Sensitivity analysis for distributed optimization with resource constraints

Emma Bowring
University of the Pacific
3601 Pacific Ave
Stockton, CA, 95211, USA
ebowring@pacific.edu

Zhengyu Yin, Rob Zinkov, Milind Tambe
University of Southern California
3737 Watt Way
Los Angeles, CA, 90275, USA
{zhengyuy,zinkov,tambe}@usc.edu

ABSTRACT

Previous work in multiagent coordination has addressed the challenge of planning in domains where agents must optimize a global goal, while satisfying local resource constraints. However, the imposition of resource constraints naturally raises the question of whether the agents could significantly improve their team performance if a few more resources were made available. Sensitivity analysis aims to answer that question. This paper focuses on sensitivity analysis in the context of the distributed coordination framework, Multiply-Constrained DCOP (MC-DCOP). There are three main challenges in performing sensitivity analysis: (i) to perform it in a distributed fashion, (ii) to avoid re-solving an NP-hard MC-DCOP optimization from scratch, and (iii) to avoid considering unproductive uses for extra resources. To meet these challenges, this paper presents three types of locally optimal algorithms: link analysis, local reoptimization and local constraint propagation. These algorithms are distributed and avoid redundant computation by ascertaining just the effects of local perturbations on the original problem. Deploying our algorithms on a large number of MC-DCOP problems revealed several results. While our cheapest algorithm successfully identified quality improvements for a few problems, our more complex techniques were necessary to identify the best uses for additional resources. Furthermore, we identified two heuristics that can help identify a priori which agents might benefit most from additional resources: *density rank*, which works well when nodes received identical resources and *remaining resource rank*, which works well when nodes received resources based on the number of neighbors they had.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Design

Keywords

Distributed constraint reasoning, DCOP, multiagent systems

Cite as: Sensitivity Analysis for Distributed Optimization with Resource Constraints, Emma Bowring, Zhengyu Yin, Rob Zinkov, Milind Tambe, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 633 – 640

1. INTRODUCTION

Previous work in multiagent coordination has focused on the challenge of multiagent planning and coordination with resource constraints [12, 9, 4, 3, 13]. Multiply-Constrained DCOP (MC-DCOP)[12, 4, 3] is a framework for performing cooperative multiagent coordination in domains where there is a group goal to be optimized as well as a set of local resource constraints to be satisfied. It has been applied to domains such as meeting scheduling, distributed software development and sensor networks.

Performing sensitivity analysis, i.e. determining whether the agents could significantly improve performance with the addition of a few more resources is especially useful on MC-DCOP problems 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 MC-DCOP formalism assumes. In some domains, additional resources could be acquired if the benefits to the team sufficiently outweighed the costs [2, 1, 8]. For example, in the distributed meeting scheduling domain, 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. Given the resource constrained nature of the problems, sensitivity analysis provides a tool to answer such questions.

A simple approach to sensitivity analysis would be to run a complete MC-DCOP algorithm on all of the possible problem variants that result from introducing additional resources and compare the solutions. However, MC-DCOP is known to be an NP-hard problem, so running the complete algorithm combinatorially many times, would be very computationally expensive. Furthermore, the introduction of additional resources at a set of agents causes only a local perturbation to the problem. Thus, re-running 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.

We developed three distributed algorithms to perform sensitivity analysis on MC-DCOP problems: link analysis, local reoptimization and local constraint propagation. *Link Analysis* examines individual links to see if there is a significant gain to be made on a single link by inserting additional resources. *Local Reoptimization* takes existing solution and uses locally optimal MC-DCOP algorithms to reoptimize after additional resources have been inserted. It benefits from the fact that the old optimal is a new satisfying solutions and the extra resources produce only a local perturbation,

allowing a locally optimal algorithm to perform fairly well. In *Local Constraint Propagation* each node collects information about its neighboring nodes and their constraints and finds the optimal way to distribute the resources among itself and its neighbors.

We experimentally compared the quality of solutions found by each of the three types of algorithms on randomly generated examples as well as problems from the sensor networks domain. We further analyzed the results to identify which nodes most commonly were identified as requiring further resources and then developed heuristics to help speedily predict nodes that could benefit from additional resources.

2. PROBLEM DEFINITION

2.1 MC-DCOP

A Multiply-Constrained DCOP (MC-DCOP)[4, 3] consists of n agents, $\{x_1, x_2, \ldots, x_n\}$, which can take on any value from the discrete finite domain D_i. The goal is to choose values such that the sum over the set of constraint reward functions (f_{ij}) is maximized. Agents x_i and x_j are considered neighbors if they share a constraint. In addition there are a set of n-ary hard constraints with a cost function g_{ij} on a subset of our agent x_i's links and a g-budget G_i which the accumulated g-cost must not exceed. Together this g-function and g-budget constitute a g-constraint. Since g-cost functions cannot be merged with f-reward functions, each value must be selected based on both f and g, and hence the problem is multiply-constrained. Figure 1 shows an example MC-DCOP. There are three agents, each with two values, 0 and 1. There is a g-budget of 3 on agent 1. Each link has both a f-reward and a g-cost. When agents x_1, x_2 and x_3 take values 0,0 and 1 respectively, the assignment yields an optimal f-reward 5, but the g-cost is 4 which violates the g-budget of agent x_1; instead, if each agent takes the value of 0, the total f-reward of 4 is maximized without violation of agent 1's g-budget.

