A Distributed Constraint Optimization Approach to Wireless Network Optimization

Heeten Choxi and **Pragnesh Jay Modi**
Department of Computer Science
Drexel University
Philadelphia, PA 19104
heeten@drexel.edu

Abstract

We present a new algorithm called Variable Message Size (VMS) ADOPT for solving Distributed Constraint Optimization Problems (DCOP) which trades off message size and memory usage for running time. The algorithm is applied to a wireless network optimization problem, in which small robots act as wireless routers with the objective of maximizing signal strength in the network by repositioning themselves. Memory and bandwidth are limited resources in this application; our algorithm incorporates features of ADOPT and DPOP and introduces a parameter which controls the memory usage and message size at each agent. Bounded-error approximation can also be used to trade off solution quality for running time.

Introduction

Distributed Constraint Satisfaction Problems(DisCSP) (Yokoo *et al.* 1998) and Distributed Constraint Optimization Problems (Modi *et al.* 2006; Petcu & Faltings 2005) provide useful models and algorithms for performing optimization tasks. In (Gerkey, Mailler, & Morisset 2006), Gerkey obtained promising results using a variant of asynchronous backtracking, a DisCSP solver, in the Commbots problem in which robot nodes change their position to optimize the quality of wireless routes. This paper applies a new DCOP algorithm to the wireless network optimization problem, a variant of the Commbots problem.

Due to the physics of signal propagation, small movements are capable of producing significant changes in the signal strength between nodes. Many laptop users have experienced this phenomena when small shifts in their laptop position resulted in a change in wireless signal strength. Robots can be used to exploit this phenomena, creating a self-optimizing network that results in improved performance.

In order to apply DCOP algorithms to this problem it is important that they can operate on robots with limited memory and communicate using wireless networks with limited bandwidth. One popular DCOP algorithm that uses polynomial amounts of memory and bandwidth is ADOPT (Modi *et al.* 2006). Although ADOPT could run within the memory and bandwidth limitations of the system it requires an exponential number of messages to be sent to perform the optimization. ADOPT supports bounded error approximation, which allows it to find a suboptimal solution in less time with less messages. Another DCOP algorithm DPOP (Petcu & Faltings 2005) takes a linear number of messages to perform optimization, but requires an exponential amount of memory and bandwidth. In order to fully exploit the resources of the robots and wireless networks we have created a new algorithm VMS ADOPT that incorporates features of DPOP into ADOPT through a parameter which trades off message size and memory usage for running time. The algorithm also performs bounded-error approximation, allowing solution quality to be traded off for running time.

In Section 2 we formally define what a DCOP is and the DCOP representation of the robotic wireless network optimization problem. In Section 3 we go over related work in the areas of DCOP and wireless network optimization. In Section 4 we briefly describe ADOPT, DPOP, and VMS ADOPT. In Section 5 we discuss our experiments. In Section 6 we discuss our results and future work.

Problem Description

In this section we describe a DCOP and the DCOP model for robotic network optimization. We use the formalization of DCOPs found in (Modi *et al.* 2006). Throughout the rest of the paper we use the terms agent, robot, and network node interchangeably.

DCOP Formalism

A Distributed Constraint Optimization Problem (DCOP) consists of n variables $V = \{x_1, x_2, ...x_n\}$, each assigned to an agent, where the values of the variables are taken from domains $D_1, D_2, ..., D_n$, respectively. The goal for the agents is to choose values for variables such that a given global *objective function* is minimized. The objective function is described as the summation over a set of *cost functions*. A cost function for a pair of variables x_i, x_j is defined as $f_{ij} : D_i \times D_j \rightarrow N$. The objective is to find an assignment $\mathcal{A}^*$ of values to variables such that the aggregate cost F is minimized.

Copyright © 2007, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Robotic Network Optimization DCOP

Each robot is represented by an agent A_i and has a variable x_i. The domain for agent A_i is D_i and consists of the valid positions the agent can move to. Constraints exist between any pair of agents that share a wireless link. The cost function for a pair of variables x_i, x_j is defined as $f_{ij} : D_i \times D_j \rightarrow N$. $f(x_i, x_j)$ represents power loss in decibels of the wireless link from robot i at position x_i to robot j at position x_j.

