of cycles, MC-MGM takes only tens of cycles to run. As with MCA, the runtime peaks at a g-budget of about 10 to 15 because that is where the most complicated tradeoff between f and g is taking place. Additionally, the graphs demonstrate that the private versions of MC-MGM-1 and 2 are slower than their non-private counterparts. This is because in the private algorithms the nodes have no knowledge of how their moves will impact their neighbors' g-constraints and so they expend cycles proposing moves that are then rejected by a virtual variable.

Under certain circumstances an agent x_i may receive proposed moves from multiple neighbors such that while no individual move violates x_i's g-constraint, the combined set of moves violates x_i's g-constraint. In this case x_i must send a blocking message. There are various possible heuristics for selecting which neighbor to block, four of which were described in Section 4.3.3. In this graph the four heuristics are used in conjunction with MC-MGM-1 on the problems from setting 2 and the quality of the final solution is

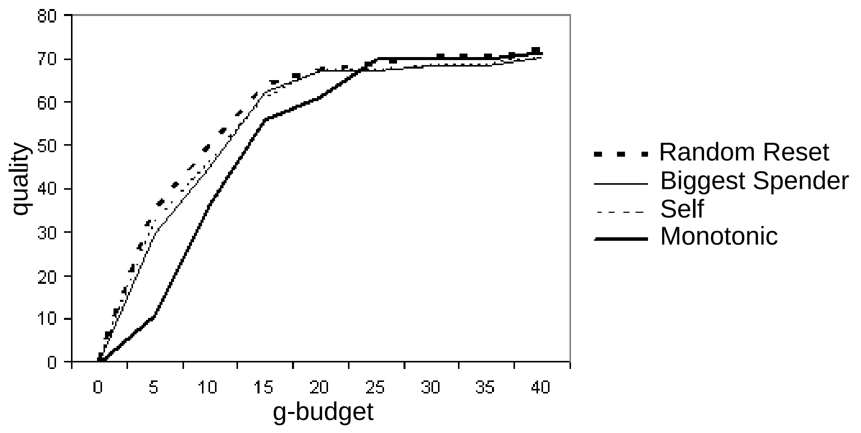


Figure 4.11: quality comparisons for different heuristics in MC-MGM-1

compared. The x-axis again shows the g-budget at each node and the y-axis measures the global reward achieved. Each data point is an average of the results from 100 runs of each of the 15 testcases in setting 2. Note that in contrast to MCA, these results are expressed in terms of reward (not cost) so a higher final quality is better. As can be seen, in these randomly generated cases, there is very little difference between the quality of the final solutions. For all the other graphs in this section, the Random Reset heuristic was used since it equaled or slightly out-performed the other heuristics.

Figure 4.12 shows the difference between the quality of solution obtained using MC-MGM-1 and MC-MGM-2. Once again, problems from setting 2 were used and each data point represents 100 runs of the algorithm on each of the 15 different problem instances. The Best heuristic was used in both algorithms. The x-axis measures the g-budget of each variable and the y-axis shows the average global reward of the final solutions. As can be seen, MC-MGM-2 finds a higher quality solution on average than MC-MGM-1 because it is able to make coordinated moves.

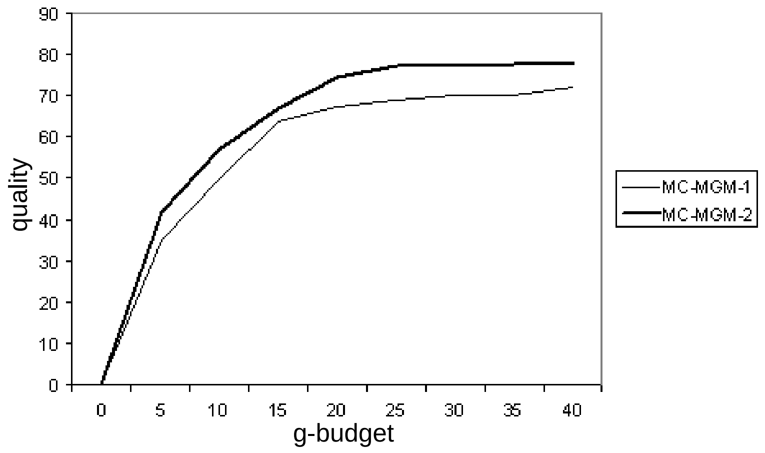


Figure 4.12: quality comparisons for MC-MGM-1 vs. MC-MGM-2

MC-MGM was run on some large-scale problems to demonstrate the scalability of incomplete MC-DCOP algorithms. The problems were randomly generated in a manner similar to the suite of testcases used in Figure 3.9. However, instead of assigning the same g-budget to every single node in a problem, each node was randomly assigned a g-budget between 0 and 40. Each data point represents 100 runs over each of 10 different problems. As can be seen in Figure 4.13, unlike with MCA, MC-MGM's runtime in cycles increases very slowly and thus the algorithm is very scalable. One caveat to note is that while the runtime in cycles remains very flat, the absolute runtime when simulating the agents on a single machine increases because each cycle requires simulating execution of an increasingly large number of agents. However, this issue would not arise in a distributed implementation.

The next set of experiments demonstrate the advantages of the per-node application of privacy in MC-MGM-2. Here, the examples from setting 2 were taken and 0%, 25%, 50%, 75% and 100% of the nodes were randomly assigned to have private g-constraints while

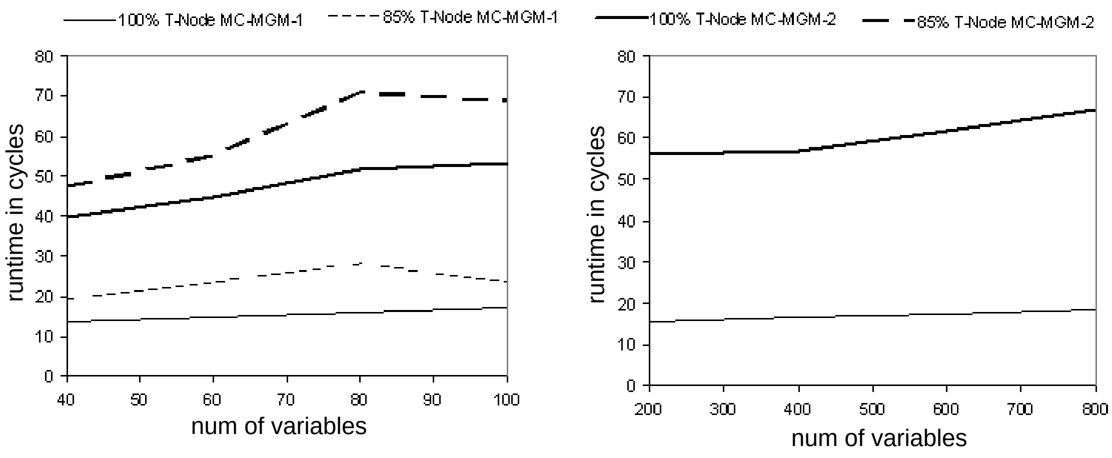


Figure 4.13: runtime vs. size for large scale problems

the remaining were assumed to be non-private. The results are shown in Figure 4.14. The x-axis again shows the g-budget applied and the y-axis measures the runtime in cycles. Each bar in the graph shows an average over 100 runs of the 15 instances. We can see that as the percentage of nodes whose g-constraint is private increases, the runtime increases. Interestingly, the increase in run-time is not uniformly proportional to the increase in the number of private g-constraints; there is a significant jump in run-time when all nodes have private g-constraints for some cases. Thus even when using an incomplete algorithm it is useful to have fine-grained control over privacy.

This chapter has presented the locally optimal MC-MGM algorithms for solving bounded optimization DCOPs. In addition to tailoring the privacy/efficiency settings to the requirements of each variable, users have the choice of a 1-optimal or 2-optimal MC-MGM algorithm. Thus users can tailor the optimality/efficiency tradeoff to suit the requirements of the domain.

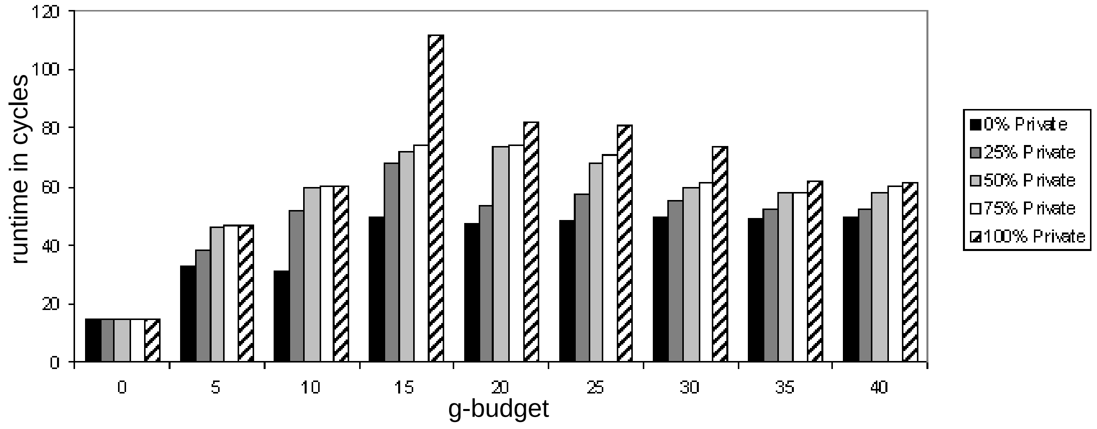


Figure 4.14: Runtime for MC-MGM-2 with varying percentages of private nodes

Chapter 5

Sensitivity Analysis

The last two chapters have focused on the challenges of algorithm design that arise when extending DCOP into bounded optimization domains. This chapter considers the issue of sensitivity analysis. In practical and complex domains, knowing the optimal solution to a problem is not always as useful as knowing whether the outcome would be significantly better, were the constraints of the problem slightly different. This analysis of the problem is known as sensitivity analysis [4]. Sensitivity analysis is a commonly studied problem in constrained optimization areas [4, 3, 27]. In general, the question posed is what effect relaxing a subset of the problem constraints would have on the solution to the problem. The problem is motivated by real-world domains where if the improvement in the overall performance of the team were significant, it might be worth reallocating resources [4]. Sensitivity analysis assumes that additional resources could be procured or reallocated, however, the benefit of doing so must outweigh the cost and inconvenience of acquiring the new resources. The primary challenge in sensitivity analysis is, given the solution to the original problem, to determine the effects of constraint relaxation without recomputing the problem from scratch. The next section (Section 5.1) will discuss how the problem of sensitivity analysis has been interpreted in this thesis and then contrast this with the problem as defined in other fields [4, 3, 27]. Section 5.2 describes the approaches developed for this thesis to tackle sensitivity analysis. Section 5.3 presents experimental results from these approaches.

