No-Commitment Branch and Bound Search for Distributed Constraint Optimization

Anton Chechetka
Robotics Institute, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh PA 15213, USA
antonc@cs.cmu.edu

Katia Sycara
Robotics Institute, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh PA 15213, USA
katia+@cs.cmu.edu

ABSTRACT

We present a new polynomial-space algorithm for solving Distributed Constraint Optimization problems (DCOP). The algorithm, called NCBB, is branch and bound search with modifications for efficiency in a multiagent setting. Two main features of the algorithm are: (a) using different agents to search non-intersecting parts of a search space concurrently, and (b) communicating lower bounds on solution cost every time there is a possibility the bounds might change due to changed variable assignments. The first leads to a better utilization of computational resources of multiple participating agents, while the second provides for more efficient pruning of search space.

Experimental results show that NCBB has significantly better performance than another polynomial-space algorithm, ADOPT, on random graph coloring problems. Under assumptions of cheap communication it also has comparable performance with DPOP despite using only polynomial memory as opposed to exponential memory for DPOP.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and Coordination*

General Terms

Algorithms

Keywords

Cooperative distributed problem solving in agent systems

1. INTRODUCTION

In recent years Distributed Constraint Optimization (DCOP) [2] has received considerable attention as a powerful paradigm for multiagent coordination. Example applications include target tracking in distributed sensor networks [7] and distributed event scheduling [1].

Several efficient distributed algorithms for solving DCOP have been developed recently, both optimal (Synchronous Branch and Bound, ADOPT [7], OptAPO [5], DPOP [8]) and approximate (Iterative Distributed Breakout). Of the optimal algorithms ADOPT, DPOP and OptAPO have the best performance to date [5, 7, 8]. However, more work is needed to make optimal DCOP algorithms efficient enough for real-life problems.

In this paper we present a decentralized algorithm, called No-Commitment Branch and Bound search (NCBB), for solving DCOP, that addresses the problem of performance. It is based on well-known branch and bound search [3]. It runs several concurrent search processes in different partitions of search space. This enables one to speed up the search due to parallelization and also reduces the amount of information that has to be synchronized among agents. We demonstrate that NCBB performs better than ADOPT on distributed graph coloring problems under a variety of metrics. When communication is cheap, it also has comparable performance with DPOP despite using only polynomial memory as opposed to exponential memory for DPOP.

2. DCOP

A Distributed Constraint Optimization problem consists of a set of variables $V = \{x_1, \dots, x_n\}$, a set of discrete finite domains for each of the variables $D = \{D_1, \dots, D_n\}$ and a set of constraints $F = \{f_1, \dots, f_m\}$ where each constraint is a function $f_i : D_{i_1}, \dots, D_{i_j} \rightarrow \mathcal{N} \cup \infty$. Each variable is controlled by an agent that can communicate with other agents. The goal of the agents is to find an assignment for all the variables $B^* = \{d_1, \dots, d_n | d_i \in D_i\}$ such that the global cost, $\sum_{i=1}^{m} f_i$, is minimized. DCOP is NP-complete.

In this article we consider a restricted version of DCOP. We assume all the constraints to depend on exactly two variables, although our approach can in principle be extended to k-ary constraints. We also assume that each variable is controlled by an independent agent and use the terms *agent* and *variable* interchangeably.

We will use the term *constraint graph* to refer to a graph where variables are nodes and two nodes have an edge between them if and only if they share a constraint. We also assume that two agents have a direct communicational link if and only if they share a constraint.

3. NCBB ALGORITHM

Before the algorithm can be executed, the agents need to be prioritized in a depth-first search (DFS) tree, in which

```
function **mainLoop**()
1  **if** (¬ IAmRoot) updateContext();
2  **while** (**true**)
3    search();
4    **if** (IAmRoot ∨ updateContext()) **break**;
5  **end while;**
6  costs[resultValue] = 0;
7  (**for** ∀ y ∈ chldren) subtreeSearch(resultValue, y);
8  (**for** ∀ y ∈ children) send “STOP” to y;
**end mainLoop;**

function **updateContext**()
10  **while** (**true**)
11    receive message m from y ∈ ancestors;
12    **if** (received “search”) bound ← m.BOUND; **return false;**
13    **else if** (received “y=d”)
14      context[y] ← d;
15      send LB(x,context,$indx_y$) - LB(x,context,$indx_y$-1) to y;
16    **else if** (received “STOP”) **return true**;
**end if; end while; end updateContext;**
```

Figure 1: Main loop for agent x

each agent has a single parent and multiple children. Constraints are only allowed between agents that are in an ancestor-descendant relationship in the DFS tree. An agent does not have information about all its ancestors or descendants, but only about those with whom it shares a constraint.