In wireless networks power loss is due to distance from the transmitter, large-scale, and small scale fading (Andersen, Rappaport, & Yoshida 1995). Fading is caused by constructive and destructive interference from radio waves propagating through an environment. Equation 1 represents power-loss due to distance and large scale fading. d represents distance in meters. $PL(d_0)$ is the power loss at a known reference distance d_0. χ_σ represents gaussian noise with a standard deviation of σ. n_p represents how quickly power is lost as distance increases. The units of $PL(d)$ are decibels (dB). Andersen et. al. provides a table with different parameters for σ and n in different environments (Andersen, Rappaport, & Yoshida 1995).

$$PL(d) = PL(d_0) + 10n_p \log(d/d_0) + \chi_\sigma \quad (1)$$

Another impact on power loss is from small scale fading, which has a significant impact over short distances (Andersen, Rappaport, & Yoshida 1995). By using mobile robots as wireless routers the robots can make small motions that improve the signal strength between them and their neighbors, exploiting constructive interference from small scale fading. Andersen shows that there is a difference of 20 dB in signal strength within a 1 meter span (Andersen, Rappaport, & Yoshida 1995). The exact form of our cost function is given in equation 2 where $dist$ represents the distance between x_i and x_j in their current positions d_i and d_j. ω represents random uniformly distributed white noise in the range of ± 10 dB.

$$f(d_i, d_j) = PL(dist) + \omega \quad (2)$$

Related Work

Related work has resulted in multiple variations of DPOP that reduce memory and bandwidth requirements. O-DPOP (Petcu & Faltings 2006) uses smaller messages than DPOP, but still may incur the same worst case memory requirements and in the worst case requires significantly more messages than DPOP and more time. Unlike O-DPOP, the memory usage of VMS ADOPT is bounded by an adjustable parameter. MB-DPOP (Petcu & Faltings 2007) also uses a parameter to control memory usage. In regions of the constraint graph that exceed this memory bound a brute force search is used to cycle through all assignments to variables in the region. Unlike MB-DPOP, VMS ADOPT uses a best-first search strategy to explore variable assignments asynchronously. This search strategy also allows VMS ADOPT to perform bounded error approximation.

Gerkey (Gerkey, Mailler, & Morisset 2006) proposes using sub-optimal distributed algorithms to optimize routes in a wireless network maintained by a set of mobile robots acting as communication relays. While this work is closely related to ours, there a number of key differences. First, we use an optimal algorithm that can use bounded-error approximation to find sub-optimal solutions while Gerkey uses algorithms with no performance guarantees. Second, we use a monotonic objective function which optimizes the aggregate link quality for every wireless link while Gerkey focuses on optimizing routes. A third key difference is how the cost of a wireless link is defined. Gerkey uses inverse distance as a link quality function. Our link quality function incorporates distance, small scale, and large scale fading. Due to the small changes in distance power loss from distance and large scale fading are negligible and the impact of small scale fading is significant.

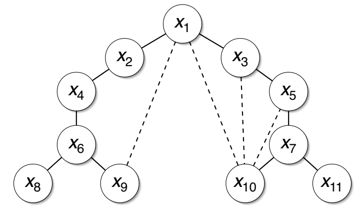

Figure 1: A DFS tree with 11 variables

Algorithm Description

Both ADOPT and DPOP make use of a Depth-First Search (DFS) tree or pseudo-tree arrangement described by Freuder and Quinn (Freuder & Quinn 1985) to determine what messages are passed between agents to find a solution to DCOPs. Figure 1 is an example of a DFS tree. There are a number of important relationships between nodes in the DFS tree which are used throughout the paper. Ancestors(x_i) represent the set of ancestors of x_i in the DFS tree including its parent. Descendants(x_i) represent the set of descendants in the DFS tree including children of x_i. Pseudo-Parents(x_i) are Ancestors(x_i) connected to x_i through back-edges in the DFS tree. Subtree-Pseudo-Parents(x_i) is defined as the union of all PseudoParents of descendants of x_i that are also ancestors of x_i.

