From: AAAI Technical Report WS-02-04. Compilation copyright © 2002, AAAI (www.aaai.org). All rights reserved.

Distributed Optimization for Overconstrained Problems and its Application

Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe
University of Southern California/Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{modi,shen,tambe}@isi.edu

Abstract

Distributed optimization requires the optimization of a global objective function that is distributed among a set of autonomous, communicating agents and is unknown by any individual agent. One major mode of attack has been to treat optimization as an overconstrained distributed satisfaction problem and exploit existing distributed constraint satisfaction techniques. These approaches require incremental searches with periodic synchronization operations and lack any guarantees of optimality. This paper presents *Adopt*, an asynchronous, distributed algorithm for overconstrained settings. The fundamental ideas in Adopt are to represent constraints as discrete functions (or valuations) — instead of binary good/nogood values — and to use the evaluation of these constraints to measure progress towards optimality. In addition, Adopt uses a sound and complete partial solution combination method to allow non-sequential, asychronous computation. Finally, Adopt is not only provably optimal when given enough time, but allows solution time/quality tradeoffs when time is limited. We apply Adopt to a real-world overconstrained distributed resource allocation problem and we present empirical results comparing Adopt to previous approaches.

Introduction

Distributed Optimization (DOP) is to optimize a global objective function that is intrinsically distributed among many agents who are autonomous and physically separated in space and/or time. DOP is different from parallel computing in the sense that the distribution of the objective function is mandated by the nature of the problem, not artificially imposed or manipulated for reasons of computational efficiency or parallel processing.

The combination of intrinsic distribution and the desire for optimality in DOP poses four very challenging technical questions: 1) How is distributed information represented so that it complies with the application domain yet allows global progress to be measured? 2) How is global progress measured when individual agents have incomplete information? 3) How do agents integrate partial solutions so that they correctly judge the quality of a global solution? and 4) How do agents manage the time-to-solution/solution-quality tradeoff when time is limited?

Throughout the history of computer science, progress has been made towards solving these challenging problems. For example, scheduling techniques have been modified and extended to meet the requirement of distribution and optimality (Liu & Sycara 1995). Seminal work has been done by Yokoo et al. in developing algorithms for distributed constraint satisfaction problems (DCSP)(Yokoo *et al.* 1998). Distributed partial constraint satisfaction problem exploit these algorithms to approximate and solve DOP (Hirayama & Yokoo 1997). Most recently, attempts have been made to use constraint satisfaction techniques to search for optimal solutions by incrementally altering the complexity of the problem using thresholds and additional domain knowledge (Hirayama & Yokoo 2000), but these approaches do not guarantee optimality.

In this paper, we present a new DOP technique, called Adopt (Asynchronous Distributed Optimization). Adopt is based on four major ideas that address the questions presented earlier. (1) Adopt represents and measures constraints not by binary "good/nogood" satisfaction, but by degrees of valuation, which is a significant generalization of distributed constraint representation; (2) Adopt abandons the idea of incremental thresholds (Hirayama & Yokoo 2000), and instead, by exploiting its generalized representation, Adopt uses the evaluation of constraints as the natural yardsticks for progress towards optimality; (3) Different from previous approaches that have similar ideas, such as (Hirayama & Yokoo 1997), Adopt also uses a non-sequential asynchronous update technique for integrating partial solutions and thus avoids unnecessary idle states for agents; (4) Finally, in cases where finding the optimal solution is prohibitively expensive, Adopt incorporates a local cost tolerance that allows tradeoffs between solution quality and time to solution. This paper also provides a systematic mapping technique for representing a given application as a DOP problem. Such a mapping is successfully applied to the domain of distributed resource allocation and an implemented real-world application of distributed sensor networks.