The purpose of such an ordering is to decompose the global cost function: given the assignments to all ancestors, agents in a given subtree can work on minimizing their part of the solution cost independently of agents in other subtrees. Any constraint graph can be ordered into some DFS tree using distributed algorithm from [4].

3.1 Auxiliary definitions

In further explanation all variables names refer to local variables of the considered agent.

DEFINITION 1. *Agent cost for agent x and assignment B*

$$AgentCost(x, B) \equiv \sum_{y \in ancestors_x} f_{x,y}(B(x), B(y))$$

where $B(x)$ is the value of x under assignment B.

DEFINITION 2. *For agent x, assignment B and integer $k \in 0..|ancestors_x|$ denote LB(x,B,k)*

$$LB(x, B, k) = \min_{d_x \in D_x} \min_{B_1 \in B|_{ancestors_x[1..k]}} AgentCost(d_x, B)$$

where $B|_{vars}$ is the set of all assignments that agree with B on the values of variables in $vars$.

3.2 Initialization

During the initialization stage agents compute global upper and lower bounds on the solution cost. Each agent chooses its value greedily given values of its ancestors so as to minimize its $AgentCost$. The costs are propagated up the tree and `bound` variable of the root agent takes the value

$$bound = \sum_{x \in V} AgentCost(x, B_{greedy}) - \sum_{x \in V} LB(x, \emptyset, 0).$$

`bound` variable of an agent plays the role of an upper bound on the cost of this agent's subtree. Thus one can see that NCBB uses nontrivial lower bounds on the solution cost during the search. This idea is closely related to preprocessing heuristics for lower bound estimation for ADOPT [1] that resulted in speedups by an order of magnitude.

```
function **search**()
20  idle ← children;
21  costs ← ∅; unexplored ← ∅; anncdVals ← ∅;
22  minCost ← LB(x, context, |ancestors|);
23  **for** ∀ d ∈ D : AgentCost(d,context) ≤bound + minCost
24    costs[d] ← AgentCost(d,context) - minCost;
25  **end for;**
26  **for** (∀ d ∈ D : costs[d] ≠ ∅) unexplored[d] ← children;
27  **while** (unexplored ≢ ∅ ∨ anncdVals ≢ ∅)
28    **while** idle ≢ ∅
29      choose child ∈ idle; idle ← (idle \ child);
30      choose d : child ∈ unexplored[d];
31      unexplored[d] ← (unexplored[d] \ child);
32      **if** (¬ subtreeSearch(d, child) ∧
33            (∃ c ∈ D: child ∈ unexplored[c]))
34        idle ← (idle ∪ child);
35    **end if; end while;**
36    **if** anncdVals ≢ ∅
37      receive $cost_y$ from y ∈ children;
38      d ← anncdVals[y]; anncdVals[y] ← ∅;
39      costs[d] ← costs[d] + $cost_y$;
40      **if** (costs[d] > bound) prune();
41      **else if** (unexplored[d] ≡ ∅) ∧ (y ∉ $\bigcup_{c \in D}$ anncdVals[c])
42        bound=costs[d]; resultValue = d; prune();
43    **end if; end if;**
44    (**if** ∃ c ∈ D: child ∈ unexplored[c]) idle ← (idle ∪ child);
45  **end while;**
46  **if** (¬ IAmRoot) send $\min_{d \in D}$ costs[d] to parent; **end if;**
**end search;**

function **subtreeSearch**(d, child)
50 send “x=d” to all descendants[child]; anncdVals[child]=d;
51 **for** ∀ y ∈ descendants[child]
52   receive $cost_y$ from y; costs[d] ← costs[d] + $cost_y$; **end for**;
53 **if** (costs[d] > bound) prune(); anncdVals[chld]=∅;
54   **return false**;
55 **else**
56   send “SEARCH, BOUND=bound-costs[d]” to child;
57   **return true**;
**end if; end subtreeSearch;**
```

Figure 2: Main search loop subroutines for agent x

3.3 Main loop

After initialization all agents execute the main search loop listed in Figures 1 and 2. An agent x does not start actively searching until it receives a $SEARCH$ message from its parent (line 12). Before this message is sent by x's parent, all x's ancestors select their values and announce them to their descendants. Therefore, when x receives a message to start searching, it has consistent knowledge about the relevant ancestors' values.

The `search` function contains two most distinct features of the algorithm. One of them is the method of branching: when x selects the next value to explore on a given subtree, it may choose a value different from those announced to x's other subtrees (30). For each value an already known part of the total cost is stored and updated (see `costs` variable). Because a particular value is generally used at different times for different subtrees, one can take into account the costs for already completed subtree searches and compute a tighter upper bound on the solution cost for the next subtree (`subtreeSearch`, (52, 56)). It can be viewed as running simultaneous search processes in different parts of search space, and exploits inherent parallelism of the problem.