Subtree-Pseudo-Parents impact the time and space complexity of ADOPT and DPOP. As the following subsections will show, VMS ADOPT exploits the trade-off between time and space complexity by how it handles subtree pseudo-parents. The rest of the paper assumes that the DFS tree exists when the algorithms are run. Distributed algorithms for creating DFS trees are presented in (Lynch 1996)

ADOPT

ADOPT is a backtracking search algorithm that is executed asynchronously and in parallel on every agent (Modi *et al.* 2006). ADOPT uses a best-first strategy that involves changing the assignment of a variable because it is no longer believed to lead to the optimal solution. ADOPT uses four types of messages, VALUE, COST, THRESHOLD, and TERMINATE. VALUE messages are sent from agents to descendants they share a constraint with. COST messages are sent to the agents parent. THRESHOLD and TERMINATE messages are sent to the agents children. COST, THRESHOLD, and TERMINATE messages contain a context field. A context represents an assignment of variables to values. Two contexts are compatible if they do not disagree about any variable assignment. The $CurrentContext$ represents the agents view of its ancestors variable assignments.

In ADOPT each agent maintains an $lb(d, x_k)$ and $ub(d, x_k)$ function when $d \in D_i$ and x_k is one of the agents children. From these functions the agents calculate a lower bound $LB(d)$ which represents the lowest cost solution the agent can produce given their $CurrentContext$. $UB(d)$ represents the highest cost solution the agent can produce with their $CurrentContext$. $LB(d)$ is calculated by summing over the agents $lb(d, x_k)$ function for all children and adding the agents local cost to the sum. $UB(d)$ is similarly calculated. At leaf agents $LB(d) = UB(d)$ always holds true, since these values do not depend on cost information coming from children agents.

An agents $lb(d, x_k)$ function is initialized to 0, and the $ub(d, x_k)$ is initialized to ∞. COST messages are propagated through the system, increasing $lb(d, x_k)$ and decreasing $ub(d, x_k)$. When $lb(d, x_k) = ub(d, x_k)$ that child has reported the actual cost of the solution at the subtree it is the root of. Both the $lb(d, x_k)$ and $ub(d, x_k)$ functions are valid only for the $CurrentContext$. When an agent's $CurrentContext$ changes, the values must be reset to their initialized value.

ADOPT uses the THRESHOLD message to quickly recalculate values that were reset. When a parent changes its value, the parents children have their $CurrentContext$ change and must reset lb and ub. However, the parent still knows the cost of the solution at the children. THRESHOLD messages let the children know the THRESHOLD the parent has stored for it and allow the children to avoid changing their variable assignment until they know their current assignment can not be under the threshold their parent sent them.

When the root node has found a solution it sends the TERMINATE message to its children. The children use the context in the message as their $CurrentContext$. They may need to continue to propagate VALUE and THRESHOLD messages and receive COST messages until they have found the best assignment for their variable. Then they send the TERMINATE message to their children.

The use of upper and lower bounds also allows ADOPT to perform bounded-error approximation. When at the root node, $UB(d) - LB(d) < bound$ and $LB(d) = min_{d' \in D_i} LB(d')$ the cost of the solution if the root node uses value d falls within $bound$ of optimal. In ADOPT setting the threshold at the root node to $min(LB + b, UB)$ where $LB = min_{d \in D_i} LB(d)$ and $UB = min_{d \in D_i} UB(d)$ results in finding a solution within the error bound b.

The largest messages in ADOPT contain context information and scale linearly with tree height. The worst case space complexity of ADOPT is $O(n^2)$. The running time, measured in number of cycles, and number of messages is exponential.

DPOP

DPOP is a dynamic programming algorithm for solving DCOPs (Petcu & Faltings 2005). The algorithm has two phases. In the first phase UTIL messages are propagated up the tree. In the second phase VALUE messages are propagated down the tree. Propagation of UTIL messages start at the leaf agents. A leaf agent sends its parent a UTIL message containing the cost of its best solution for every possible combination of variable assignments the agents pseudo-parents can have. The agents parent takes in UTIL messages from its children, combines them, adds in dimensions for its pseudo-parents not in the message, adds its cost information to the message and then eliminates itself from the message by selecting the best assignment for its variable for every combination of ancestor assignments in the UTIL message. This process continues until the UTIL message has reached the root agent.