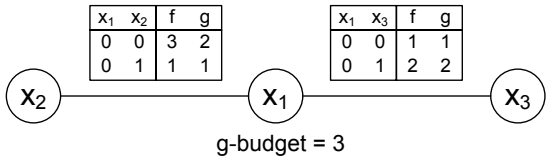

Figure 1: Multiply-Constrained constraint graph

The objective of an MC-DCOP is to find an assignment A of values to agents s.t. F(A) is maximized: $F(A) = \sum_{x_i,x_j \in V} f_{ij}(d_i, d_j)$, where $x_i \leftarrow d_i, x_j \leftarrow d_j \in A$
and $\forall x_i \in V$
$\sum g_{ij}(d_i, \{d_j | x_j \in neighbors(x_i)\}) \leq G_i$

There are two types of algorithms available for solving MC-DCOP algorithms: complete algorithms, e.g. MCA [4] and ADOPT-N [14], that find the globally optimal solution and incomplete algorithms, e.g. MC-MGM-1 is an 1-optimal or 1-coordinated algorithm, whereas MC-MGM-2 [3] is a 2-optimal or 2-coordinated algorithm, that rapidly find a locally optimal solution.

2.2 Sensitivity Analysis

Sensitivity analysis asks whether a significantly better team solution (higher f-quality) could be obtained by inserting a few additional resources (g) at one or more nodes. 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-reward of the globally optimal solution to the original MC-DCOP problem
- F^*_{new} is the 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 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 α; α is a proportional factor, which specifies the change in f that would typically accompany the insertion of 1 extra unit of g. If no MC-DCOPs yielding a gain/unit greater than α can be found, then there is no use investing in further resources. 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.

There are two slightly different sensitivity analysis problems: indivisible and divisible resources. In the indivisible resources problem, extra resources are given to a single agent and can only be used by that agent. In contrast, in the divisible resources problem, an agent can redistribute its extra resources to its neighbors. The potential solutions to the indivisible resources problem is a subset of the possible solutions to the divisible resources problem. In some circumstances, the ability to transfer resources is critical to improving the solution quality. Consider the example shown in Figure 2. In this example, the optimal solution to the original problem is (0, 0, 0). If we have 2 units of additional resources, then an algorithm that cannot divide the resources will settle on the solution (0, 1, 1) and give x_3 the extra resources. The globally optimal solution (1, 1, 1) can be reached only if we allow these additional 2 units of resources to be redistributed from x_3 to x_1 — assuming we allow x_3 to keep 1 unit of the resource and transfer the remaining 1 unit to x_1.

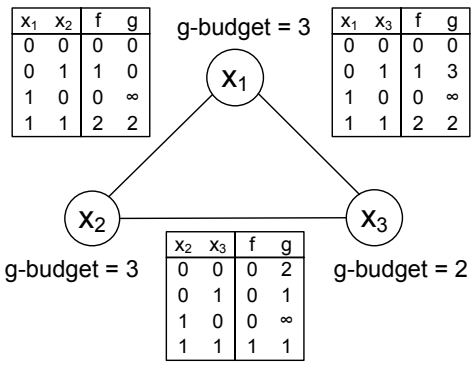

Figure 2: Indivisible Resources

The naive approach to sensitivity analysis is to re-run a complete algorithm on all of the possible problem variants and compare the solutions. However, this approach would involve running an exponential algorithm combinatorially many times (assuming resources could be divided among all n agents). The cost of performing the analysis would outweigh the benefits of performing it. Furthermore, the introduction of additional resources causes only a local perturbation to the problem. Thus, re-running from scratch on each variant would require performing many duplicate computations. Therefore, the main challenge in sensitivity analysis is to reduce both the time taken to solve each problem variant by using incomplete algorithms and to target high probability variants using heuristics derived from experimental analysis. We look at the former problem in Section 3 and the latter in Section 4.

2.3 Motivating Domains

In the experimental section of this paper we apply our algorithms to the wireless mobile sensor network domain[1]. Each sensor can receive and transmit signals from its neighbors and they must establish a communication network on the fly. The radio communication between these sensors suffers from reflection, scattering and diffraction. The actual signal strength encountered in two sensor locations is sampled from a probability distribution such as a normal distribution. The result is that small-scale movements of these sensor nodes in a fixed network topology can create significant improvements in communication strength. Furthermore, each sensor has a finite battery power, which directly controls its signal strength. Sensors in two locations require a certain battery power to ensure that the mean signal strength reaches a minimum threshold. If the sensors are farther apart, more battery power must be consumed by each sensor to maintain the same minimum threshold.