5.1 Sensitivity Analysis in MC-DCOP

Within the framework of Multiply-Constrained DCOP, the problem of sensitivity analysis can be naturally construed as asking whether a significantly better team solution (higher f-quality or lower f-cost) could be obtained by inserting a small amount of additional resource (g) at one or more nodes. In this case, the constraints being relaxed are a subset of the n-ary resource constraints. In many cases, adding additional units of g will yield only a proportional change in f or no change at all. The goal of sensitivity analysis is to identify disproportionate gains. The notion of disproportionate gain is captured in the concept of *gain/unit*, which is defined as:

- gain / unit = $|F^*_{new} - F^*_{orig}|/R$, where
- F^*_{orig} is the f-cost or f-reward of the globally optimal solution to the original MC-DCOP problem
- F^*_{new} is the f-cost or f-reward of the globally optimal solution to the new MC-DCOP problem, which is identical to the original MC-DCOP problem except that R additional units of g have been distributed to some subset of the g-budgets

The use of the absolute value of the difference between the two solution qualities allows for the same formulation to work regardless of whether the MC-DCOP is formulated as one that minimizes costs or maximizes rewards. F^*_{new} will never be worse than F^*_{orig}, and thus any difference between the two is attributable to an improvement in F^*_{new}. The goal of sensitivity analysis is to identify new MC-DCOPs which introduce no more than R_{max} additional units of g and which have a gain/unit greater than c, where c is a proportional factor, which is set by the user. In the examples in this chapter, c = 1, because the f- and g-functions were randomly generated from the same range of values $\{0, \ldots, 10\}$.

If no MC-DCOPs yielding a gain/unit greater than c can be found, then there is no use investing in further resources since they will not yield a significant improvement in the quality of the overall solution. If more than one such MC-DCOP can be found then there are two ways to select which one to use: highest gain/unit or highest absolute gain. Highest gain/unit makes sense when the cost of adding more resources is proportional to the quantity of resources added, for example, when reallocating money from a different project. Thus, although up to R_{max} resources could be added, the user would prefer to add only as many resources as necessary to get the highest rate of return. The highest absolute gain makes sense when the cost of acquiring R_{max} or fewer additional resources is flat, for example, investing in the next largest size battery. In many cases, particularly when R_{max} is small, these two metrics will yield the same answer.

5.1.1 Challenges of Sensitivity Analysis

The naive approach to sensitivity analysis is to re-run MCA on all of the possible problem variants and compare the solutions. However, this approach would involve running an NP-complete algorithm an additional x times, where:

- $x \geq \sum_{i=1}^{S_{max}} nCi$

- n is the number of variables (potentially a large number in complex domains)

- S_{max} is the maximum size group of nodes that is being considered

The simple approach would be very computationally expensive, doubly-exponential, making the cost of performing the analysis outweigh the benefits of performing it. Furthermore, the introduction of additional resources at one set of variables causes only a local perturbation to the problem. Thus, re-running MCA from scratch on each problem variant would require performing many duplicate computations. There are two main causes of this computational burden. The first is that analysis must be performed on each subgroup of agents that contains up to S_{max} agents. In theory S_{max} could be as large as n, which leads to a combinatorial number of subgroups that must be checked, $O(2^n)$. Another source of computational expense in this naive approach is that the algorithm being run a combinatorial number times (MCA) is NP-complete. Therefore, the main challenge in sensitivity analysis is to reduce both of these sources of computational expense by using heuristics to target high probability sub-groups and by using an efficient mechanism to reoptimize each problem variant. This chapter will primarily focus on the efficient mechanism approach, but in the experimental results section, one promising heuristic will be identified.

5.1.2 Other Approaches to Sensitivity Analysis

The problem of sensitivity analysis, while new in the distributed constrained reasoning field, has been extensively studied in areas such as linear and integer programming

(LP/IP), constraint satisfaction (CSP) and multi-criteria optimization. In linear and integer programming the problem solved is effectively a multiply-constrained non-distributed constraint optimization problem, and there are several algorithms for using the dual of a linear problem to perform sensitivity analysis [4, 19, 8, 22, 59, 51]. This allows for rapid reoptimization of the relaxed problem, but, taking the dual of the problem is not possible in integer programs, when constraint and variable information is distributed, or when multiple constraints are being altered simultaneously. The only sensitivity analysis work currently existing in the area of constraint reasoning is CSP work in the area of constraint relaxation. The work in CSP looks at the problem of reoptimizing after removing constraints entirely rather than making them easier to satisfy [3, 55]. This is a different problem to the one considered in this thesis and since the techniques used there don't consider soft constraints, they would not work for the problem defined in this thesis. Finally, multi-criteria optimization algorithms seek a pareto-optimal solution to problems with multiple constraint functions and sensitivity analysis is generally framed as a question of solution robustness [27, 24]. This is again distinct from the problem being tackled here. Additionally, all of these fields take a centralized approach to the sensitivity analysis problem, rather than a distributed one. The distributed sensitivity analysis problem presents a new challenge because the information needed to estimate the effects of relaxing the resource constraints is distributed among various agents. Thus a primary contribution of this work is to explore an approach to sensitivity analysis in distributed domains.

5.1.3 Semi-cooperativeness

Semi-cooperativeness is the reverse of the sensitivity analysis problem. In general, semi-cooperative work looks at solving multi-agent coordination problems where agents are assumed neither to be fully cooperative, like in DCOP [36, 1, 48, 33], nor fully self-interested, like in game theory [9]. Instead, agents are assumed to have both cooperative and self-interested tendencies [52, 5, 50, 15]. There are many challenges to building semi-cooperative agent algorithms. One particular problem parallels the sensitivity analysis problem discussed in this chapter. If a team of heterogeneous semi-cooperative agents are performing a coordinated task using a framework that assumes cooperativeness, how significant is the effect of the agents' self-interested tendencies? The techniques developed for sensitivity analysis could also be used to estimate the reduction in quality of the global optimal if a particular agent or set of agents holds back some of their local resources (which is equivalent to a problem with less g). This would allow the identification of nodes where honesty and cooperativeness is more crucial. This chapter will phrase the problem in terms of sensitivity analysis, but an equivalent formulation of all the concepts can be made for semi-cooperativeness.

5.2 Approaches to Sensitivity Analysis

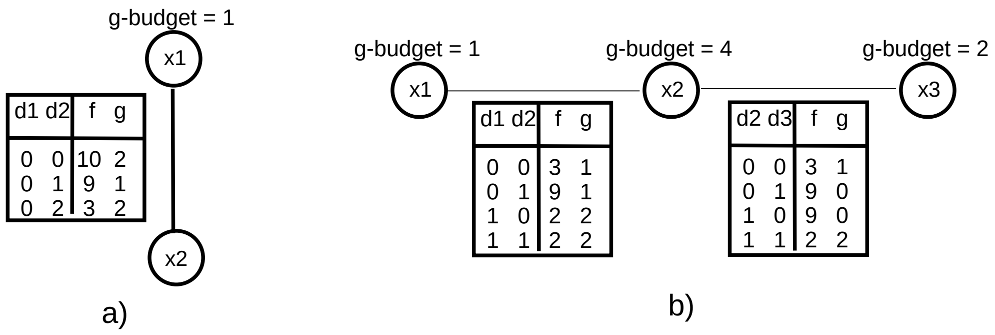
This section will first examine the two ways that disproportionate gains can occur after a fixed amount of additional resource has been added to one or more variables: single-link gain and chain reaction. Then the algorithms developed to perform distributed sensitivity analysis will be described. The first algorithm, link analysis, is a computationally inexpensive approach to identifying single-link gains. The second set of algorithms use the MC-MGM algorithms to identify both single-link and chain reaction gains.

5.2.1 Gain Types

To simplify the discussion, it is worth first looking at how the addition of extra g at a single node could affect the overall solution quality. There are two basic ways that a fixed amount of extra g inserted at a single node could yield a disproportionate improvement in f. Examples of the two types of gain are shown in Figure 5.1. One occurs when the g vs. f function on an individual link has a slope greater than c, where c is the proportional gain that would typically accompany the insertion of one additional unit of g. If one plots the local f vs. g for a link and eliminates all dominated value-pairs (those with both high g-cost and high f-cost/low f-reward), the slope can be: less than c, equal to c, or greater than c. The local slope is the slope in the interval which is within a small distance (R_{max}) of the current assignment. In the example in Figure 5.1a, the addition of one unit of g at variable x_1 would allow the variable to reduce the f-cost on that link from 9 to 3. This yields a disproportionate improvement in the team solution just by looking at one link. Single-link gains are difficult to extend to sub-groups of variables larger than those connected to a single link because only an individual link has a g vs. f function for which the slope can be taken. Multiple links can be analyzed independently.