When the root agent collects all the UTIL messages from its children it begins the VALUE propagation phase. The root node sends its children a VALUE message with a context containing the assignment of its variable. Its children then select their best variable assignment, add it to the context and send this to their children. This process continues until VALUE messages reach all the nodes. At this point all variables have an assignment.

The memory requirements of DPOP are bounded by a property of the DFS tree called induced width. The induced width of a DFS tree is defined as the maximum number of Subtree-Pseudo-Parents for all subtrees in the DFS tree. When there are no back edges in the DFS tree this value is one. In the worst case the induced width of a DFS tree is equal to n, the number of variables in the DFS tree. The memory requirements for DPOP are $O(|D|^w)$ where w represents the induced width. The largest messages in DPOP are UTIL messages which are $O(|D|^w)$. The number of messages required scale linearly with the number of agents. The number of message passing cycles scales linearly with the tree height.

Variable Message Size ADOPT

ADOPT and DPOP both have messages to send cost information to parent nodes. COST messages in ADOPT contain cost data only for the $CurrentContext$, while UTIL messages in DPOP contain cost data for all assignments of an agents Subtree-Pseudo-Parents. In VMS ADOPT we modify the COST message to incorporate multiple assignments of Subtree-Pseudo-Parents. This modification requires the DFS tree to be created and agents to receive the domain of their Subtree-Pseudo-Parents as pre-processing steps. A parameter M represents the maximum number of Subtree-Pseudo-Parents that have every possible assignment of their variables incorporated in COST messages. When M is equal to the induced width of the DFS tree, the COST message contains the same variables and assignments as UTIL messages in DPOP.

Additional variables are introduced to ADOPT to support this modification. m_i denotes the number of ancestor variables included in COST messages generated by agent A_i.

c_i represents a composite variable consisting of m_i ancestor variables. An ancestor variable x_a is a candidate to be included in c_i if $depth(x_i) - depth(x_a) \leq M + 1$ and $x_a \in$ Subtree-Pseudo-Parents(x_i) and adding the candidate would not result in $|c_i| > M$ and all the nodes between x_i and x_a in the DFS tree have been considered.

To help illustrate how variables are included in the composite variable we provide a few examples of c_i for agents in the DFS tree in Figure 1 with different values of M. When M equals 0 every agents composite variable is empty. When M equals 1 every agents composite variable includes the agents parent, except the root which has no parent. When M equals two: $c_7 =< x_5, x_3 >, c_{10} =< x_7, x_5 >, c_{11} =< x_7 >$. Notice in this case that c_{11} still only contains the parent of x_{11}. This is because x_{11} has no other Subtree-Pseudo-Parents. Also notice that c_7 includes x_3 even though x_7 and x_3 do not share a constraint. This is a result of the constraint x_{10} has with x_3. When M is set to the induced width of the DFS tree $c_i = SubtreePseudoParents(x_i)$. The composite variable c_i also has a composite domain J_i.

J_i represents all possible assignments of values for variables in c_i. j is used to donate a member of J_i. $CurrentContext$ contains agent A_i's view of the assignments of ancestors not contained in c_i. $CompositeContext$ represents agent A_i's view of the assignments of ancestors contained in c_i. $JointContext$ represents the union of $CurrentContext$ and $CompositeContext$.

An agent A_i calculates the values for cost, lower bounds, and upper bounds; using the following definitions:

- **Definition 1:** $\delta(d_i, j) = \sum_{(x_j,d_j)\in j} f_{ij}(d_i, d_j) + \sum_{(x_j,d_j)\in CurrentContext} f_{ij}(d_i, d_j)$ is the *local cost* at x_i, when x_i chooses d_i and c_i chooses j.
- **Definition 2:** $\forall j \in J_i, LB(d, j) = \delta(d, j) + \sum_{x_l\in Children} lb(d, j, x_l)$ is a *lower bound* for the subtree rooted at A_i, when A_i chooses d and c_i chooses solution j.
- **Definition 3:** $\forall j \in J_i, UB(d, j) = \delta(d, j) + \sum_{x_l\in Children} ub(d, j, x_l)$ is a *upper bound* for the subtree rooted at A_i, when A_i chooses d and c_i chooses solution j.
- **Definition 4:** $\forall j \in J_i, LB(j) = \delta(d_i, j) + \sum_{x_l\in Children} lb(d_i, j, x_l)$ is a *lower bound* for the subtree rooted at A_i, when c_i chooses solution j.
- **Definition 5:** $\forall j \in J_i, UB(j) = \delta(d_i, j) + \sum_{x_l\in Children} ub(d_i, j, x_l)$ is a *upper bound* for the subtree rooted at A_i, when c_i chooses solution j.
- **Definition 6:** $LB = \min_{d\in D_i} LB(d, CompositeContext)$ is a *lower bound* for the subtree rooted at A_i with the current $JointContext$.
- **Definition 7:** $UB = \min_{d\in D_i} UB(d, CompositeContext)$ is an *upper bound* for the subtree rooted at A_i with the current $JointContext$.

Algorithm 1 Received (**VALUE**, (x_j,d_j))

```
if TERMINATE not received from parent then
   if x_j ∈ c_i then
      add (x_j,d_j) to CompositeContext;
   else
      add (x_j,d_j) to CurrentContext;
   end if
   for all d ∈ D_i, j ∈ J_i, x_l ∈ Children do
      if context(d, j, x_l) incompatible with CurrentContext then
         lb(d, j, x_l) ← 0; t(d, j, x_l) ← 0;
         ub(d, j, x_l) ← ∞; context(d, j, x_l) ← {};
      end if
   end for
   MaintainThresholdInvariant; Backtrack;
end if
```

Algorithms 1 through 3 describe the different operations VMS ADOPT performs. It shares the same methods as ADOPT, however the methods are modified to support larger cost messages and the composite variable c_i. The methods for handling terminate, maintaining the allocation invariant, maintaining the child threshold invariant, maintain threshold invariant, and receiving threshold messages are omitted. The only difference between these methods and ADOPT is the use of $JointContext$ in place of $CurrentContext$. The initialize method is also omitted, the only difference from ADOPT is that the initialized functions include all values of j contained in J_i. The process for performing bounded-error approximation is also the same as in ADOPT.

The memory usage of VMS ADOPT is $O(n^2 |D|^{M+1})$. The largest message is now the COST message which is $O(n^2 |D|^M)$. When M is set to the max value the number of message cycles required is linear. The maximum value of M is equal to the induced width of the DFS tree.

Experiments

Pseudo-randomly generated networks were used to evaluate how VMS ADOPT works with a diverse set of network topologies. Experiments were performed to evaluate how the behavior of VMS ADOPT changes as the value of M changes and the problem size grows. Experiments were also performed to evaluate how changing M impacts bounded error approximation. Multiple trials were run for both cases.

For every trial a DCOP problem was created with each node represented by an agent with one variable. The domain of the variable represented the discrete positions the agent could occupy. Equation 2 represents our cost function, with large-scale fading used to represent variations in powerloss over larger distances and small-scale fading used to represent variations in powerloss over distances on the order of wavelengths.

Execution of each agent occurred in a synchronous manner to permit collecting data on the number of execution cycles each agent ran for. In each cycle an agent could receive messages and send messages. The backtrack function was called once per cycle, after all messages were received. Messages sent in one cycle were received in the next cycle.