This domain can be mapped onto a MC-DCOP where the sensors are agents and the domain values are possible locations for the sensor. The signal strength between two sensors is the f-reward and the battery power consumed and the total power available map onto g-cost and g-budget respectively. Thus, given an agent x_1 at location l1 and x_2 at location l2, their signal strength is one sampled from a normal distribution. The amount of battery power required is a function of the Euclidean distance between the agents and gaussian noise to account for error in the battery and communication channel.

Sensitivity analysis is particularly relevant in this domain, because the noise in the domain means that small adjustments in the position and power consumption of the sensors could dramatically affect the strength of the communication links in the network.

3. ALGORITHMS

3.1 Link Analysis

One way that a fixed amount of extra g inserted at a single node could yield a disproportionate improvement in f could occur is when the g vs. f function on an individual link has a slope greater than α. If one plots the local g vs. f for a link and eliminates all dominated value-pairs (those with both high g-cost and low f-reward), the slope can be: less than α, equal to α, or greater than α. The local slope is the slope in the interval which is within a small distance (R_{max}) of the current g-budget. Link analysis examines the g vs. f function at every single link by constructing a g vs. f map and seeing whether the local slope is greater than α. It can only consider changes in assignment that will not create new g-constraint violations for any of the neighboring nodes. It can thus identify single link gains.

Link analysis (Algorithm 1) is a fully distributed algorithm. Nodes are given a top to bottom priority ordering as often done in DCOP algorithms. Each agent then checks gain due to additional g-budget and passes the max gain to the next agent in a round-robin fashion; max gain is reported.

Algorithm 1 Link Analysis

```
 1: maxgain = α
 2: for All nodes x_i from top to bottom priority do
 3:    maxgain = ReceiveMaxGain()
 4:    for All neighbors x_j do
 5:       ComputeGFmap(x_i, x_j)
 6:       for i = g_cur ... g_cur + R_max do
 7:          s = FindSlope(g_cur, i)
 8:          if s > maxgain and NoViolations() then
 9:             maxgain = s
10:       PassMaxGain(maxgain)
11: ReportMaxGain()
```

3.2 Local Reoptimization

Since the additional resources have been inserted at a localized set of nodes, their 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 because their optimization mechanism revolves around optimizing and re-optimizing the local value assignments. 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 [10].

The local reoptimization approach to sensitivity analysis involves running MC-MGM on variants of the original MC-DCOP problem. Here nodes are again organized into top to bottom priority ordering. The pseudo-code for the local reoptimization algorithm is shown in Figure 11. 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 node is 0, then the algorithm abandons the attempt to insert $R-1, R-2, \ldots, 1$ units of g into that particular node because they will also yield a gain of 0 (line 5). Additionally, while we have gain/unit as the criterion here, absolute gain could used as well. Each node obtains gain from line 6 and 7 by initiating MC-MGM algorithms. Control is then passed from node to node in a round-robin fashion. The entire process is fully distributed (however, please see footnote 2).

Algorithm 2 Local Reoptimization (LR)

```
 1: maxgain = α
 2: for all nodes in top to bottom priority order do
 3:    R = R_max
 4:    maxgain = ReceiveMaxGain()
 5:    while R ≥ 1 and gain > 0 do
 6:       run MC-MGM on new problem
 7:       gain = computeGain()
 8:       if gain > maxgain then
 9:          PassMaxGain(gain)
10:       decrement R
11: ReportMaxGain()
```

[1]This domain is motivated by DARPA's "LANDroids" project: http://www.darpa.mil/ipto/programs/ld/ld.asp

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, 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. 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.

3.3 Local Constraint Propagation (LCP)

The local constraint propagation algorithm is based on the idea that an agent can reimburse its neighbors for any additional g-cost they incur due to the agent changing values. It was inspired by the optimization mechanisms used in the DCOP algorithms OptAPO [11] and DPOP [15].

The agents are organized into a DFS tree — similar to MCA[4] — so that there are links between agents and their ancestors /descendants, but not between separate sub-trees. Like in Local Reoptimization, agents initially take on the optimal solution to the original problem to reduce redundant computation. Execution then proceeds in rounds. In each round, each agent x_i sends a table to each of its neighbors x_j indicating by how much x_i's g-budget would be exceeded if x_j took on each of the values in its domain (line 4). Each agent x_i then considers all of the possible valid moves it could make and picks out the best one (line 6). A move is valid if the total additional g required for the move by x_i and all its neighbors is less than or equal to the additional units of g being inserted. After that, a bottom to top synchronous communication of messages is used to identify the agent who can gain the maximum reward by taking its best move. In this phase, each agent collects the gains reported from its children, compares them to its own gain (line 9), and reports the maximum gain in the subtree to its parent (line 10). Finally, the root will decide which agent is allowed to take its move and a top to bottom communication is used to inform all agents whether they should move or not (line 11 to line 23). At the same time, all the agents are informed how many shared resources have been consumed so that they know how many available units they can use for the next round (line 14). Notice here the unit consumption can be negative which means some units can be put back to the shared budget if after the move the total g required goes down. The algorithm terminates when no valid moves remain that can yield a positive gain. Pseudo-code using pre-defined tree hierarchy and message passing is shown below.