The second source of disproportionate gains comes from chains of proportional gains. In this situation, no individual link yields a gain greater than c, but the change in values caused by the introduction of the extra g causes a knock on set of value changes that each yield additional improvement in the team utility. Figure 5.1b shows a chain reaction.

In this case, the optimal solution to the original problem is $\{0 \leftarrow x_1, 0 \leftarrow x_2, 0 \leftarrow x_3\}$. Inserting one additional unit of g at variable x_1 would allow x_1 to change value from 0 to 1 and reduce the f-cost on the $x_1 - x_2$ link by 1. This value change allows x_2 and x_3 to make a coordinated value change from $\{0 \leftarrow x_2, 0 \leftarrow x_3\}$ to $\{1 \leftarrow x_2, 1 \leftarrow x_3\}$. This value change yields an additional 1 unit of savings in f-cost on the $x_2 - x_3$ link. Thus while each link only yields a proportionate saving in f (1 unit), together they yield a disproportionate saving (2 units of f for 1 unit of g). The interaction of adding g at multiple variables would be similar to that of a chain reaction from a single variable, in both cases multiple local reoptimizations need to be performed. Therefore, the methods used to identify chain reactions could also be used to solve the sensitivity analysis problems that involve inserting g at multiple variables.



d1	d2	f	g
0	0	10	2
0	1	9	1
0	2	3	2

d1	d2	f	g
0	0	3	1
0	1	9	1
1	0	2	2
1	1	2	2

d2	d3	f	g
0	0	3	1
0	1	9	0
1	0	9	0
1	1	2	2

Figure 5.1: a) Single link gain b) Chain reaction gain

5.2.2 Link Analysis

Identifying link functions where the local slope is greater than c would be one efficient way to do sensitivity analysis. That is because when the local slope is greater than c,

```
Link Analysis ()
(1)  $maxgain = c$
(2)  **for all** links $x_i - x_j$
(3)    ComputeGFmap($x_i - x_j$)
(4)    **for** $i = g_{cur} \ldots g_{cur} + R_{max}$
(5)      $s$ = FindSlope($g_{cur}$, $i$)
(6)      **if** $s > maxgain$ and NoViolations()
(7)        $maxgain = s$
```

Figure 5.2: Link Analysis Pseudo-code

adding additional units of g will yield a disproportionate single-link gain. This method of performing sensitivity analysis is referred to as *link analysis.*

The pseudo-code for the link analysis algorithm is shown in Figure 5.2. Link analysis involves examining the g vs. f function at every single link by constructing a GFmap like the one used in MCASA (lines 2-3) and seeing whether the local slope is greater than c (lines 4-6). It can only consider changes in assignment that will not create new g-constraint violations for any of the neighboring nodes (line 6). It can thus identify single link gains like that in Figure 5.1a, where the GFmap is:

g	f
1	9
2	3

This GFmap gives a local slope of 6 without introducing any new g-constraint violations, so it would be identified by link analysis as a problem that merited the insertion of more resources. Link analysis is difficult to extend to problems where S_{max} was greater than the number of variables connected to a single link because only an individual link has a g vs. f function for which the slope can be taken.

```
Local Reoptimization ()
(1)   $maxgain = c$
(2)   for all subgroups $S$ smaller than $S_{max}$
(3)      $R = R_{max}$
(4)      while $R \geq 1$ and $gain > 0$
(5)         for all distributions of $R$
(6)            run MC-MGM on new problem
(7)            $gain$ = computeGain()
(8)            if $gain > maxgain$
(9)                $maxgain = gain$
(10)        decrement $R$
(11)  ReportMaxGain()
```

Figure 5.3: MC-MGM Based Local Reoptimization Pseudo-code

5.2.3 Local Reoptimization

The other method of performing sensitivity analysis, local reoptimization, involves using the MC-MGM algorithms. Since the additional resource has been inserted at a localized set of nodes, its effects percolate out from a single area through many local interactions. Incomplete algorithms like MC-MGM-1 and MC-MGM-2 are a natural choice for getting an estimate of the ripple effects of adding more g to a set of variables because their entire optimization mechanism revolves around optimizing and re-optimizing the local value assignments. This makes them able to identify both single-link and chain reaction gains as well as handle problems with larger values of S_{max}. One reason for using incomplete algorithms rather than complete ones is that complete algorithms may find an optimal rapidly, but must still expend time systematically proving that the optimal has been reached [30].

The local reoptimization approach to sensitivity analysis involves running MC-MGM on variants of the original MC-DCOP problem where subgroups of nodes have been given slightly more g. The pseudo-code for the local reoptimization algorithm is shown in

Figure 5.3. MC-MGM-1, MC-MGM-2 or any future MC-MGM-k algorithms could be used in this approach. One thing to note is that this approach starts by inserting R_{max} units of extra resource and then decrements from there. If the gain for inserting R units of additional resource into a particular subgroup is 0, then the algorithm abandons the attempt to insert $R-1, R-2, \ldots, 1$ units of g into that particular sub-group because they will also yield a gain of 0 (line 4). Additionally, whether the absolute gain or gain/unit is used as the selection criteria is left up to the user and computeGain() will report whichever metric the user has selected (line 7).

In order to speed up computation and find an answer close to the global optimal, the MC-MGM algorithms take the old optimal solution (which is a satisfying solution in the new problem) as their initial assignment. The reason for using the old global optimal is two-fold: speed and quality. In terms of speed, every additional cycle of execution of the MC-MGM algorithms means $O(2^n)$ more cycles over all to perform sensitivity analysis. Using the old global optimal instead of starting from scratch yields a significant savings in runtime. In terms of quality, the old global optimal is the n-optimal solution to the original problem, so it is very high quality. There is no guarantee that the 1- and 2-optimal MC-MGM algorithms would find their way to such a high quality solution without being seeded. However, since they are seeded with an assignment of this quality, they will try no assignments of lower quality when performing sensitivity analysis. This is because variables only switch assignments when doing so will yield an improvement in the quality of the solution. One reason that using the MC-MGM algorithms is much more efficient than using MCA for sensitivity analysis is because they are locally optimal. Even if seeded with the old optimal solution MCA would have to try many other assignments in order to prove that it had reached the global optimal because its previous estimates of lower and upper bounds would no longer be valid. This would involve doing many redundant computations for assignments which were already determined to be suboptimal. In contrast, MC-MGM can start from the old optimal and opportunistically use the extra resources to explore only those assignments that yield an improvement over the existing assignment.

5.3 Experimental and Analytical Results

The randomly generated experiments from settings 3 and 4 from Chapter 3 were used to run sensitivity analysis. The effect on the solution quality of the addition of up to 5 extra units of g at each individual node was examined ($S_{max} = 1$, $R_{max} = 5$). Among the testcases available, the ones with an initial g-budget of 15 were selected because this was the g-budget at which the greatest tradeoff of f and g was being made and thus at which the question of sensitivity analysis would be most relevant. The 3 different sensitivity analysis algorithms were run on 50 different problem variants (10 variables, 5 possible units of extra resource) for each of the 15 testcases in each of the settings. For each of the cases, the algorithms attempted to identify which of the 50 problem variants had a gain/unit greater than 1 (which was the proportional factor, c). If there was more than one such variant, the algorithms identified which variant had the highest gain/unit and which had the highest absolute gain. This amounted to identifying which (if any) variable in a particular case should be given more resources and how many resources (up to 5 units) should be given to the identified bottleneck variable. Since link analysis is deterministic, it was run just once on each variant for each case. For the MC-MGM algorithms, their results were averaged over 100 runs on each of the variants.

5.3.1 Effectiveness

The first surprise was that link analysis failed to work experimentally because it was only able to consider alterations to the values of the two variables (x_i and x_j) on the link. It could not alter the values of any of x_i or x_j's neighbors. Link analysis was too constrained by the neighbors' choices of values and found no opportunities for disproportionate gain. While link analysis might still work on sparser graphs than those used in these experiments, its runtime savings would be diminished by the fact that incomplete DCOP algorithms run rapidly on sparse graphs [44].

The results for local reoptimization using the MC-MGM algorithms are shown in Figure 5.4. MC-MGM-1 identified 7 cases (out of the 15) in setting 3 and 6 cases in setting 4 where disproportionate gains could be achieved by inserting up to 5 units of g at one node in the problem. However, by only being allowed to change one value at a time, MC-MGM-1 was still constrained as to its use of the extra g. MC-MGM-2 found 7 cases in both settings where disproportionate gains could be found. Additionally, in two of the cases (case 14 for setting 3 and case 1 for setting 4) where both MC-MGM-1 and MC-MGM-2 identified the opportunity for disproportionate gains, MC-MGM-2 found a use for the extra g that yielded a greater gain even when inserted at the same variable. Experiments later in this section demonstrate that, MC-MGM-2 takes a greater number of cycles to execute on these problems than MC-MGM-1 (6.7 cycles vs. 3.2 cycles). Since these performance differences will be magnified by the combinatorial number of runs of these algorithms, there is a quality vs. efficiency tradeoff to be made in picking an MC-MGM algorithm to do sensitivity analysis.