The second major feature of NCBB is eager propagation of cost lower bound changes. As agent x learns about its ancestors' assignments, it can compute $LB(x, context_x, k)$ for larger and larger k. Every time x receives a message with the value of its ancestor y, it immediately sends back the amount of change in lower bound on x's $AgentCost$ (15). This leads to more precise lower bounds on partial assignment optimal cost at the point of control (agent y in this case) and in turn to better pruning.

An agent keeps track of the solution costs known so far for each one of its values (`costs` map), which subtrees have not yet explored which of its values (`unexplored`) and what value is currently used in what subtree (`anncdVals`). It updates this information as the subtrees return their search results (lines 39,52).

4. ALGORITHM PROPERTIES

PROPERTY 1. *NCBB always terminates after finite time.*

PROPERTY 2. *The amount of memory required for agent x is $O(|D_x| \times |neighbors_x|)$.*

PROPERTY 3. *After terminating, resultValue variables values of all agents form optimal variable assignment B^*:*

$$B^* = \arg\min_B \sum_{i=1}^{m} f_i(B)$$

PROPERTY 4. *The number of messages processed by any agent between receiving SEARCH message and reporting results in corresponding $cost_x$ message is $O(|D_x| \times |neighbors_x|)$. Also, the size of each message does not depend on the problem, i.e. algorithm uses only messages of size $O(1)$.*

5. EVALUATION

The main metric for evaluation was concurrent constraint checks (CCC) [6]. It combines computational and communicational cost of solving the problem and allows for parallelism. The unit of cost is time needed to perform one constraint check. The cost of a message is defined in terms of constraint checks and does not depend on message size. It is easy to add CCC as a benchmark for a simulated distributed system by adding a special field with a value of local "tick counter" v to every message. Upon receiving the message, the receiver updates its counter value according to

$$v_{receiver} = \max\{v_{receiver}, v_{sender} + cost_{message}\},$$

When an agent evaluates a constraint, v is increased by 1. The cost of solving the problem is the maximal value of v over all agents.

Fig. 3(a), 3(b) and 3(c) show the dependency of the problem solving cost on the number of variables for sparse (3(a)) and dense (3(b), 3(c)) settings. The respective message costs were $cost_{message} = 10^3, 0, 10^6$ ($cost_{message} = 0$ corresponds to pure computational cost of an algorithm). In Fig. 3(d) one can see the total number of messages sent by all agents during the solution process depending on the problem size.

Summarizing the experimental results, we conclude that for almost all parameter settings NCBB performance-wise dominates ADOPT, and DPOP dominates NCBB. However,

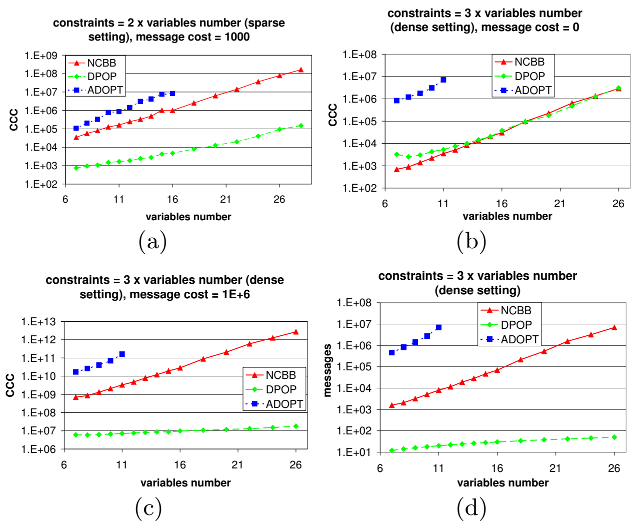

Figure 3: NCBB performance results

DPOP requires exponential memory on each agent, so it is not always applicable. For example, nodes in sensor networks have scarce memory, and it is enough even for one agent to not have enough space for DPOP to fail. For such applications NCBB may be the optimal choice given its performance advantage over another polynomial algorithm and the fact that NCBB is itself polynomial-space.

6. CONCLUSIONS

We have presented a novel backtracking-based algorithm for solving DCOP. It has polynomial memory requirements and uses messages of constant size. It offers significant performance improvements over the state of the art polynomial-memory algorithm ADOPT, and thus may be the optimal choice for settings with scarce memory, such as sensor nets.

7. REFERENCES

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

[2] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.

[3] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[4] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

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

[6] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. of DCR Workshop, AAMAS*, 2002.

[7] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 16(1–2):149–180, 2005.

[8] A. Petcu and B. Faltings. An efficient constraint optimization method for large multiagent systems. In *Proc. of the LSMAS Workshop, AAMAS*, 2005.