In LCP-1, only one agent can change its value in a single round. In LCP-2, by contrast, agents can consider both unilateral moves and coordinated moves by pairs of agents. To coordinate their actions, agents must share more elaborate $GExceed$ tables that show how both a neighbor's potential value changes and all potential joint moves would affect the resource requirements of a particular agent. In particular, sharing $GExceed$ tables proceeds in two phases – request phase and response phase. In the request phase, every agent sends a request table with all its possible joint moves to all neighbors to inquire about their g-budget excess if a certain joint move happens. In the response phase, each agent computes and fills in its g-budget excess for each joint move in the request table and sends the table back. (Notice if an agent is the neighbor of two agents that propose a joint move, it will report its g-budget excess regarding this move to only one of them.) It also requires additional cycles of communication to coordinate the joint moves at the end of each round.

Algorithm 3 Local Constraint Propagation-1 (LCP-1)

```
1:  Pre-Processing: Build DFS tree
2:  repeat
3:      Phase 1: Message Broadcasting
4:      Send_Message(Neighbors, GExceed)
5:      ExceedTable = Receive_Message(Neighbors)
6:      [myValue, myGain, myUnits] = Max_Gain(ExceedTable, AvailabeUnits)
7:      Phase 2: Bottom to Top
8:      if isLeaf() or receiveFromAllChildren() then
9:          [ID, Gain, Units] = Compare_Gain(myGain, max(ChildrenGain))
10:         Send_Message(Parent, Gain, Units)
11:     Phase 3: Top to Bottom
12:     if isRoot() or receiveFromParent() then
13:         [canMove, UnitsUsed] = Receive_Message(Parent)
14:         AvailableUnits = AvailableUnits - UnitsUsed
15:         if canMove then
16:             if ID = Self then
17:                 Take_Move(myValue)
18:                 Send_Message(Children, False, UnitsUsed)
19:             else
20:                 Send_Message(ID, True, UnitsUsed)
21:                 Send_Message(OtherChildren, False, UnitsUsed)
22:         else
23:             Send_Message(Children, False, UnitsUsed)
24: until Cannot find valid move with gain>0
```

4. EXPERIMENTAL RESULTS

We conducted experiments on two domains: random graphs and sensor networks, to attempt to answer several questions: (i) whether the ability to redistribute divisible resources makes a significant difference in the quality of the solution found by sensitivity analysis, (ii) how the different sensitivity analysis algorithms perform in comparison with each other in terms of quality and runtime, (iii) what heuristics could be used to target high probability nodes for sensitivity analysis in the future.

4.1 Random Graphs

The experiments in this section were run on four sets of random graph problems, with 80 problems in each set. The first set consisted of random graphs where all nodes were provided a fixed g-budget and their g-costs per link were sampled from a uniform random distribution ranging from 1 to 10. The second set also had 10-node graphs, but their g-budgets varied from node to node based on the link density of the node. The third and fourth sets were similar except that they were 15-node graphs. The size of the graphs was limited by the run-time for obtaining an optimal solution.

The MC-DCOP instances of most interest to us for sensitivity analysis turned out to be particularly difficult. More specifically, Bowring et al [4] illustrate that when agents' g-budgets are too low or too high relative to the overall g-cost on their individual links, MC-DCOPs are solved quickly; in contrast, mid-range g-budgets create hard MC-DCOP problems. We created MC-DCOP problem instances with such mid-range g-budgets — it is when we have mid-range g-budgets that sensitivity analysis is most useful. If budgets are too low or too high, adding in some extra resources would likely provide little additional benefit.

For each set of graphs we used MCA to obtain the globally optimal solution to the original problem and then we performed sensitivity analysis to see how adding 5 extra units of g would affect the solution. We ran each of the following five algorithms on the problem: Link Analysis, Local Reoptimization with MC-MGM-

1 (LR-1), Local Reoptimization with MC-MGM-2 (LR-2), Local Constraint Propagation 1 (LCP-1) and Local Constraint Propagation 2 (LCP-2). We then compared the quality of the solutions they each found. Since all of these algorithms reoptimize rather than starting from scratch we wanted to see how their results compared to the optimal solution, so we used a centralized brute force algorithm to calculate the globally optimal solution. This gave us an indication of how well the algorithms were reoptimizing. In the following section, we first compare the quality of the solutions obtained by the different algorithms and their runtimes. We next look at the effects of changing a problem from one of indivisible to divisible resources and finally analyze the globally optimal solutions to see what features of individual agents make them more likely to benefit from additional resources.