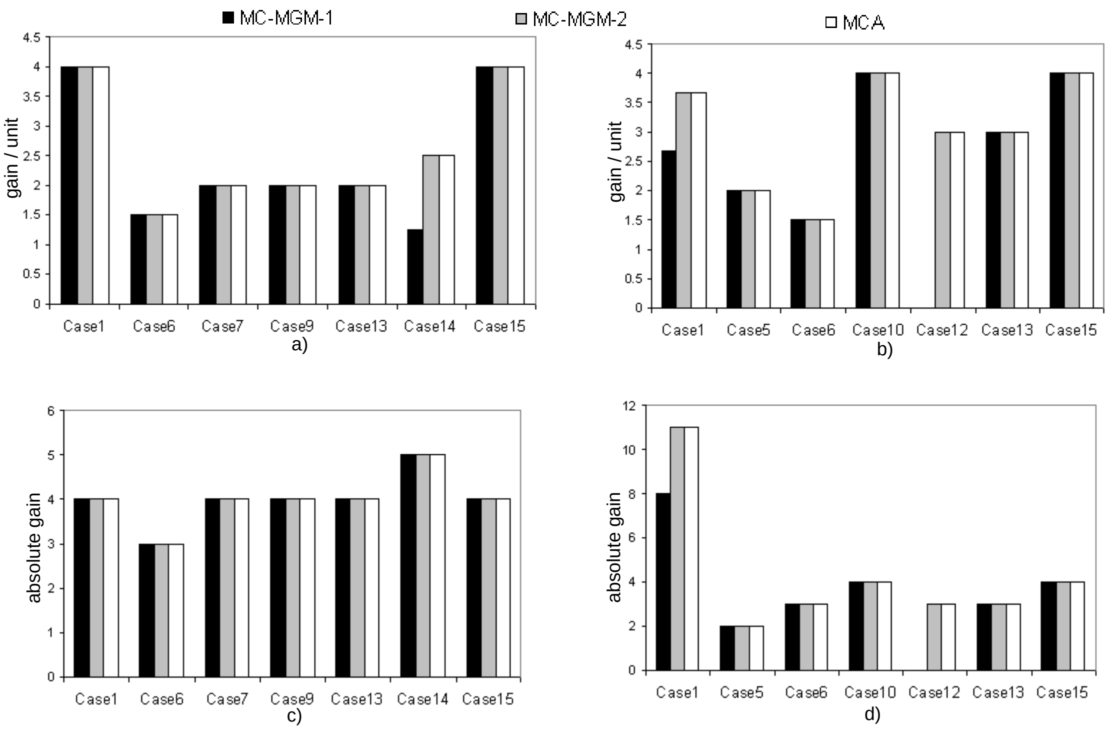


Figure 5.4: Sensitivity analysis a) 100% T-Node cases using gain/unit b) 85% T-Node cases using gain/unit c) 100% T-Node cases using absolute gain b) 85% T-Node cases using absolute gain

To get an indication of how well MC-MGM-2 was performing the local reoptimization, MCA was run on the problem variant that MC-MGM-2 indicated had the most disproportionate gain for each of the 14 cases MC-MGM-2 identified as being able to be improved by the insertion of additional resources. As can be seen, MC-MGM-2 managed to reoptimize to the new global optimal. To see whether any of the 16 cases MC-MGM-2 flagged as negative were false negatives, MCA was run on the problem variants for each case that involved inserting the full 5 units of additional g at each variable. A negative result when inserting 5 units of extra g necessarily implies that inserting 4, 3, 2 or 1 units would also yield no improvement in the quality of the solution. The experiments confirmed that MC-MGM-2 flagged no false negatives.

5.3.2 Effects of Link Density

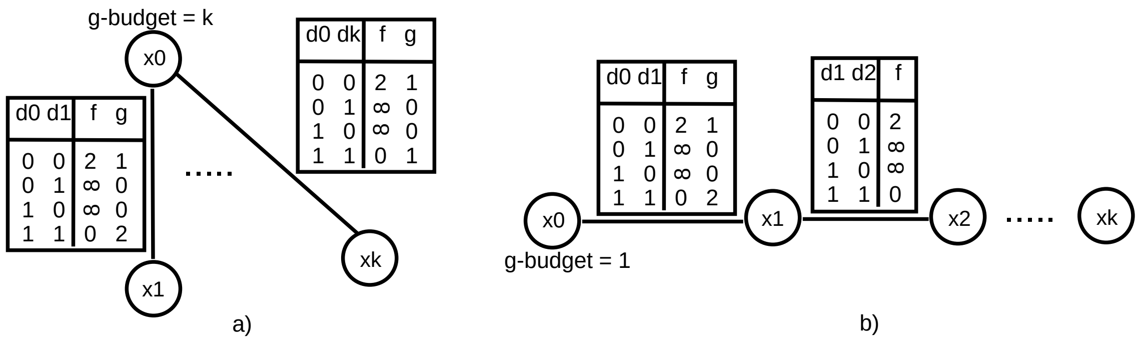


Figure 5.5: Example of MC-MGM-k failure a) star topology b) chain

The experimental results in Figure 5.4 demonstrated that MC-MGM-2 finds the globally optimal solution to the problem variants in the test suite developed for this thesis. However, no MC-MGM-k algorithm can be guaranteed to reach the global optimal unless k = n, where n is the number of variables in the problem. Two examples are shown in Figure 5.5 to demonstrate that for any value of k, MC-MGM-k is unable to guarantee reaching the new global optimal for problems with k+1 variables. In the example in Figure 5.5a, there are assumed to be k+1 variables $\{x_0, \ldots, x_k\}$ and k links in a star topology. The f- and g-cost functions for all of the links that are not shown is assumed to be the same as the one shown for the link between x_0 and x_k. The constraint between x_0 and x_1 has a slightly different link function. Variable x_0 has a g-constraint and the initial available g-budget is k. Thus, the global optimal to the original problem is the assignment $\{x_0 \leftarrow 0 \ldots x_k \leftarrow 0\}$, which causes 1 unit of g to be consumed on each link. If sensitivity analysis is examining the problem of inserting 1 additional unit of g at variable x_0, any locally-optimal algorithm which does not allow all k+1 variables to change values simultaneously will be unable to move to the new global optimal of $\{x_0 \leftarrow 1 \ldots x_k \leftarrow 1\}$. The reason it won't reach the new global optimal is that any move of k or fewer agents will leave at least one link with the variables on each end having different values. Since this accrues an infinite cost, the move will not be profitable and the agents won't change values. In the example in Figure 5.5b, there are again k+1 variables and k links, this time in a chain topology. The f-cost functions for all of the links not shown is assumed to be the same as the one shown for the link between x_1 and x_2. Variable x_0 has an initial available g-budget of 1, which makes the globally optimal assignment $\{x_0 \leftarrow 0 \ldots x_k \leftarrow 0\}$. If inserting 1 extra unit of g at variable x_0, it will once again be necessary for all k+1 variables to change values simultaneously in order to reach the new global optimal of $\{x_0 \leftarrow 1 \ldots x_k \leftarrow 1\}$. Any move of k or fewer agents will again leave at least one link with the variables on each end having different values. Since this accrues an infinite cost, the move will not be profitable and the agents won't change values.

In the centralized CSP literature, graph structures can be categorized based on their structural properties (k-trees) to determine the complexity of k-consistency algorithms that need to be applied to make the CSP graph consistent [16, 28]. While k-consistency differs from k-optimality [**?**], there was an expectation, nonetheless, that it would be possible to identify simple tree structures to determine which MC-MGM-k algorithms would guarantee reaching the global optimal. However, the examples given in Figure 5.5 use a star and a chain topology, which both have the minimum average link density.

The chain topology also has the lowest maximum link density. The fact that these examples have a minimum link density demonstrates that choosing the correct MC-MGM-k algorithm to run on a particular problem instance is not a simple matter of looking at the link density.

Another result to emerge from the experiments in Figure 5.4 was that the most highly connected nodes were also the nodes most frequently identified as bottlenecks. The first step in identifying this pattern was to look at the number of times that a particular variable, x_i, was identified as the bottleneck node as well as the number of links connected to x_i. The results were then averaged over all the nodes with a particular link density to show how many times on average a variable with a particular link density was identified as the bottleneck. If there were no variables with a particular link density, a result of N/A is indicated. As can be seen in Table 5.1, the greater the number of links a node had, the greater the average number of times it was identified as a bottleneck for both 100% and 85% T-node cases. This demonstrates that, for problems like those developed for this thesis, the problem of sensitivity analysis could be solved even more efficiently by focusing the local reoptimization just on those nodes or groups of nodes with high link density. This means that instead of running MC-MGM-1 or MC-MGM-2 on problem variants involving all of the $O(2^{S_{max}})$ possible subgroups of size less than or equal to k, only the most likely variants need to be run.

The heuristic identified in Table 5.1 does not contradict the demonstration in Figure 5.5 that link density is not a useful metric in choosing the best MC-MGM-k algorithm to run on a particular problem. Incomplete algorithms cannot be guaranteed to reach the global optimal for problems with more variables than can be moved simultaneously. The

Links	**100% T-Node Average Times Identified**	**85% T-Node Average Times Identified**
1	0	0
2	1	0.5
3	1	N/A
4	4	1.5
5	N/A	2

Table 5.1: Properties of Bottleneck Nodes

examples in Figure 5.5 show that this result holds even if the assumption is made that the variables are in a minimally connected topology. In contrast, the results in Table 5.1 indicate that when the new global optimal is found for all of the sensitivity analysis problem variants, it is likely that the variant that involved inserting additional resources at the most highly connected nodes, will be the one with the highest quality. This result is a feature of the problems whereas the rather than a property of the incomplete algorithms. It makes no assumptions about how the new global optimal is reached.