Algorithm 2 Received (**COST**, x_k, $costs$, $jContext$)

```
for all (x_j,d_j) ∈ jContext do
  if x_j ∈ c_i and x_j is not my neighbor then
    add(x_j,d_j) to CompositeContext
  end if
end for
for all (context, lb, ub) ∈ costs do
  d ← value of x_i in context;
  remove (x_i,d) from context;
  s ← {};
  for all (x_j,d_j) ∈ context do
    if x_j ∈ c_i then
      remove (x_j,d_j) from context;
      add (x_j,d_j) to s;
    end if
  end for
  if TERMINATE not received from parent then
    for all (x_j,d_j) ∈ context and x_j is not my neighbor do
      add (x_j,d_j) to CurrentContext;
    end for
    for all d' ∈ D_i, j ∈ J_i, x_l ∈ Children do
      if context(d', j, x_l) incompatible with CurrentContext then
        lb(d', j, x_l) ← 0; t(d', j, x_l) ← 0;
        ub(d', j, x_l) ← ∞; context(d', j, x_l) ← {};
      end if
    end for
  end if
  for all j ∈ J_i do
    if s compatible with j and context compatible with CurrentContext then
      lb(d, j, x_k) ← lb; ub(d, j, x_k) ← ub;
      context(d, j, x_k) ← context;
    end if
  end for
end for
MaintainChildThresholdInvariant;
MaintainThresholdInvariant; Backtrack;
```

Execution cycles continued to occur until all agents terminated.

Results were collected using platform independent performance metrics: number of execution cycles, number of messages, number of constraint checks, and aggregate message size. Execution cycles relate to the amount of time optimization takes. Messages relate to the amount of information transmitted through the network. Constraint checks relate to the amount of processing the algorithm requires. The aggregate message size was calculated by having each agent report the largest message sent at the end of execution and summing up this value for every agent. Aggregate message size represents the amount of bandwidth required by the system.

Algorithm 3 Backtrack

```
if threshold == UB then
  d_i ← d that minimizes UB(d, CompositeContext);
  MaintainChildThresholdInvariant;
else if LB(d_i, CompositeContext) > threshold then
  d_i ← d that minimizes LB(d, CompositeContext);
  MaintainChildThresholdInvariant;
end if
SEND (VALUE, (x_i, d_i)) to each lower priority neighbor;
MaintainAllocationInvariant;
if threshold == UB then
  if TERMINATE received from parent or x_i is root then
    SEND (TERMINATE, JointContext ∪ {(x_i, d_i)}) to each child;
    Terminate execution;
  end if
end if
costs ← {};
for all j ∈ J_i do
  add (j ∪ CurrentContext, LB(j), UB(j)) to costs;
end for
SEND (COST, x_i, costs, JointContext) to parent;
```

Random Network Experiment

The random network experiments were performed to evaluate the performance of VMS ADOPT with diverse network setups. The structure of these networks was pseudo-randomly created using the following procedure. The first node is randomly placed and than each following node is placed within communications range of the previously placed nodes while keeping a minimum separation of 17.25 meters between every node. A wireless link is created between every pair of nodes in communications range of each other. The communications range was set to 40 meters. The ratio between the minimum separation and communications range match parameters in (Gerkey, Mailler, & Morisset 2006). Equation 2 describes the cost function used with n_p equal to 2.4, σ set to 9.6 for and having small scale fading adding ± 10 dB of noise. The parameters for the power loss model and small scale fading are based on results gathered from indoor office environments described in (Andersen, Rappaport, & Yoshida 1995).

Figure 2 shows the average results from 9 trials for the random network experiments with 2 to 10 nodes with values of M in the range of 0 through 5. Increasing M results in a clear decrease in the number of cycles and messages and an increase in message size. The results for number of constraint checks are less clear and additional experiments we have performed suggest that the structure of the DFS tree impacts how this metric performs for different values of M.

Bounded-Error Experiment

Experiments using bounded-error approximation were performed on random networks with 100 nodes to explore if increasing M would result in lower cost approximate solutions. Thirty trials were run with 100 robots and M values