Quality Gains: Tables 1 and 2 show the results of our analysis on the 10- and 15-node indivisible resource problems respectively. In the 10-node case, with fixed g-budget, 53 out of 80 graphs showed a significant improvement due to sensitivity analysis; the proportion was higher, 72/80, for graphs with variable g-budgets. For 15-node problems, sensitivity analysis identified more situations where extra resources could be used. This is because larger graphs have more constraints to potentially act as bottlenecks. In each table, the first column shows the algorithm used and the second column shows the average total gain in solution quality. The third column shows the percentage of times the algorithm was able to identify the maximum gain, e.g. LR-2 for 10-node fixed g-budget graphs was able to identify the maximum gain achievable in 56.6% of the cases (out of 53), whereas LCP-2 was able to do so in 62.3% of the cases. The fourth column shows the percentage of cases where the algorithms found significant gains, i.e. gains higher than the proportionality factor α. As can be seen in the optimal row, it was not always possible to find a significant gain by inserting resources: for only 43.4% of the fixed budget case and 50% of the variable budget cases would it be worthwhile to add additional resources. The remaining columns repeat the information, but for graphs where nodes were given g-budgets based on link density.

Table 1: Absolute Gain: 10-node graphs

Algorithm / Improved Ratio	10 Fixed 53/80			10 Variable 72/80		
	Avg	%W	%U	Avg	%W	%U
Link Analysis	0.43	7.5	1.9	2.49	33.3	23.6
LCP-1	3.25	45.3	28.3	3.54	51.4	36.1
LCP-2	3.81	62.3	30.2	4.25	73.6	44.4
LR-1	2.85	43.4	26.4	3.58	51.4	36.1
LR-2	3.23	56.6	28.3	3.86	58.3	40.3
Optimal	4.92	100.0	43.4	5.06	100.0	50.0

Table 2: Absolute Gain: 15-node graphs

Algorithm / Improved Ratio	15 Fixed 64/80			15 Variable 78/80		
	Avg	%W	%U	Avg	%W	%U
Link Analysis	0.56	1.6	6.3	2.41	26.9	21.8
LCP-1	3.08	39.1	25.0	3.51	48.7	33.3
LCP-2	3.84	51.6	32.8	4.44	69.2	42.3
LR-1	3.08	39.1	25.0	3.50	47.4	33.3
LR-2	3.78	54.7	34.4	3.59	51.3	34.6
Optimal	5.72	100.0	51.6	5.53	100.0	60.3

We can draw the following conclusions from the results:

- Link analysis, the cheapest of our algorithms performs poorly on graphs of fixed g-budgets, rarely obtaining the maximum gain and obtaining an average of 0.43 or 0.56 in reward gain. However, link analysis performs better in the variable-g case, obtaining a higher reward gain on average. The reason for this is that in the fixed budget cases, the nodes identified as requiring further resources were generally the most highly connected nodes. For highly connected nodes link analysis was too constrained by its requirement to not change the values of any neighbors to effectively use the additional resources. By contrast, the nodes that received extra resources in the variable budget cases were sometimes highly connected and sometimes not highly connected, so link analysis was able to perform better.
- The performance of the LCP-1 and LR-1 algorithms are almost identical. In these cases, it appears that the inability to break out of local maxima had more of an effect on their performance than the algorithm applied to the problem. LCP-2 performs moderately better than LR-2 at identifying uses of additional resources. The LR-2 and LCP-2 algorithms outperform the LR-1 and LCP-1 algorithms because they were able to use joint moves to break out of some of the local optima.
- In comparing the results of the incomplete algorithms to the global optimal (in particular by examining the %W column) we can see that the incomplete algorithms performed well. The 2-coordinated algorithms found the globally optimal solution in 51 - 74% of the cases where gains could be achieved and the unilateral algorithms found the globally optimal solution in 39 - 51% of the cases.

We performed a similar analysis on problems where we maximized the gain per unit of resource inserted rather than total absolute gain for all 320 cases. The results here are similar to the ones seen in the absolute gain case. Link analysis was still ineffective on the fixed g-budget problems and LCP-2 performed moderately better than LR-2.