Distributed Resource Allocation

A general distributed resource allocation problem consists of a set of agents that can each perform some set of operations and a set of tasks to be completed. In order to be completed, a task requires some subset of agents to perform the necessary operations. Thus, we can define tasks by the operations that agents must perform in order to complete them. The problem to be solved is an allocation of operations to tasks

such that all tasks are performed. More formally, a Distributed Resource Allocation Problem is a structure $<\mathcal{A}g, \mathcal{O}, \mathcal{T}, w>$ where

- $\mathcal{A}g$ is a set of agents, $\mathcal{A}g = \{A_1, A_2, ..., A_n\}$.
- $\mathcal{O} = \{O_1^1, O_2^1, ..., O_p^i, ..., O_q^n\}$ is a set of operations, where operation O_p^i denotes the p'th operation of agent A_i. An agent can only perform one operation at a time.
- $\mathcal{T}$ is a set of tasks, where a task is a collection of sets of operations. Let T be a task in $\mathcal{T}$ ($T \subseteq$ power set of $\mathcal{O}$). $t_r \in T$ is a set of operations called a *minimal set* because it represents the minimal resources necessary to complete the task. There may be alternative minimal sets that can be used to complete a given task. Minimal sets from two different tasks *conflict* if they contain operations belonging to the same agent.
- w: $\mathcal{T} \rightarrow N \cup \infty$ is a *weight function* that quantifies the cost of not completing a given task.

A *solution* to a resource allocation problem involves choosing minimal sets for tasks such that the chosen minimal sets do not conflict. However, conflict may be unavoidable when there are too many tasks and not enough resources. In such cases, the agents must allocate resources only to the most important tasks. More formally, we wish to find a $\mathcal{T}_{ignore} \subseteq \mathcal{T}$ such that $\sum_{T \in \mathcal{T}_{ignore}} w(T)$ is minimized and there are enough agents to complete all tasks in $\mathcal{T} \setminus \mathcal{T}_{ignore}$. We state the complexity of this problem as a theorem.

Theorem 1: A distributed resource allocation problem given by $<\mathcal{A}g, \mathcal{O}, \mathcal{T}, w>$ is NP-Complete.

A concrete instantiation of the above resource allocation problem is the following distributed sensor network domain. It consists of multiple fixed sensors, each controlled by an autonomous agent, and multiple targets moving through their sensing range. Each sensor is equipped with three radar heads, each covering 120 degrees. Resource contention may occure because an agent may activate at most one radar head, or sector, at a given time. Three sensors must turn on overlapping sectors to accurately track a target. For example in Figure 1 (left), when agent 1 detects a target in its sector 0, denoted as *operation* O_0^1, it must coordinate with neighboring agents so that they activate their respective sectors that overlap with agent 1's sector 0. Targets in a particular region are *tasks* that need to be completed/tracked and a choice of three sensors to track a target corresponds to a *minimal set*, e.g. $\{O_0^1, O_2^2, O_1^4\}$ is a minimal set for Target 1.

Figure 1(right) shows a configuration of 9 agents and an example of resource contention. Since at least three neighboring agents are required to track each target and no agent can track more than one, only two of the four targets can be tracked. The agents must find an allocation that minimizes the weight of the ignored targets. In addition, when the targets are moving quickly and time is limited, we may be willing to give up optimality for a fast solution.

Distributed Optimization Problem

A Distributed Optimization Problem 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 finite, discrete

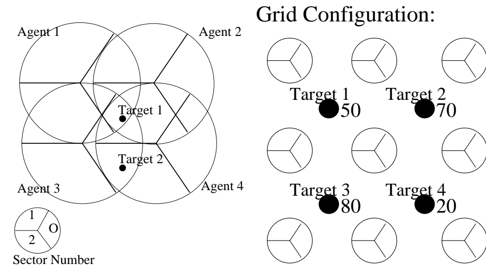

Figure 1: Sensor sector schematic(left) and a grid layout configuration with weighted targets (right)

domains $D_1, D_2, ..., D_n$, respectively. Only the agent who is assigned a variable has control of its value and knowledge of its domain. The objective is to choose values for variables such that some criterion function over all possible assignments is at an extremum. In general optimization problems, one can imagine any arbitrarily complex criterion function. In this paper, we restrict ourselves to functions that can be decomposed into the sum of a set of binary (and/or unary) functions. Thus, for each pair of variables x_i, x_j, we are given a *cost function* $f_{ij} : D_i \times D_j \rightarrow N \cup \infty$. Intuitively, one can think of the cost function as quantifing the degree to which a particular assignment of values to a pair of variables is "deficient", or less than optimal. The objective is to find a complete assignment $\mathcal{A}^*$ of values to variables such that the total deficiency is minimized. (An assignment is *complete* if all variables in V are assigned some value.) More formally, let $\mathcal{C} = \{\mathcal{A} \mid \mathcal{A}$ is a complete assignment of values to variables in $V\}$. We wish to find $\mathcal{A}^*$ such that $\mathcal{A}^* = \arg\min_{\mathcal{A} \in \mathcal{C}} F(\mathcal{A})$, where