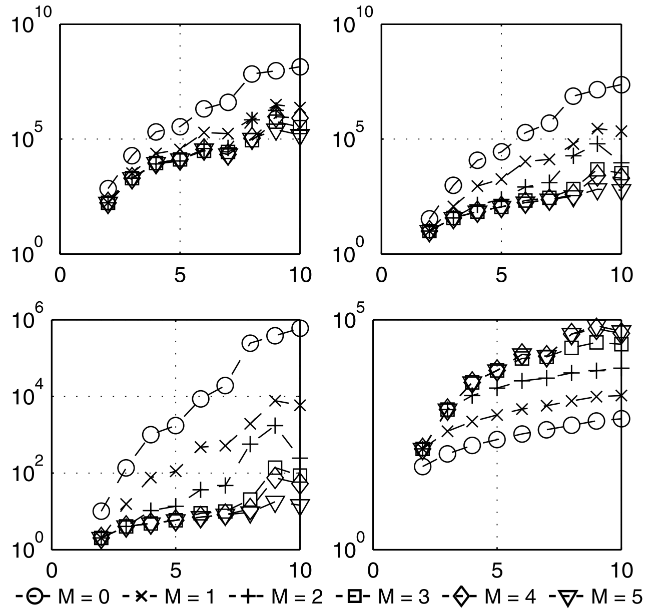

Figure 2: Constraint checks (upper left), Messages (upper right), Cycles (lower left), and Message Size in bytes (lower right) plotted against number of agents for Random networks with optimal solutions

of 0, 1, 2, and 3. The error bound was set to $10,000$. Each run was capped to executing for 10 hours, if this limit was reached the variable assignments agents had selected at that point was used to calculate solution quality. Figure 3 shows our results for cycles and relative quality plotted against different values of M. The error bars represent standard deviation. Relative quality is calculated for each problem instance by dividing the quality found by VMS ADOPT by the quality found by VMS ADOPT for that problem instance when M equals 0. Increasing M does improve the quality of solutions found using bounded error approximation. Unlike the experiments without bounded-error approximation the number of cycles does not consistently decrease as M is increased.

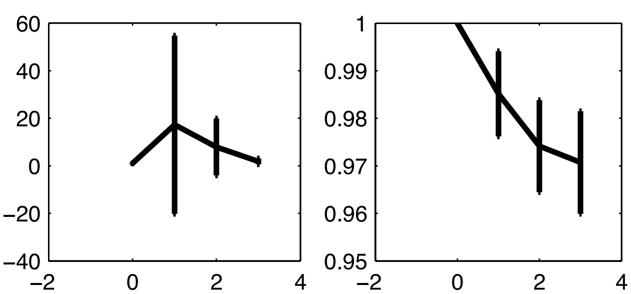

Figure 3: Relative number of cycles (left) and solution cost (right) plotted against values of M for bounded-error experiments

Discussion and Future Work

The combination of DCOPs, wireless network optimization and robotics is full of a number of interesting possibilities. The requirements introduced by interacting with real hardware platforms while trying to perform optimization with performance guarantees brings up a number of new challenges and complex problems to solve. VMS ADOPT partially addresses these by taking advantage of the decreased number of cycles and messages required by DPOP without having to support an exponential amount of memory. Support for bounded error approximation in VMS ADOPT also allows the algorithm to run on large problems.

Dealing with unknown constraint graphs, which define the relationship between robots, their positions, and signal strengths is an area to investigate. Currently we assume this is known a prior and constant, however in the real world this data will have to be collected and is time-varying. In addition, some degree of synchronization is required to collect this data since robots do not change their position instantly and data collected about a link while one of the nodes is moving will be inaccurate. We are currently investigating extensions to VMS ADOPT that deal with these situations.

References

Andersen, J.; Rappaport, T.; and Yoshida, S. 1995. Propagation measurements and models for wireless communications channels. *IEEE Communications Magazine* 33(1):42–49.

Freuder, E., and Quinn, M. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the International Joint Conference of AI*, 1076–1078.

Gerkey, B. P.; Mailler, R.; and Morisset, B. 2006. Commbots: Distributed control of mobile communication relays. In *Proc. of the AAAI Workshop on Auction Mechanisms for Robot Coordination*, 51–57.

Lynch, N. 1996. *Distributed Algorithms*. Morgan Kaufmann.

Modi, P.; Shen, W.; Tambe, M.; and Yokoo, M. 2006. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161(1-2):149–180.

Petcu, A., and Faltings, B. 2005. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *IJCAI 05*, 266–271.

Petcu, A., and Faltings, B. 2006. O-DPOP: An algorithm for Open/Distributed Constraint Optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-06*, 703–708.

Petcu, A., and Faltings, B. 2007. MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*.

Yokoo, M.; Durfee, E.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *Knowledge and Data Engineering, IEEE Transactions on* 10(5):673–685.