5.3.3 Justification for Reoptimizing

Finally, experimental results provide justification for starting from the old global optimal instead of beginning from scratch. The average runtime for a single run of MC-MGM-1 on the original problem was 11.4 cycles (as shown in Figure 4.10) and the average runtime for MC-MGM-2 was 34.9 cycles. These averages were over 100 runs of the setting 3 and setting 4 testcases with a g-budget of 15. In contrast, the average runtime for a single run of MCA on the original problem was 587 cycles (as shown in Figure **??**), which is significantly slower than MC-MGM-1 and MC-MGM-2. The average runtime for a single run of MC-MGM-1 to reoptimize a single problem variant was 3.2 cycles and for MC-MGM-2 it was 6.7 cycles. These averages were calculated over 100 runs of the 50 variants of the setting 3 and setting 4 testcases with a g-budget of 15. Thus reoptimizing rather than starting from scratch saved an average of 8.2 cycles per variant when running MC-MGM-1 and 28.2 cycles per variant when running MC-MGM-2. Even restricting the maximum group size (S_{max}) to be one, there were 50 problem variants for each of the testcases. Thus on each testcase, reoptimizing rather than starting afresh saved an average of 410 cycles for MC-MGM-1 and 1410 cycles for MC-MGM-2. For larger values of S_{max}, the runtime savings per case would be even larger. The results are summarized in Table 5.2.

	MC-MGM-1	**MC-MGM-2**
Runtime from Scratch	11.4	34.9
Runtime to Reoptimize	3.2	6.7
Saving per Run	8.2	28.2
Saving per Case	410	1410

Table 5.2: Runtime Savings when Reoptimizing

This chapter has demonstrated how to use the incomplete MC-MGM algorithms in tandem with MCA to perform sensitivity analysis on the bounded optimization DCOP problems. MC-MGM-1 is slightly more efficient but misses some opportunities for improvement in the problem that MC-MGM-2 identifies. In addition, for randomly generated examples, the search for bottleneck nodes can be targeted at those groups of variables that are the most highly connected.

Chapter 6

Related Work

This chapter examines four main areas of work that is related to the research in this thesis: (i) efficiency work in Distributed Constraint Optimization (DCOP) algorithms; (ii) advances in multi-objective tradeoffs within DCOPs; (iii) research on multi-criteria collaboration beyond DCOPs; and (iv) work on sensitivity analysis.

6.1 Efficiency in DCOPs

There is significant continued progress in singly-constrained complete and incomplete DCOP algorithms [36, 60, 63, 1, 48]. There are three leading complete algorithms, which all use slightly different mechanisms to solve the DCOP problem: Adopt, OptAPO and DPOP.

- *Adopt*[36]: This thesis already provided detailed background information on Adopt and the complete multiply-constrained DCOP algorithms developed as part of this thesis built upon the Adopt algorithm. Other work has continued to build upon and improve Adopt. For example, [30] proposed node ordering heuristics, to construct shallower trees in Adopt, which allowed for greater efficiency. Ali et al [1] presented preprocessing heuristics that propagate lower bounds on f-costs to variables in the Adopt DFS tree and allow for quicker convergence on the optimal solution. These heuristics led to an order of magnitude improvement in Adopt's performance. Additionally, Davin and Modi [7] provide techniques to efficiently handle multiple variables per agent in Adopt since previous work had tended to focus on problems with a single variable per agent. Finally, Pecora et al [47] looked at creating a version of Adopt that could handle n-ary DCOP constraints in addition to the binary constraints it was initially designed to work with.

- *OptAPO*[33]: OptAPO emerged as an early competitor to Adopt and makes use of a partial centralization mechanism. Adopt uses very little centralization in its execution, the main exception being the propagation of f-costs to the root node. In OptAPO, partial centralization allows an individual variable to manage a group of other nodes by deciding their assignments; in the extreme case, a single node may manage assignments for all other nodes, leading to full centralization. There has been much debate about how to compare the efficiency of Adopt and OptAPO[33, 6] with different performance metrics giving an edge to one algorithm over the other. Ultimately, the relative costs of computation and communication in the domain appear to govern the choice of an appropriate algorithm.

- *DPOP*[48]: DPOP is the latest entry into the field of complete DCOP algorithms. The key difference in DPOP is that it is a variable elimination algorithm, where individual variables communicate all their information to their neighbor in one-shot. This contrasts with the repeated rounds of value and cost message exchange in Adopt. The result is that DPOP leads to a significantly smaller number of messages exchanged among agents; unfortunately, the size of each message grows exponentially in the link density of a variable and the domain size. DPOP's exponential memory requirements are problematic in some domains, but its low levels of communication make it an option for domains where the more communication intensive algorithms are infeasible. Recent research in DPOP has attempted to reduce the memory requirements of the algorithm [49].

The work in this thesis is complementary to these advances. The multiply-constrained DCOP formulation presents a new challenge for all of these algorithms. Some of the techniques developed here would transfer to these other algorithms, e.g. MCAS and MCASA style techniques could be applied to algorithms like OptAPO. Whether the DPOP algorithm [48] will similarly benefit from the techniques introduced here is a challenge for future work. Algorithms such as DPOP may face challenges when addressing multiply-constrained DCOPs because the variable elimination algorithms rely on there being a single best response to any combinations of a variable's ancestors' values. However, in MC-DCOP, there are multiple best responses to a particular combination of ancestors' values, one for each way the sets of g-budgets could be split. This would lead to a massive increase in the amount of information a variable would need to send its parent and thus to an explosion in the space requirements of the parent. Since space requirements are the main limiting factor on how well DPOP can scale, this would exacerbate an existing problem.

In addition to the complete DCOP algorithms that guarantee optimality, there have been significant advances in incomplete algorithms [45, 58, 64, 61]. These algorithms tradeoff optimality (settling for a local optimality) to gain in efficiency. There are several leading incomplete DCOP algorithms: MGM, DSA, and LA-DCOP.

- *MGM*[45, 44, 14]: The MGM algorithms (MGM-1 and MGM-2) developed by Pearce et al were extensively described earlier since they provide the basis for the new MC-MGM algorithms described earlier. Recent research on the algorithms has established that the algorithms are k-optimal and that two types of guarantees can be made about the quality of the solution reached by these algorithms [44]. The first type of guarantee fixes an upper bound on the number of k-optima that can occur in a problem. The second type of guarantee establishes a lower bound on the quality of the k-optimum as a percentage of the quality the globally optimal solution. These results allow users to estimate the benefits of using a particular level of k-optimum on a particular problem because the efficiency/performance tradeoff has been rigorously quantified.

- *DSA*[45, 44, 14]: The Distributed Stochastic Algorithm (DSA) is similar to the MGM algorithms but introduces a stochastic element to the algorithm [14, 44]. Instead of an agent simply changing values when it has the highest local gain, an agent decides randomly whether to move or not. This reduces the messaging load of the algorithm and in some cases allows the agents to reach a higher quality k-optimum than MGM, while in others, MGM outperforms DSA.

- *LA-DCOP*[41]: The Low-Communication Approximate DCOP (LA-DCOP) algorithm was developed to meet the challenges of role-allocation DCOP domains requiring dynamism, low communication and rapid execution. LA-DCOP employs a token-based approach to DCOP. Agents pass tokens and can only take on a particular role if they have the corresponding token in their possession. Local probabilistic information is used to determine whether to accept a token or pass it along. Agents self-select into teams by choosing tokens that represent the same joint-task without explicitly coordinating their choice. LA-DCOP has been shown to scale to very large-scale problems.

My work is complementary to these advances. Multiply-constrained DCOP is a new challenge that cannot yet be tackled by any of these algorithms. Some of the techniques developed here would transfer to these other algorithms. Given that DSA and MGM are very similar, the techniques used to build MC-MGM would transfer fairly easily to DSA. LA-DCOP's token based approach could be adapted to meet the challenges of MC-DCOP domains. LA-DCOP already has a notion of resource constraints and local capabilities due to its having originally been developed in the context of role-allocation domains, making it straightforward to introduce MC-DCOP's local constraints.

6.2 Multiple Objectives within DCOPs

While this thesis focuses on multiple cost objectives (i.e. f-costs and g-costs), complete DCOP algorithms already engage in a complex tradeoff between privacy, efficiency, diversity and optimality. Error-bounds in Adopt [36] allow it to trade the optimality of a solution for an improvement in efficiency. Maheswaran et al [31] and Greenstadt et al [21, 20] studied the privacy loss in different complete DCOP algorithms, thus providing a better understanding of the privacy-efficiency tradeoffs in these algorithms. Traditionally, DCOP algorithms tolerate some loss in privacy for the sake of efficiency. However, Silaghi [54] coined the term 'semi-cooperative algorithms' to describe DCOP algorithms that required agents to attain global optimality while maintaining privacy. Yokoo et al[62] looked at how to use multiple external server agents to find the global optimal while guaranteeing significant privacy. However, this guarantee came at the cost of efficiency.

The balancing of multiple objectives within regular DCOP algorithms also comes to light in the work on incomplete algorithms. Incomplete algorithms[45, 58, 64, 61, 41] trade off optimality to gain efficiency. Pearce et al.'s k-optimality [44] presented a novel consideration of multiple objectives within DCOPs: solution quality and solution diversity. Thus, multiple solutions are chosen not only due to their low cost but also their diversity, i.e. how far away the solutions are from each other. Thus, Pearce et al's technique focuses on choosing multiple DCOP solutions that are each locally optimal, and are guaranteed to be each a distance of k agent's value assignments away from other solutions.