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

This change in representation, where constraints have continuous values, as opposed to binary (satisfied/not-satisfied values), is a major shift from previous approaches to addressing overconstrained problems. Indeed, ours is a strict generalization of the standard representation of the Distributed Constraint Satisfaction Problem from (Yokoo *et al.* 1998). For example, a "hard" constraint is modelled as having infinite cost for all pairs of variable values that violate the constraint and zero cost otherwise.

Solving Resource Allocation Problems

We illustrate a general methodology for mapping the resource allocation problems (e.g., the distributed sensor nets) into the distributed optimization problem.

- **Variables:** $\forall T_r \in \mathcal{T}, \forall O_p^i \in \bigcup_{t_r \in T_r} t_r$, create a DOP variable $T_{r,i}$. The value of this variable is controlled by agent A_i.
- **Domain:** For each variable $T_{r,i}$, create a value $t_{r,i}$ for each minimal set in T_r, plus a "I" value (ignore). The I value does not conflict with any minimal set and thus when resources are limited, it allows agents to avoid assigning resources to less importance tasks.

We define two constraints on variables (tasks) that specify a valid allocation of resources. The first constraint prevents agents from assigning conflicting minimal sets to tasks and the second requires agents to agree on allocations.

$$\forall r,s,i : f(T_{r,i},T_{s,i}) = \begin{cases} \infty & if\ the\ minimal\ sets \\ & conflict, \\ 0 & otherwise \end{cases}$$

$$\forall r,i,j : f(T_{r,i},T_{r,j}) = \begin{cases} \infty & if\ T_s, T_r\ have \\ & unequal\ values, \\ 0 & otherwise \end{cases}$$

Finally, a third constraint requires agents to pay a cost equal to the weight of the task whenever it is ignored:

$$\forall r,i : f(T_{r,i}) = \begin{cases} w(T_r) & if\ T_r\ has \\ & value\ "I", \\ 0 & otherwise \end{cases}$$

In fact, the first two constraints alone already define a satisfaction problem, but the third one makes it a DOP.

The Adopt Algorithm

Preliminaries

A set of variable/value pairs specifing a (possibly incomplete) assignment is called a *view*).

- **Definition:** A *view* is a set of pairs of the form $\{(x_i,d_i), (x_j, d_j)...\}$. A variable can appear in a view no more than once. Two views are *compatible* if they do not disagree on any variable assignment and a view is *larger* than another if it contains more variables.

The deficiency of a value of a variable in a view is determined by the sum of its cost functions.

- **Definition:** The *local deficiency* of a given view vw wrt variable x_i is defined as

$$\delta(x_i, vw) = \sum_{x_j \in V} f_{ij}(d_i, d_j) \quad , where\ x_i = d_i, \\ x_j = d_j\ in\ vw$$

The Adopt algorithm requires variables to have a fixed tree structured priority ordering. Any such ordering is sufficient and lexicographic ordering is the simplest method. Figure 3.a shows an example constraint graph and an associated priority order. Two agents with variables x_i, x_j are *connected* if their cost function f_{ij} is not a constant. The tree ordering can be formed in a preprocessing step, or alternatively, can be discovered during algorithm execution. For simplicity of description of the algorithm, we will assume the tree is already formed in a preprocessing step. Figure 3.b shows the search tree formed from the constraint graph in Figure 3.a.