Table 3: Gain per Unit: 10-node graphs

Algorithm / Improved Ratio	10 Fixed 53/80			10 Variable 72/80		
	Avg	%W	%U	Avg	%W	%U
Link Analysis	0.19	5.7	11.3	1.36	33.3	52.8
LCP-1	1.76	37.7	50.9	2.03	50.0	66.7
LCP-2	2.07	56.6	64.2	2.52	70.8	73.6
LR-1	1.75	37.7	50.9	2.03	50.0	66.7
LR-2	1.93	49.1	56.6	2.13	55.6	68.1
Optimal	2.96	100.0	79.2	3.06	100.0	83.3

Table 4: Gain per Unit: 15-node graphs

Algorithm / Improved Ratio	15 Fixed 64/80			15 Variable 78/80		
	Avg	%W	%U	Avg	%W	%U
Link Analysis	0.38	1.6	9.4	1.28	24.4	53.8
LCP-1	1.64	39.1	54.7	2.22	47.4	71.8
LCP-2	2.08	53.1	68.8	3.10	67.9	89.7
LR-1	1.63	39.1	53.1	2.17	44.9	71.8
LR-2	2.01	53.1	67.2	2.26	48.7	74.4
Optimal	2.98	100.0	87.5	3.84	100.0	96.2

Indivisible vs. Divisible Resources: In the previous experiment, resources were considered indivisible. Tables 5 and 6 shows the results of running the LCP-1, LCP-2 and optimal algorithms

on the 320 example problems in our testsuite under one of two assumptions: either (i) resources are indivisible and must be used by the agent they are given to, or (ii) resources are divisible and can be redistributed by agents to their neighbors and beyond. In all cases, allowing the agents to redistribute resources yields a much better final solution quality: in some cases a 2-fold improvement in the quality gain. This is because agents who aren't using all of their available resources can donate them to another agent, allowing the network to make more efficient use of the resources available. Thus, in problems where resources are transferable from agent to agent, it is well worth using an algorithm that will consider sharing resources between agents.

Table 5: Indivisible vs. Divisible Resources: 10-node graphs

Algorithm Improved Ratio	10 Fixed 59/80			10 Variable 74/80		
	Avg	%W	%U	Avg	%W	%U
LCP-1 indivisible	2.92	20.3	18.8	3.45	24.3	32.5
LCP-1 divisible	4.31	40.7	31.3	4.85	44.6	45.0
LCP-2 indivisible	3.42	30.5	20.0	4.14	33.8	40.0
LCP-2 divisible	5.03	64.4	36.3	6.00	73.0	53.8
Optimal indivisible	4.42	44.1	28.8	4.92	43.2	45.0
Optimal divisible	6.71	100.0	42.5	6.86	100.0	60.0

Table 6: Indivisible vs. Divisible Resources: 15-node graphs

Algorithm Improved Ratio	15 Fixed 66/80			15 Variable 78/80		
	Avg	%W	%U	Avg	%W	%U
LCP-1 indivisible	2.98	22.7	20.0	3.51	11.5	32.5
LCP-1 divisible	3.91	34.8	25.0	5.65	42.3	56.3
LCP-2 indivisible	3.73	34.8	26.3	4.44	12.8	41.3
LCP-2 divisible	4.76	48.5	36.3	7.31	64.1	73.8
Optimal indivisible	5.55	54.5	41.3	5.53	16.7	58.8
Optimal divisible	6.83	100.0	52.5	8.65	100.0	82.5

Run-time Results: Figure 3 provides run-time results for our algorithms from the gain per unit experiments. To find the maximum gain per unit, one must look at potentially inserting R_{max}, $R_{max} - 1, \ldots, 1$ additional units of resources, so the runtime is greater than for problems seeking just to maximize the absolute gain. The local reoptimization algorithms ran more quickly than the local constraint propagation algorithms. For example, for LR-1, for 10-node fixed-g problems, the average run-time was 14.11 cycles, but it was 108.39 for LCP-1, which constitutes a 7.68-fold slowdown. The slowdown in LCP-1 occurs because messages must be passed up and down the DFS tree. Thus, every value change in LCP-1 requires $2 * depth$ message communication cycles, and since it iterates over R_{max} units, it has a minimum cost of $R_{max} * 2 * depth$. Thus, the improved quality of the solutions that LCP-1 identified came at a cost of an 8-fold slowdown in cycles.[2] The algorithms running on problems with divisible resources ran slightly slower than those running on equivalent problems with indivisible resources. However, the 8 - 21% increase in runtime for running on divisible resource problems, is more than offset by the improved quality of the solution.

Heuristic Analysis: We analyzed the globally optimal solutions to the random graph problems to identify which nodes most benefited from additional resources. We then looked at what features of these nodes might lead to their being bottlenecks in the hopes

[2]However LR currently does not include termination detection and hence LCP's slowdown, reported here, may be an overestimate.