There is, therefore, a vast space of tradeoffs in DCOP algorithms, and the Multiply-Constrained DCOP algorithms in this thesis explore one area within this space. Indeed, the algorithms in this thesis add another dimension to this tradeoff space, that of resource constraints. Furthermore, this thesis has explored the privacy-efficiency tradeoff as it occurs in MC-DCOP domains by examining the effects of having fine-grained control over the privacy of resource constraints. This extends beyond the whole algorithm approaches that have been used in studying the privacy-efficiency tradeoff in regular DCOP algorithms [20, 31, 21].

This thesis briefly touched on another area where two objectives require balancing: semi-cooperativeness. Semi-cooperativeness involves trading off team objectives for self-interest. In general, multi-agent coordination work has fallen into two categories, those where agents are assumed to be fully cooperative, like in DCOP [36, 1, 48, 33], or these where agents are fully self-interested, like in game theory [9]. Recent research in economics has suggested that people have both cooperative and self-interested tendencies [52, 5, 50, 15] and therefore may require agents that can balance these two objectives. Several distributed constraint reasoning algorithms have been built that don't assume either fully cooperative or fully self-interested agents [12, 65, 13]. In this thesis one particular problem related to semi-cooperativeness was examined, that of identifying agents whose critical position in the team makes honesty and cooperativeness more critical. This is complementary to the work on semi-cooperative algorithms.

6.3 Multi-criteria collaboration in general

Also related to the multiply-constrained DCOP work in this thesis is research into multi-criteria collaboration [38, 11, 39, 23, 34] which looks at finding a pareto-optimal for problems containing multiple objectives rather than optimizing a single objective subject to resource constraints. While that work has focused on applications such as distributed planning, it did not benefit from recent research that formalized DCOP and developed efficient algorithms for it. The algorithms in this thesis built on these efficient DCOP algorithms. Approaches to multi-criteria constraint satisfaction and optimization problems have tackled the problem using centralized methods [17, 25], but a central contribution of this thesis is in tackling the distributed problem, which requires designing algorithms where agents function without global knowledge [25].

Research on distributed Markov Decision Processes (MDPs) has also focused on multiple objectives: attempting to maximize total expected reward of the distributed MDPs while balancing a second objective [42, 2]. This work takes as its multi-objective tradeoff that of balancing optimality with security, which is taken to mean randomization. When randomized policies are obtained in distributed MDP settings, the key challenge that arises is one of coordinating multiple agents without the benefit of unbounded communication. Paruchuri et al [42, 43] present efficient linear programming techniques to address this problem. However, this work focuses on centralized planning for distributed execution whereas the algorithms in this thesis focus on distributed planning for distributed execution.

6.4 Sensitivity Analysis

The problem of sensitivity analysis, while new in the distributed constrained reasoning field, has been extensively studied in areas such as linear and integer programming (LP/IP), constraint satisfaction (CSP) and multi-criteria optimization.

- *Linear and Integer Programming*[4, 19, 8, 22, 59, 51]: Linear programming (LP) techniques such as branch and bound and the simplex algorithm can be used to solve resource-constrained optimization problems almost identical to the bounded optimization DCOP problems tackled in this thesis. The primary differentiations are that DCOP is distributed and uses discrete domains for variables whereas linear programming assumes continuous domains. The problem of sensitivity analysis has been well-studied in linear programming and it is possible to rapidly reoptimize a solution after one of the resource constraints has been relaxed by using the dual of the problem. The technique cannot handle relaxing multiple constraints simultaneously. There are various theoretical results on the effectiveness of the re-optimization [4, 19, 8, 22]. Integer programming (IP) algorithms allow for solving problems with discrete domains similar to those represented in a non-distributed MC-DCOP. While the problem of sensitivity analysis has been less extensively studied in IP, there are still several algorithms for performing sensitivity analysis, but they involve resolving the problem to a far greater extent [59, 51, 8].

- *Constraint Satisfaction*[3, 55]: CSP work in the area of constraint relaxation is a form of sensitivity analysis. In constraint relaxation the relaxation of the problem is taken to mean removing constraints rather than making them easier to satisfy.

CSP work also involves hard constraints and does not have the soft optimization constraints represented in MC-DCOP [3, 55].

- *Multi-Criteria Optimization*[27, 24]: As described in section 6.3, multi-criteria optimization looks at problems with multiple objective functions. However, multi-criteria optimization algorithms seek a pareto-optimal solution to problems with multiple constraint functions rather than seeking to optimize one function while satisfying another. In this field, sensitivity analysis is generally framed as a question of solution robustness. In this case, the problem is not to look for simple modifications that would improve the solution, but to find solutions that are still optimal even if the problem varies slightly [27, 24].

All of these areas take a centralized approach to the sensitivity analysis problem, rather than a distributed one which is required for distributed MC-DCOP domains. The distributed sensitivity analysis problem presents its own challenges and opportunities. It presents a new challenge because information needed to estimate the effects of relaxing the resource constraints is distributed among various agents, thus taking the dual of the problem is not possible. However, this distribution also presents an opportunity for parallelization not present for centralized approaches. In addition, the work previously done in the areas of CSP and multi-criteria optimization have tended to frame the goal of sensitivity analysis slightly differently to the way it is defined in this thesis.

Chapter 7

Conclusion

7.1 Summary

This thesis focused on extending the expressivity of DCOP to capture more complex domains, in particular distributed bounded optimization domains. These domains require that networks of agents not only optimize a global objective function as in DCOP, but also satisfy local resource constraints. One example of a bounded optimization domain is distributed meeting scheduling. In distributed meeting scheduling domains the global goal of maximizing everyone's satisfaction with the schedule of meetings has to be balanced with the limited local travel budgets. Existing DCOP algorithms are unable to solve bounded optimization problems because they are unable to handle local, possibly private resource constraints in addition to the team goal. This thesis defined the MC-DCOP framework, which is an extension to DCOP, to capture these problems. It also made two primary contributions toward extending DCOP to bounded optimization domains: (i) designing new complete and incomplete algorithms to solve bounded optimization

DCOPs and (ii) exploring ways to perform distributed sensitivity analysis on bounded optimization problem instances.

The algorithm design challenge required creating distributed MC-DCOP algorithms to solve bounded optimization problems efficiently. The goal in designing the algorithms was to prevent a significant increase in runtime complexity by using mutually-intervening search. The idea behind mutually-intervening search was to use the information inherent in the resource constraints to preemptively prune the search space for the global goal. In order to obtain the maximum benefit from mutually-intervening search, it was useful to have as many agents as possible make their resource constraint information available. Thus, giving fine-grained control over the privacy of resource constraints was important. This meant that the introduction of a single private constraint into the problem did not necessitate using a completely private algorithm and the information available from the non-private constraints could still still be exploited to increase efficiency. The other tradeoff to be considered in designing the new algorithms was scalability vs. optimality. Some bounded optimization problems require complete algorithms which can find the most effective use of scarce resources, while others require incomplete algorithms which can rapidly solve large-scale problems at the cost of finding suboptimal solutions.

One of the main contributions of this thesis was to present both complete and incomplete multiply-constrained DCOP algorithms that employed mutually-intervening search. Three different algorithms were developed: one complete algorithm and two incomplete algorithms. The complete algorithm, Multiply-Constrained Adopt (MCA), found the globally-optimal solution to bounded optimization problems. The two incomplete algorithms differed in the locality of the optimal that they were designed to reach. Multiply-Constrained Maximum Gain Message-1 (MC-MGM-1) was a 1-optimal algorithm which only considered moves that were profitable for an individual agent. MC-MGM-2 allowed pairs of agents to make coordinated moves, which allowed it to break out of some of the lower quality local optima that MC-MGM-1 could not escape. The three algorithms allowed for trading off optimality and scalability. Additionally, all three algorithms allowed for fine-grained control over the privacy vs. efficiency tradeoff. There were four main algorithmic innovations in designing the multiply-constrained DCOP algorithms: (i) transforming the network to allow private n-ary constraints to be enforced; (ii) assigning upper-bounds on resource consumption to neighbors, in order to gain efficiency; (iii) identifying a structural property of the graph – T-nodes – which allowed agents to calculate exact bounds on resource consumption; (iv) using a virtual value assignment to identify when resource constraints have rendered a problem unsatisfiable. Proofs of correctness were provided for both sets of algorithms.

Experiments were run to demonstrate the efficacy of the MC-DCOP algorithms at solving bounded optimization problems as well as to explore the tradeoffs that were built into the algorithms: privacy vs. efficiency and optimality vs. scalability. Experimental results confirmed the usefulness of fine-grained control over the privacy/efficiency tradeoff. Designating fewer than 25% of the agents' resource constraints as private had a negligible effect on the runtime. However, as the percentage of resource constraints designated as private increased, the effect on the runtime became more pronounced. When all of the resource constraints were kept private, the runtime was an order of magnitude greater than when all were non-private. Experimental results also demonstrated that MC-MGM-1 executed more rapidly than MC-MGM-2, but generally reached a lower quality final solution. Both algorithms took two orders of magnitude fewer cycles to execute than MCA on the most challenging bounded optimization problems.