Adopt

Adopt is a provably optimal, asynchronous distributed optimization algorithm based on four major ideas. First, it operates on a generalized representation, described in the previous section, where constraints have degrees of quality (cost) of variable assignment rather than simple binary good/nogood values. Second, Adopt uses the evaluation of these constraints as a solution quality metric and to measure progress towards the optimal solution. Third, Adopt performs a distributed branch-and-bound search, asynchronously combining partial solutions into larger solutions as non-local information is received from other agents. Finally, in cases where finding the optimal solution is prohibitively expensive, Adopt incorporates a local cost tolerance that allows faster algorithm termination. Adopt's four ideas contrast with the leading approach to overconstrained DCSP (Hirayama & Yokoo 2000). First, they continue to rely on a satisfaction based approach to value assignment and thus they use incremental thresholds to progress towards better solutions, *which does not guarantee optimality*. Also in their approach, complexity is limited by using thresholding as a global rigid cutoff, i.e. all agents eliminate any constraints that are below the threshold value from consideration before search begins. This may a priori eliminate constraints that could have been easily satisfied without backtracking. Finally, the approach in (Hirayama & Yokoo 1997) uses sequential, synchronous searches to build root to leaf paths which fail to exploit the parallelism in distributed computation (Hirayama & Yokoo 1997),

Procedures from the Adopt algorithm are shown in Figure 2. x_i represents the agent's local variable and d_i represents its current value. The algorithm begins by each agent instantiating its variable concurrently and sending this value to all its connected lower priority agents via a VALUE mes-

Initialize: $Currentvw \leftarrow \{\}$;$d_i \leftarrow$ null;
  $\forall x_l \in Children$:
    $c(x_l, \{\}) \leftarrow 0$;
    $Views(x_l) \leftarrow \{\}$;
  hill_climb;
when received (**VALUE**, (x_j, d_j))
  add (x_j, d_j) to $Currentvw$;
  hill_climb;
when received (**VIEW**, x_l, vw, $cost$)
  add vw to $Views(x_l)$;
  $c(x_l, vw) \leftarrow cost$;
  hill_climb;
procedure hill_climb
  $\forall d \in D_i$:
    $e(d) \leftarrow \delta(x_i, Currentvw \cup \{(x_i, d)\})$;
    $\forall x_l \in Children$:
      $vw \leftarrow$ largest view in $Views(x_l)$ compatible with $Currentvw \cup \{(x_i, d)\}$);
      $e(d) \leftarrow e(d) + c(x_l, vw)$;
  choose d that minimizes $e(d)$; **— (ii)**
  if $d_i \neq d$ then
    $d_i \leftarrow d$;
    SEND (**VALUE**, (x_i, d_i)) to all connected lower variables;
  end if;
  choose d_h where $(parent, d_h) \in Currentvw$;
  if $e(d_i)$ greater than τ then **— (iii)**
    SEND (**VIEW**, x_i, $Currentvw$, $e(d_i)$) to parent;

Figure 2: Procedures from the Adopt algorithm

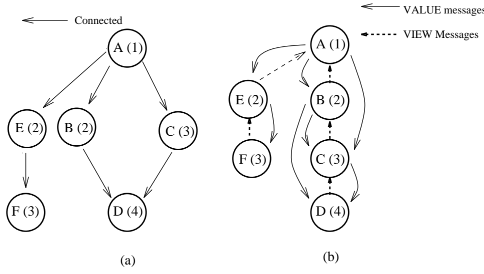

Figure 3: (a) Constraints between agents with priority order in parentheses. (b) Flow of VALUE and VIEW messages between agents.

sage. After this, agents asynchronously wait for and respond to incoming messages. Lower priority agents choose values that have the least deficiency given the current values of higher priority agents stored in the $CurrentView$ variable. This often leads to quick, possibly suboptimal solutions. In order to escape local minima, lower priority agents report feedback to higher priority agents. When a lower priority agent evaluates its local cost functions and realizes the system is incurring cost greater than *tolerance level* τ, shown in Line (iii) of Figure 2, it constructs a VIEW message which contains its current view of the higher priority agents' assignments and the associated amount of cost. It sends this VIEW message only to the lowest higher priority connected agent (its parent). As an agent receives VIEW messages, it maintains the set of views and associated costs reported to it from its children. Then, the agent will either abandon its current variable value in favor of one with less total cost (Line (ii)) or pass a VIEW message up to *its* lowest higher priority agent. Figure 3.b shows the flow of VALUE and VIEW messages between agents as the algorithm executes. An important property of this algorithm is that an agent only stores variable/value information about connected variables, rather than building a complete path from root to leaf.