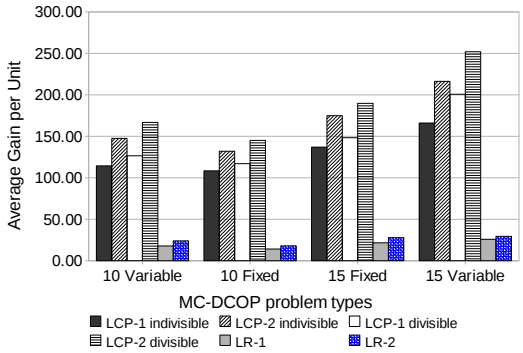

Figure 3: Cycle Comparison Chart

that we could identify heuristics that would allow us to target our analysis a priori to high probability nodes in the future. We identified two key heuristics: *degree rank* and *remaining resource rank* and analyzed how well they selected the bottleneck node(s). The *degree rank* of a node x_i is n, if there are $n-1$ nodes in the graph that have a degree greater than x_i. Thus the node with degree rank 1 is the most highly connected node. The *remaining resource rank* of a node x_i is n, if there are $n-1$ nodes in the graph that have a remaining g-budget after solving the original MC-DCOP smaller than x_i's remaining g-budget. Thus any nodes that used their entire g-budget to solve the original MC-DCOP would have a remaining resource rank of 1.

Table 7 shows the performance of the Degree Rank heuristic as applied to ten and fifteen node problems. For problems where each node was given the same g-budget, Degree Rank identifies the nodes most likely to require additional resources with high frequency. For example, it identified in the 10-node fixed g-budget cases, that in 55 out of 115 cases it is the node with the highest degree rank appeared to be the most critical. It is successful because the g-budget each node receives is generally sufficient to meet the needs of nodes with only a few links on which to expend resources, but not sufficient for those agents with many neighbors. In problems where the g-budget assigned to each node was proportional to its link density, the Degree Rank heuristic was largely unsuccessful at identifying the bottleneck node. This was because the agents had already been given resources based on their link density and so highly connected nodes were no more likely to need additional resources than any other node.

Table 7: Degree Rank Frequency Pattern

Rank	10 Fixed	10 Variable	15 Fixed	15 Variable
1	55	22	64	12
2	21	17	21	13
3	23	23	13	18
4	8	13	4	16
5	5	12	5	17
6	2	18	5	16
7	1	15	1	19
8	0	12	0	12
9	0	14	0	8
10	0	9	0	24
11	-	-	0	7
12	-	-	0	13
13	-	-	0	8
14	-	-	0	10
15	-	-	0	4

Table 8 shows the performance of the Remaining Resource Rank Heuristic. The heuristic is fairly effective in problems with identical g-budgets for each agent and variable g-budgets assigned based on link density. While not performing as well as Degree Rank on the fixed g-budget problems, Remaining Resource Rank made much better predictions on problems where the g-budgets varied. The heuristic relies on the idea that an agent that has already used up all of its resources is more likely to have a need for more than an agent that still has some unused resources remaining after solving the initial problem.

Table 8: Remaining Resource Rank Frequency Pattern

Rank	10 Fixed	10 Variable	15 Fixed	15 Variable
1	47	39	50	46
2	14	17	16	5
3	16	9	17	16
4	15	13	12	18
5	11	21	8	11
6	5	20	4	14
7	2	10	5	7
8	0	11	1	12
9	1	11	0	10
10	4	4	0	13
11	-	-	0	11
12	-	-	0	12
13	-	-	0	8
14	-	-	0	8
15	-	-	0	6

4.2 Sensor Networks

In the sensor network domain, networks were created by uniformly generating a location of a single node and then proceeding to generate each subsequent node such that it was within signal range of an existing node. This reflects the fact that the network must be connected. Two nodes were considered to be within signal range when the distance between them was below a particular threshold.

Ten networks, each with ten nodes, were created and density was varied by altering how close together agents had to be to communicate. F-rewards and g-costs were generated for each possible configuration the two neighboring agents could take. A g-budget was determined that was sufficiently tight that tradeoffs had to be made between signal strength and power consumption. This was done to see if our algorithms could capitalize on opportunities to improve the solution.

Quality Gains: Figures 4 and 5 show the results of performing sensitivity analysis on 10 and 15-node problems from the mobile sensor network domain. In these problems, agents were initially given 1000 units of battery capacity and the problem given to the sensitivity analysis algorithms was to see whether any of the agents should have their battery upgraded to the next largest capacity, which was 100 units larger. In these problems, α has the value 5 because inserting 1000 additional units of battery capacity usually yields an improvement of 5000 in the solution quality. Sensitivity analysis identified 8 out of 20 problems where additional resources could significantly improve the team solution. For the 15-node problems, the algorithms were able to more dramatically improve the quality of the team solution than for the 10-node problems. This pattern is even more pronounced than it was for the random graph problems.

Heuristic Analysis: We analyzed the globally optimal solutions to the sensor network problems to identify which heuristics best classified the features of the nodes most likely to be identified as bottlenecks in the sensor network domain. As can be seen in Table 9, the Remaining Resource Rank heuristic performed well in this domain. This agrees with our expectations in mobile sensor networks, since more battery power allows the agent to connect to more nodes i.e. the problem is similar to the variable g-budget problem previously. It can be seen that this heuristic works well in both problems that resources are indivisible and divisible.