The second main contribution of this thesis was in addressing the problem of sensitivity analysis. The sensitivity analysis challenge was prompted by real-world engineering domains where resource constraints are not always as rigid as the DCOP formalism assumes. System users may be interested to know whether investing in a few more resources could significantly improve the functioning of the network of agents. In particular, the question posed in bounded optimization DCOPs is whether the addition of c units of resource at a particular variable or set of variables could yield more than c units of improvement in the global goal. The main challenges are to identify these bottleneck nodes in a distributed manner and without performing redundant computation.

This thesis explored two approaches to sensitivity analysis on bounded optimization DCOPs: link analysis and local reoptimization. Link analysis was a simple approach which examined each of the resource constraints in isolation to see whether the addition of resources locally would cause a significant improvement in the local contribution to the global objective. Local reoptimization harnessed the strengths of the complete and incomplete MC-DCOP algorithms by starting from the old global optimal and using the incomplete algorithms to locally re-optimize.

Experimental results demonstrated that considering the individual constraints in isolation from the network was too restrictive an approach. Link analysis was unable to identify any opportunities for disproportionate gains in the testcases developed for this thesis. Local reoptimization, which allowed for consideration of ripple effects in the network of agents, was more successful. Once again there was a tradeoff to be made between optimality and efficiency. MC-MGM-1 was able to do the reoptimization extremely rapidly and it identified most of the opportunities for improvement. MC-MGM-2 allowed for all of the opportunities in the developed testcases to be identified. However, it took longer to execute than MC-MGM-1. Finally, a heuristic was discovered for targeting the use of local reoptimization in sensitivity analysis. Variables with a high link density were identified as bottlenecks more frequently than those with a low link density.

In summary, this thesis contributes a more expressive extension to DCOP, called MC-DCOP, new algorithms for solving MC-DCOP problems and new approaches to performing sensitivity analysis. These contributions allow for the extension of the widely-used DCOP framework into domains that it previously could not encompass. These are domains with both a global objective and local resource constraints.

7.2 Future Work

The integration of intelligent agents into our environment, from routine office and classroom environments to extreme disaster response and sensor web environments, will lead to a large number of agents being connected and available continuously. In these future environments, algorithms that enable collaboration among networks of agents will be extremely valuable. Currently, certain fundamental issues in the creation of collaborative networks of agents remain open, specifically uncertainty and semi-cooperativeness.

Uncertainty: Whereas efficient models such as DCOP assume certainty about the interaction structure and the costs and benefits of interaction, uncertainty in interactions is often a prominent feature of real domains. Agents must interact with others without having certain knowledge of the benefits or costs of their coordinated actions. Additionally, network connectivity may not be completely reliable, with agents moving in and out of communication range. The Distributed Partially-Oberservable Markov Decision Process (Distributed POMDP) framework allows networks of agents to handle problems that involve uncertain information. However, Distributed POMDP algorithms are NEXP-complete, so they remain impractical for extremely large-scale domains [56, 40]. An open challenge is to develop more expressive models and algorithms that address both types of uncertainty, while maintaining the algorithmic efficiency of the current DCOP models.

Semi-cooperativeness: While this thesis has initiated research on semi-cooperative domains in my thesis, there is a wealth of new research to draw upon in nascent fields such as neuro-economics which indicates that while humans look out for their own interests, they do act in a non-self-interested fashion on many occasions [50, 15, 52, 5]. However, most research in the agents arena has ignored this work, basing itself on either purely self-interested users or purely cooperative agents. The investigation of the implications of having networks of agents with varying degrees of trust and cooperativeness working together is an important open problem. New algorithms are needed to allow users to make fine-grained tradeoffs between efficiency and self-interest as well as protocols for dealing with agents of differing or unknown cooperativeness.

Reference List

[1] S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS*, 2005.

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

[3] R.R. Bakker, F. Dikker, F. Tempelman, and P.M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI*, 1993.

[4] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[5] S. Bowles and H. Gintis. Social capital and community governance. *Economic Journal*, 112:419–436, 2002.

[6] J. Davin and P.J. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *AAMAS*, 2005.

[7] J. Davin and P.J. Modi. Hierarchical variable ordering for multiagent agreement problems. In *DCR*, 2006.

[8] M. Dawande and J. N. Hooker. Inference-based sensitivity analysis for mixed integer/linear programming. *Operations Research*, 48:623–634, 2000.

[9] F.Y. Edgeworth. *Mathematical Psychics: an Essay on the Application of Mathematics to the Moral Sciences.* Kegan Paul, 1881.

[10] J. A. Espinosa and E. Carmel. The impact of time separation on coordination in global software teams: a conceptual foundation. *Software Process Improvement and Practice*, 8, 2004.

[11] A. El Fallah, P. Moratis, and A. Tsoukias. An aggregation-disaggregation approach for automated negotiation in multi-agent systems. In *MultiAgents and Mobile Agents*, 2000.

[12] B. Faltings, D. Parkes, A. Petcu, and J. Shneidman. Optimizing streaming applications with self-interested users using mdpop. In *Workshop on Computational Social Choice*, 2006.

[13] A. Fedoruk and J. Denzinger. A general framework for multi-agent search with individual and global goals: Stakeholder search. *International Transactions on Systems Science and Applications*, 1:357–362, 2006.

[14] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–295. Kluwer, 2003.

[15] R.H. Frank. *Passions Within Reason: The Strategic Role of the Emotions.* W.W. Norton and Company, 1988.

[16] E.C. Freuder. Complexity of k-structured constraint satisfaction problems. In *AAAI*, 1990.

[17] M. Gavanelli. An algorithm for multi-criteria optimization in CSPs. In *ECAI*, pages 136–140, 2002.

[18] S. Goldmann, J. Mnch, and H. Holz. Distributed process planning support with milos. *International Journal of Software Engineering and Knowledge Engineering*, 2000.

[19] C. Gomes. Artificial intelligence and operations research: Challenges and opportunities in planning and scheduling. *Knowledge Engineering Review*, 15:1–10, 2000.

[20] R. Greenstadt. *Improving Privacy in Distributed Constraint Optimization.* PhD thesis, Harvard University, 2007.

[21] R. Greenstadt, J. P. Pearce, and M. Tambe. Analysis of privacy loss in DCOP algorithms. In *AAAI*, 2006.

[22] N.G. Hall. Sensitivity analysis for scheduling problems. *Journal of Scheduling*, 7:49–83, 2004.

[23] T. Hanne and S. Nickel. 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.

[24] D. Huisman. *Integrated and Dynamic Vehicle and Crew Scheduling.* PhD thesis, Erasmus University, 2004.

[25] I.B. Jaafar, N. Khayati, and K. Ghedira. Multicriteria optimization in CSPs: Foundations and distributed solving approach. In *AIMSA*, pages 459–468, 2004.

[26] P. Jalote and G. Jain. Assigning tasks in a 24-hour software development model. In *Asia-Pacific Software Engineering Conference (APSEC04)*, 2004.

[27] G. Kharmanda, N. Olhoff, A. Mohamed, and M. Lemaire. Reliability-based topology optimization. *Journal of Structural and Multidisciplinary Optimization*, 26:295–307, 2004.

[28] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13:32–44, 1992.

[29] V. Lesser, C. Ortiz, and M. Tambe. *Distributed Sensor Networks: A Multiagent Perspective*. Springer, 2003.

[30] 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 *AAMAS*, 2004.

[31] R. T. Maheswaran, J. P. Pearce, P. Varakantham, E. Bowring, and M. Tambe. Valuation of possible states: A unifying quantitative framework for evaluating privacy in collaboration. In *AAMAS*, 2005.

[32] R.T. Maheswaran, J.P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *International Conference on Parallel and Distributed Computing Systems (PDCS)*, 2004.

[33] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, 2004.

[34] N.F. Matsatsinis and P. Delias. A multi-criteria protocol for multi-agent negotiations. *Lecture Notes in Computer Science*, 3025, 2004.

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

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

[37] P. J. Modi and M. Veloso. Bumping strategies for the multiagent agreement problem. In *AAMAS*, 2005.

[38] P. Moraitis and A. Tsoukias. A multicriteria approach for distributed planning and negotiation in multiagent systems. In *ICMAS*, 1996.

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

[40] R. Nair. *Coordinating Multiagent Teams in Uncertain Domains Using Distributed POMDPs*. PhD thesis, University of Southern California, 2005.

[41] S. Okamoto. Allocating roles in large scale teams: An empirical evaluation. Master's thesis, University of Southern California, 2004.

[42] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Towards a formalization of teamwork with resource constraints. In *AAMAS*, 2004.

[43] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Security in multiagent systems by policy randomization. In *MSDM*, 2006.

[44] J.P. Pearce. *Local Optimization in Cooperative Agent Networks*. PhD thesis, University of Southern California, 2007.

[45] J.P. Pearce, R.T. Maheswaran, and M. Tambe. Solution sets for dcops and graphical games. In *AAMAS*, 2006.

[46] J.P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI*, 2007.

[47] F. Pecora, P.J. Modi, and P. Scerri. Reasoning about and dynamically posting n-ary constraints in adopt. In *Workshop on Distributed Constraint Reasoning*, 2006.

[48] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, Edinburgh, Scotland, Aug 2005.

[49] A. Petcu and B. Faltings. Mb-dpop: A new memory-bounded algorithm for distributed optimization. In *IJCAI*, 2007.

[50] M. Ridley. *The Origins of Virtue : Human Instincts and the Evolution of Cooperation*. Penguin Publishers, 1998.

[51] L. Schrage and L.A. Wolsey. Sensitivity analysis for branch and bound integer programming. *Operations Research*, 33:1008–1023, 1984.