When time is limited, we can increase the value of the τ parameter, which in turn allows agents to ignore smaller costs by not reporting to higher priority agents. This prevents higher priority agents from switching their values, thus allowing the system to reach a stable state more quickly. A key property of this local tolerance is that agents still attempt to find values that minimize costs, as long as it can be done locally without backtracking. In general, the τ parameter would need to be engineered using domain knowledge but our point is that Adopt is "tune-able" in this way.

Algorithm correctness

We show that if Adopt reaches a stable state (all agents are waiting for incoming messages), then the complete assignment chosen by the agents is equal to the optimal assignment. We first state Lemma 1 which states that an agent never overestimates cost, and Lemma 2 which says that in a stable state, an agent's estimate of cost is equal to the true cost.

Let $C(x_i, vw)$ denote the local cost at x_i plus the cost of the optimal assignment to descendents of x_i, given that x_i and its ancestors have values fixed to those given in vw.

Lemma 1: An agents estimate of the cost of a solution is never greater than the actual cost. $\forall x_i \in V, \forall d \in D_i$,

$$e(d) \leq C(x_i, Currentvw \cup \{(x_i, d)\})$$

Lemma 2: Assume Adopt is in a stable state. $\forall x_i \in V$, if d_i is x_i's current value, then,

$$e(d_i) = C(x_i, Currentvw \cup \{(x_i, d_i)\})$$

Theorem 1: Assume Adopt is in a stable state. $\forall x_i \in V$, if d_i is the value of x_i, then $(x_i, d_i) \in \mathcal{A}^*$.

proof: Lemma 2 states that an agent's estimate of cost for its final choice is equal to the minimum cost possible given what it and higher priority agents have chosen, and Lemma 1 says that its estimate for the rest of its choices are a lower bound on the minimum cost for that choice. Each agent chooses the value that minimizes its estimate given what higher priority agents have chosen. Clearly, the highest priority agent chooses optimally and thus by induction, the entire system chooses optimally. □

Finally, we have left to show that the algorithm does indeed reach a stable state in which all agents are waiting for incoming messages. We first state Lemma 3 which states that an agents estimate of cost never decreases.

Lemma 3: For all $d \in D_i$, for all $x_l \in Children$, the value of $c(x_l, vw)$ is non-decreasing over time.

Theorem 2: Adopt will reach a stable state.

proof: x_i sends a VALUE message only in reponse to the receipt of a VIEW message that changes cost $c(x_l, vw)$. By Lemma 3, $c(x_l, vw)$ is non-decreasing and by Lemma 1, has an upper bound. So, eventually $c(x_l, vw)$ must stop changing and x_i will stop sending VALUE messages. VIEW messages are only sent to a higher priority agent, and the highest priority agent never sends VIEW messages. So eventually, agents will stop sending VIEW messages. Thus, the system reaches a stable state. □.

Evaluation

In this section, we present the empirical results from five experiments using Adopt to solve an overconstrained distributed resource allocation problem in a sensor domain. These are not toy examples but rather real problems from hardware sensor configurations similar to that shown in Figure 1(right). To measure progress over time in a system that is completely asynchoronous and distributed, previous methods of counting "synchronous cycles" were not realistic and acceptable. Instead, we measure the number of search cycles by counting the maximum number of VIEW messages sent by an agent in the system.

Hypothesis 1: Adopt scales well with increasing number of agents but number of interacting tasks constant. In terms of the constraint graph, adding more agents corresponds to adding new variables and associated new constraints. However, keeping the number of tasks constant means that choosing zero-cost assignments for the new constraints is easy. Few cycles should be required between agents with these types of variables. We experiment with

two configurations, chain and grid, and keep the number of interacting tasks constant at one, two, three or four. Figure 4 plots the results with number of agents on the x-axis and number of cycles to optimal solution on the y-axis. We see that our hypothesis is verified. This means that the number of agents, by itself, does not adversely affect time to solution. This is a desirable property of our approach since despite a large number of agents (100s), we may sometimes have only a few interacting tasks.