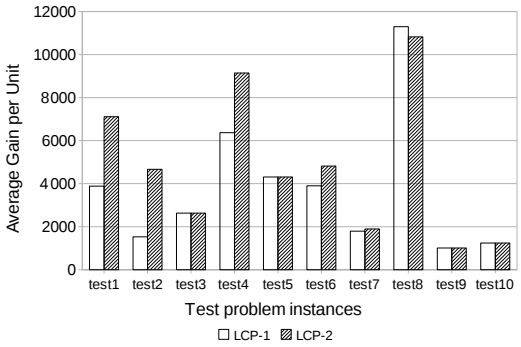

Figure 4: 10-Node Sensor Network Problems

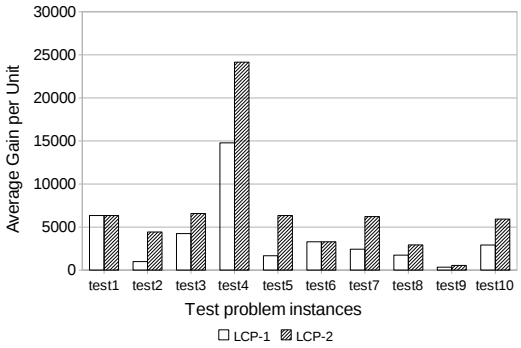

Figure 5: 15-Node Sensor Network Problems

Table 9: Remaining Resource Rank Frequency Pattern

Rank	Divisible	Indivisible
1	8	6
2	3	1
3	3	1
4	3	0
5	1	1
6	0	0
7	2	0
8	0	0
9	0	0
10	0	0

5. CONCLUSION AND RELATED WORK

While significant recent research has focused on multiagent planning with resources constraints [12, 9, 4, 3, 13], little attention has been focused on the challenge of sensitivity analysis. This paper in MC-DCOPS, i.e. focused on the problem of sensitivity analysis or whether the solution to a resource constrained optimization problem might be significantly improved if more resources were made

available. The three primary challenges were (i) to perform sensitivity analysis in a distributed fashion, (ii) to minimize the redundant computation when reoptimizing, and (iii) to find heuristics that would allow the search for bottleneck agents to be targeted a priori. This paper presented three new algorithms for performing sensitivity analysis: link analysis, local reoptimization and local constraint propagation. These algorithms start from the globally optimal solution to the original MC-DCOP problem and try to ascertain just the effects of the local perturbation caused by inserting additional resources. This helps to minimize the redundant computation.

Our experimental analysis led to the following observations. (i) The local constraint propagation algorithms found slightly better uses for additional resources than the local reoptimization algorithms. However, local reoptimization ran significantly more rapidly than local constraint propagation. (ii) Link analysis proved largely ineffectual on problems where the bottleneck nodes were those of high link density because it was too constrained to make much use of the additional resources available. (iii) Allowing agents to divide up their resources to other agents caused teams to perform significantly better, with only a small increase in the runtime required to solve the problem. Experimentally we also identified two heuristics that can help identify high probability nodes for giving extra resources to: *density rank*, which works well when nodes received roughly the same initial g-budget and *remaining resource rank*, which works well when nodes received resources based on link density.

The problem of sensitivity analysis, while new in the distributed constraint reasoning field, has been extensively studied in linear and integer programming (LP/IP) and centralized constraint satisfaction (CSP). 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 [2, 6, 5, 7, 18, 16]. 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. Related work in CSP looks at the problem of reoptimizing after removing constraints entirely rather than making them easier to satisfy [1, 17]. This is a different problem to the one considered in this paper and since the techniques don't consider soft constraints, they would not work for the problem defined in this paper.

ACKNOWLEDGEMENTS

This research was supported by a subcontract from Perceptronics Inc.

6. REFERENCES

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

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

[3] E. Bowring, J. Pearce, C. Portway, M. Jain, and M. Tambe. On k-optimal distributed constraint optimization algorithms: New bounds and algorithms. In *AAMAS*, 2008.

[4] E. Bowring, M. Tambe, and M. Yokoo. Multiply-constrained dcop for distributed planning and scheduling. In *AAMAS*, 2006.

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

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

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

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

[9] H. Li, H. Durfee, and K. G. Shin. Multiagent planning for agents with internal execution resource constraints. In *AAMAS '03*, pages 560–567, New York, NY, USA, 2003.

[10] 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.

[11] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society.

[12] T. Matsui, H. Matsuo, M. Silaghi, K. Hirayama, and M. Yokoo. Resource constrained distributed constraint optimization with virtual variables. In *AAAI*, pages 120–125, 2008.

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

[14] F. Pecora, P. Modi, and P. Scerri. Reasoning About and Dynamically Posting n-ary Constraints in ADOPT. In *Proceedings of 7th Int. Workshop on Distributed Constraint Reasoning, at AAMAS*, 2006.

[15] A. Petcu, B. Faltings, and R. Mailler. PC-DPOP: A new partial centralization algorithm for distributed optimization. *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 7:167–172, 2007.

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

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

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