[52] A. Sen. Rational fools: A critique of the behavioral foundations of economic theory. *Philosophy and Public Affairs*, 6, 1977.

[53] M. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, 2004.

[54] M. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, 2004.

[55] K. Stergiou and T. Walsh. Encoding non-binary constraint satisfaction problems. In *AAAI*, 1999.

[56] P. Varakantham. *Towards Efficient Planning for Real World Partially Observable Domains*. PhD thesis, University of Southern California, 2007.

[57] B. Viswanathan and M. desJardins. A model for large-scale team formation for a disaster rescue problem. In *AAMAS*, 2005.

[58] N. Vlassis, R. Elhorst, and J. R. Kok. Anytime algorithms for multiagent decision making using coordination graphs. In *International Conference on Systems, Man and Cybernetics*, 2004.

[59] L.A. Wolsey. Integer programming duality: Price functions and sensitivity analysis. *Mathematical Programming*, 20:173–195, 1981.

[60] 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.

[61] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *ICMAS*, 1996.

[62] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.

[63] Makoto Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer, 2001.

[64] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS*, 2003.

[65] X. Zhang and V. Lesser. Solving Negotiation Chains in Semi Cooperative Multi-Agent Systems. Technical report, University of Massachusetts, 2005.

Appendix A

Curriculum Vitae

Emma Bowring
University of Southern California
Powell Hall of Engineering, Room 516
Los Angeles
CA 90089
(213) 740-9569
bowring@usc.edu

A.1 Education

Doctor of Philosophy in Computer Science (in progress)
University of Southern California, Los Angeles, CA
Adviser: Professor Milind Tambe

Master of Science in Computer Science
University of Southern California, Los Angeles, CA 2006

Bachelor of Science in Computer Science
University of Southern California, Los Angeles, CA 2003
Minor: Spanish
Recognition: Summa Cum Laude

A.2 Experience

A.2.1 Research

Research Assistant Computer Science Dept, USC Los Angeles, CA 8/03-present
Developed an algorithm for distributed multi-criteria constraint optimization. Investigated privacy in distributed constraint optimization algorithms. Integrated distributed constraint optimization into CALO, a personal assistant project, for privacy preserving meeting and task negotiation.

Merit Research Scholar Integrated Media Systems Center, USC Los Angeles, CA 9/99-12/00
Redesigned Immersive Audio GUI and created test programs for Immersive Audio project.

A.2.2 Teaching

Course Co-Designer Computer Science Dept, USC Los Angeles, CA 1/06-1/07
Co-developed with Milind Tambe (Computer Science) and Anne Balsamo (Cinema) "Science, Technology and Culture: Artificial Intelligence and Science Fiction," a new interdisciplinary seminar course to be offered to freshman students at USC; experience developing the syllabus and selecting readings.

Co-Instructor School of Engineering, USC, Los Angeles, CA 10/06 Co-taught with Milind Tambe a seminar offered during Trojan Family Weekend to parents of Engineering students demonstrating innovative teaching techniques at USC. It covered material from our "Intelligent Agents and Science Fiction," class.

Co-Instructor Office of Undergraduate Programs, USC, Los Angeles, CA 8/06 Co-taught with Milind Tambe a micro-seminar offered during Welcome Week at USC to incoming freshmen excerpted from our "Intelligent Agents and Science Fiction" class.

Course Co-Designer Computer Science Dept, USC Los Angeles, CA 9/05-9/06
Co-developed with my advisor (Milind Tambe) "Intelligent Agents and Science Fiction," a new undergraduate course to be introduced as a technical elective in the computer science department; experience developing the syllabus, selecting readings and proposing the class to the computer science faculty.

Teaching Assistant Computer Science Dept, USC Los Angeles, CA 8/04-12/04
Taught "Advanced Artificial Intelligence," a graduate course, for one semester including experience conducting lectures, grading assignments and exams, organizing a panel discussion and addressing students' concerns.

A.2.3 Other

Jr. Software Engineer Symmetry Communications San Jose, CA 5/01-8/01
Developed an automated test suite for GPRS system using TCL and Expect and designed basic test suite architecture. Tested basic GPRS unit functionality.

A.3 Publications

A.3.1 Journals

- R.T. Maheswaran, J.P. Pearce, P. Varakantham, E. Bowring and M. Tambe, "Privacy Loss in Distributed Constraint Reasoning: A Quantitative Framework for Analysis and its Applications." Journal of Autonomous Agents and Multiagent Systems (JAAMAS), Springer vol 13 num 1, July 2006.

- P. Paruchuri, E. Bowring, R. Nair, J. P. Pearce, N. Schurr, M. Tambe and P. Varakantham, "Multiagent Teamwork: Hybrid Approaches." Communications of the Computer Society of India, 2006.

A.3.2 Conferences

- E. Bowring, M. Tambe and M. Yokoo, "Multiply-Constrained Distributed Constraint Optimization." Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06), Hakodate, Japan, May 8-12, 2006.

- M. Tambe, E. Bowring, H. Jung, G. Kaminka, R.T. Maheswaran, J. Marecki, P.J. Modi, R. Nair, P. Paruchuri, J.P. Pearce, D. Pynadath, P. Scerri, N. Schurr and P. Varakantham, "Conflicts in Teamwork: Hybrids to the Rescue." Proceedings of The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05), Utrecht, Netherlands, July 25-29, 2005.

- R.T. Maheswaran, J.P. Pearce, P. Varakantham, E. Bowring and M. Tambe, "Valuation of Possible States: A Unifying Quantitative Framework for Evaluating Privacy in Collaboration." Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05), Utrecht, Netherlands, July 25-29, 2005.

- R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce and P. Varakantham, "Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling." Proceedings of The Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04), New York, NY, July 19-23, 2004.

A.3.3 Short Papers

- R. Greenstadt, J. P. Pearce, E. Bowring and M. Tambe, "An Experimental Analysis of Privacy Loss in DCOP Algorithms." Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06), Hakodate, Japan, May 8-12, 2006.

A.3.4 Refereed Workshops and Symposia

- R. Greenstadt, J. P. Pearce, E. Bowring and M. Tambe, "An Experimental Analysis of Privacy Loss in DCOP Algorithms." Proceedings of Workshop on Distributed Constraint Reasoning (DCR-06), Hakodate, Japan, May 8-12, 2006.

- N. Schurr, P. Varakantham, E. Bowring, M. Tambe and B. Grosz, "Asimovian Multiagents: Applying Laws of Robotics to Teams of Agents and Humans." Proceedings of Workshop on Programming Multiagent Systems Languages and Tools (ProMAS-06), Hakodate, Japan, May 8-12, 2006.

- M. Tambe, E. Bowring, J.P. Pearce, P. Varakantham, P. Scerri, and D. Pynadath, "Electric Elves: What Went Wrong and Why." American Association of Artificial Intelligence (AAAI) Spring Symposium on What Went Wrong and Why, Stanford, CA, March 27-29, 2006.

- E. Bowring, M. Tambe and M. Yokoo, "Multiply-Constrained DCOP for Distributed Planning and Scheduling." American Association of Artificial Intelligence (AAAI) Spring Symposium on Distributed Plan and Schedule Management, Stanford, CA, March 27-29, 2006.

- E. Bowring, M. Tambe and M. Yokoo, "Distributed Multi-Criteria Coordination: Privacy vs. Efficiency." Proceedings of Workshop on Distributed Constraint Reasoning at the Nineteenth International Joint Conference on Artificial Intelligence (DCR-05), Edinburgh, Scotland, July 30 - Aug 5, 2005.

- E. Bowring, M. Tambe and M. Yokoo, "Distributed Multi-Criteria Coordination in Multi-Agent Systems." Proceedings of Workshop on Declarative Agent Languages and Technologies at the Fourth International Joint Conference on Agents and Multiagent Systems (DALT-05), Utrecht, Netherlands, July 25-29, 2005.

- E. Bowring, M. Tambe and M. Yokoo, "Optimize My Schedule But Keep It Flexible: Distributed Multi-Criteria Coordination for Personal Assistants." American Association of Artificial Intelligence (AAAI) Spring Symposium on Persistent Assistants: Living and Working with AI, Stanford, CA, March 21-23, 2005.

- R.T. Maheswaran, J.P. Pearce, P. Varakantham, E. Bowring, and M. Tambe, "Valuations of Possible States (VPS): A Quantitative Framework for Analysis of Privacy Loss Among Collaborative Personal Assistant Agents." American Association of Artificial Intelligence (AAAI) Spring Symposium on Persistent Assistants: Living and Working with AI, Stanford, CA, March 21-23, 2005.

A.4 Professional Activities

A.4.1 Reviewer

- Distributed Constraint Reasoning Workshop (DCR) 2006
- Journal of Artificial Intelligence Research (JAIR) 2005
- Distributed Constraint Reasoning Workshop (DCR) 2005
- International Joint Conference on Artificial Intelligence (IJCAI) 2005
- FLAIRS 2005
- Physics of Life Reviews 2004
- Brazilian Symposium on Artificial Intelligence 2004

A.4.2 Co-Chair

- AAMAS Women's Lunch and Panel 2005

A.5 Awards

- Outstanding Scholar, Computer Science Dept. 2003
- Order of Troy 2003
- USC Renaissance Scholar 2003
- Del Amo Study in Madrid Scholarship 2002
- WVT Rusch Engineering Honors Program 2001-2003
- Robert C Byrd Scholarship 1999-2003
- Jimmy Treybig Scholarship 1999-2003
- USC Trustee Scholar 1999-2003
- USC Merit Research Scholar 1999-2000