Hypothesis 2: Adopt scales exponentially as number of interacting tasks increases. As the number of interacting tasks increases, the constraint graph become harder to optimize. Thus, a larger number of cycles are necessary to reach the optimal solution because more search in necessary. Figure 5 (left) shows results for two sensor configurations. The chain configuration has 28 variables with domain size 5 and 94 constraints and the grid configuration has 36 variables with domain size 5 and 154 constraints. In both configurations, the variables have a linear (total) priority order. The graph shows that the number of cycles (y-axis) increases exponentially as more interacting tasks (x-axis) are added.

Hypothesis 3: Time to solution decreases if priority ordering of agents reduces the total distance from the highest to lowest priority agent. To mitigate the exponential increase in time to solution when the number of interacting tasks increases, we minimize the total distance from the highest to lowest priority agent. We conjecture that if feedback reaches higher priority agents quicker, perhaps the exponential increase effect will not be as severe. Figure 5 (right) shows the same two configurations as in Experiment 2, but with a tree-structured (partial) priority order. The results show that a tree structured priority ordering is superior to a linear ordering in terms of time to solution and thus supports our hypothesis. How the agents can autonomously determine the best priority ordering quickly is a potentially valuable issue for future work.

Hypothesis 4: Increasing tolerance level τ allows agents to tradeoff time-to-solution for solution-quality. We steadily increase τ and measure the time to solution (cycles) and the quality of solution obtained. Quality of solution is measured as the percentage of best solution possible, so 100% corresponds to the optimal solution. Figure 6 shows the results for four types of configurations. The left graph shows the number of cycles required to reach a stable state for each configuration, while the right graph denotes the quality of the solution obtained for each configuration. We can see that the τ parameter allows agents to trade-off computation time for solution quality. As τ is increased, the time to solution falls as the solution quality also degrades. Only in the most severe cases where τ is set very high does Adopt begin to settle upon sub-optimal solutions, and this drop off is abrupt. This suggests that it is possible to engineer the τ parameter to obtain high gains in time to solution without losing much in solution quality. A key surprise was the non-monotonic decrease in time to solution as τ is increased in the left graph. This suggests that very small cost tolerance is actually worse than no tolerance at all.

Hypothesis 5: Adopt is more robust to variance in constraint valuations than incremental thresholding methods. We are interested in how the valuations of constraints in the problem effect the performance of different constraint optimization approaches. Approaches that rely on iterative thresholds require the constraints to be ordered by their valuation and the way in which the constraints are ordered may affect the results dramatically.

In order to compare Adopt with satisfaction approaches that use incremental thresholds, we experiment with two iterative thresholding schemes. In the Removing scheme, a search for a satisfactory solution is attempted with the entire problem. If it is found to be unsolvable, constraints below some threshold are iteratively removed from the problem until a solvable subproblem is found. The Adding scheme is the opposite approach beginning with an empty solvable problem. A crucial point of comparison is the number and cost of synchronizations required. In the Removing scheme, if no solution can be found at the current threshold t, agents must synchronize and somehow decide to drop further constraints at some $t + \epsilon$ level. Correctly determining this ϵ is a major difficulty of incremental thresholding methods. To be conservative for these experiments, we assume no synchronization cost and ϵ equals one fifth of the highest weighted constraint. If ϵ is set higher, the problem space search becomes very coarse and solution quality is poor. If ϵ is too low, many iterations are required to find a solution and thus time to solution is adversely affected.

We experiment with three problem classes: **equal** represents problems in which tasks all have the same weight, **random** represents problems in which tasks are randomly assigned weights in the range [10,100] and in **bipolar** problems the set of tasks have either weight 10 or 100 and furthermore, if all tasks of weight 10 are ignored, the rest of the tasks can be performed. Table 1 compares the time to solution and quality of solution for the three different approaches. We average the results of four different problems in each problem class, so a 100% in the column **%opt** means that the algorithm found the optimal solution in all four problems in the given class, while 0% means the algorithm found no solution for all four problems. The results show that performance of the two iterative approaches depends heavily on the ordering of the given constraints, while Adopt is able to perform well over a range of different input structures.

Finally, Adopt has been used to track actual targets in a real hardware sensor setup. In the largest scale experiment to date, 8 sensors/agents were set up in a 60x60 ft area with two targets moving through the sensing range at approx 1/2 ft/sec. Each hardware sensor was driven by a Pentium PC and agent communication was done via TCP/IP. Initial results show that agents were able to successfully produce tracks of targets despite overconstrained situations.

Summary and Related Work

Distributed optimization arises in many domains where information and control is distributed among autonomous communicating agents. We have presented the Adopt algorithm, a provably optimal asynchronous distributed algorithm for distributed optimization. Adopt represents a new approach and a departure from previous methods in that it i)

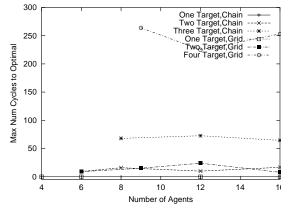

Figure 4: Number of cycles with increasing number of agents and constant number of tasks

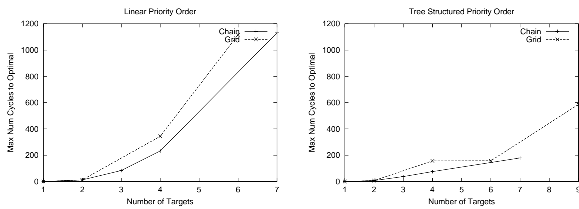

Figure 5: Number of cycles with increasing number of interacting tasks for a linear priority order (left) and tree structured priority order (right)

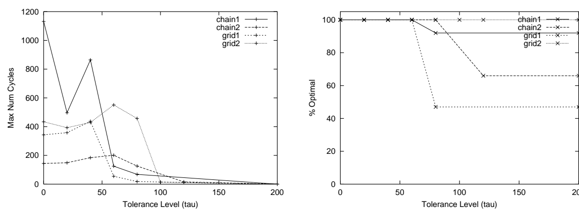

Figure 6: Effect of increasing tolerance level on time to solution(left) and solution quality (right)

Table 1: Adopt (τ = 60) vs. Iterative Search

	Adopt+τ		Removing		Adding	
Prob	Cycles	%Opt	Cycles	%Opt	Cycles	%Opt
equal	294	87%	165	0%	37	0%
rand	70	100%	118	83%	54	83%
bipolar	31	100%	41	93%	41	93%

operates on a valued constraint representation, ii) uses constraint evaluation to measure progress towards the optimal solution, iii) uses a novel procedure for asynchronously integrating partial solutions into larger solutions, iv) includes a novel method for trading off solution quality for time to solution. We show empirical results illustrating Adopts performance on a distributed sensor network resource allocation problem.

Other approaches to distributed optimization include market based systems (Walsh & Wellman 1998), which offer an promising, alternative path of investigation. We view these methods as complementary to constraint-based approaches and hybrid approaches may yield significant advantages over either alone. Others approaches have used a "coordinator" agent technique, whereby a special agent collects information about variables and domains from other agents and finds a solution (Lemaitre & Verfaillie 1997). In many highly distributed domains, this is an infeasible solution.

References

Hirayama, K., and Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In Smolka, G., ed., *Principles and Practice of Constraint Programming – CP97*. 222–236.

Hirayama, K., and Yokoo, M. 2000. An approach to over-constrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proc. of the 4th Intl. Conf. on Multi-Agent Systems(ICMAS)*.

Lemaitre, M., and Verfaillie, G. 1997. An incomplete method for solving distributed valued constraint satisfaction problems. In *Proc. of the AAAI Workshop on Constraints and Agents*.

Liu, J., and Sycara, K. 1995. Exploiting problem structure for distributed constraint optimization. In *Proc. of the 1st Intl. Conf. on Multi-Agent Systems(ICMAS)*.

Walsh, W., and Wellman, M. 1998. A market protocol for decentralized task allocation. In *Proc. of the 3rd Intl. Conf. on Multi-Agent Systems(ICMAS)*.

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