Directed Soft Arc Consistency in Pseudo Trees

Toshihiro Matsui
Nagoya Institute of Technology
Gokiso-cho, Showa-ku
Nagoya 466-8555, Japan
matsui.t@nitech.ac.jp

Marius Călin Silaghi
Florida Institute of Technology
150 W. University Blvd.
Melbourne, FL 32901
msilaghi@fit.edu

Katsutoshi Hirayama
Kobe University
5-1-1 Fukaeminami-machi,
Higashinada-ku,
Kobe 658-0022, Japan
hirayama@maritime.kobe-u.ac.jp

Makoto Yokoo
Kyusyu University
744 Motooka, Nishi-ku
Fukuoka 819-0395, Japan
yokoo@is.kyushu-u.ac.jp

Hirohsi Matsuo
Nagoya Institute of Technology
Gokiso-cho, Showa-ku
Nagoya 466-8555, Japan
matsuo@nitech.ac.jp

ABSTRACT

We propose an efficient method that applies directed soft arc consistency to a Distributed Constraint Optimization Problem (DCOP) which is a fundamental framework of multi-agent systems. With DCOPs a multi-agent system is represented as a set of variables and a set of constraints/cost functions. We focus on DCOP solvers that employ pseudo-trees. A pseudo-tree is a graph structure for a constraint network that represents a partial ordering of variables. Most pseudo-tree-based search algorithms perform optimistic searches using explicit/implicit backtracking in parallel. However, for cost functions taking a wide range of cost values, such exact algorithms require many search iterations, even if the constraint density is relatively low. Therefore additional improvements are necessary to reduce the search process. A previous study used a dynamic programming-based preprocessing technique that estimates the lower bound values of costs. However, there are opportunities for further improvements of efficiency. In addition, modifications of the search algorithm are necessary to use the estimated lower bounds.

The proposed method applies soft arc consistency (soft AC) enforcement to DCOP. In the proposed method, directed soft AC is performed based on a pseudo-tree in a bottom up manner. Using the directed soft AC, the global lower bound value of cost functions is passed up to the root node of the pseudo-tree. The value of each cost function is also reduced. As a result, the original problem is converted to an equivalent problem which is efficiently solved using common search algorithms. The performance of the proposed method is evaluated by experimentation. The results show that it is more efficient than previous methods that estimate the lower bound of costs. Moreover, the proposed method is efficient for approximation algorithms that use bounded errors.

Categories and Subject Descriptors

I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence

Cite as: Directed Soft Arc Consistency in Pseudo Trees, Toshihiro Matsui, Marius Călin Silaghi, Katsutoshi Hirayama, Makoto Yokoo, Hirohsi Matsuo, *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. 1065–1072

General Terms

Algorithms

Keywords

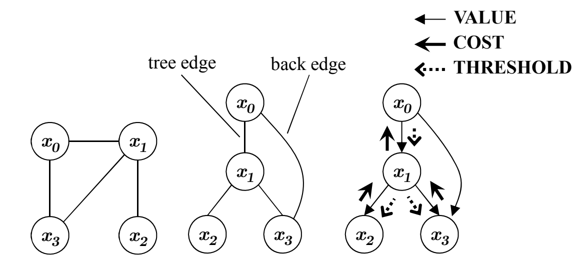
constraint reasoning, distributed constraint optimization problem, soft arc consistency, multi-agent systems

1. INTRODUCTION

A Distributed Constraint Optimization Problem (DCOP) is a fundamental framework of multi-agent cooperation [8, 9, 10, 14, 16]. With DCOPs multi-agent systems are formalized as a set of variables, a set of constraints, and a set of cost functions related to the constraints. Each variable represents the state of an agent. Constraints and related cost functions represent the relationships between agents. Distributed resource allocation problems, which including sensor networks and meeting scheduling, are modeled as DCOPs [4, 6, 7, 15]. Distributed cooperative search algorithms are employed to obtain a solution that globally optimizes the values of cost functions.

In this work, we focus on exact search algorithms that exploit a pseudo-tree [9]. A pseudo-tree [5, 12] is a graph structure that gives a partial order of the variables in a constraint network. Most pseudo-tree-based exact search algorithms perform an optimistic search with implicit/explicit backtracking that takes place in parallel. However, when cost functions return a wide range of values, these exact algorithms take many search iterations, even if the density for constraints is relatively low. The main reason for this drawback is the redundant search for the lower bounds of evaluated values. Therefore additional efficient methods are necessary to reduce the search.

Candidates of additional efficient methods are categorized into preprocessing and add-on processing. Preprocessing is reasonable if it is simply added to existing preprocessing for pseudo-tree generation. Add-on processing may be efficient because it exploits more information obtained during the search process. However the modification of the distributed search algorithm may not be easy. In a previous study, a preprocessing technique estimated the lower bound of evaluated values [1, 7]. A dynamic programming method performs the estimation. However, there are opportunities for further improvements of efficiency. In addition, the search algorithm has to be modified to use the estimated lower bounds.



(a) constraint network (b) pseudo-tree (c) message paths of in ADOPT

Figure 1: pseudo-tree and ADOPT

We propose an efficient method that applies soft arc consistency (soft AC) [3, 13] to DCOPs. The soft AC generalizes arc consistency and can be applied to extended classes of constraint satisfaction problems including DCOPs. For constraint optimization problems, additional control is necessary to avoid the infinite loop of soft AC. In the proposed method, soft AC is performed based on pseudo-trees in a bottom up manner. Such processing can be considered a version of directed soft arc consistency [3]. Using pseudo-tree-based directed soft arc consistency, the global lower bound value is passed up to the root node of the pseudo-tree, reducing values of each cost function. As a result, the original problem is converted to an equivalent problem that is efficiently solved using common search algorithms. No modification of search algorithms is necessary except for the evaluation of unary constraints generated by soft arc consistency. The directed soft AC in pseudo-trees resembles a previous method using dynamic programming. However, it not only estimates the lower bound but also modifies the problem. Redundancy of the original problem is well reduced by the modification. Moreover, the proposed method is efficient for approximation algorithms that use bounded errors.

The outline of the paper is as follows. In Section 2, the background of the work is shown, which includes DCOP, pseudo-tree-based cost computation, directed soft AC. Then we propose a pseudo-tree based directed soft AC in Section 3. Our proposed method is evaluated by the results of experiments in Section 4 and proposed method is compared with the preprocessing methods of a previous work. In Section 5, considerations are shown about related works. We present our conclusion in Section 6.

2. BACKGROUND

The background of our work includes the distributed constraint optimization problem, pseudo-tree-based cost computation, and directed soft arc consistency enforcing.

2.1 Distributed constraint optimization

A distributed constraint optimization problem is defined by a set A of agents, a set X of variables, a set C of binary constraints and a set F of binary functions. Agent i has its own variable x_i. x_i takes a value from discrete finite domain D_i. The value of x_i is controlled by agent i. Constraint $c_{i,j}$ represents the relationship between x_i and x_j. The cost of an assignment $\{(x_i, d_i), (x_j, d_j)\}$ is defined by a binary function $f_{i,j}(d_i, d_j) : D_i \times D_j \rightarrow \mathbb{N}$. The goal is to find a global optimal solution $\mathcal{A}$ that minimizes the global cost function: $\sum_{f_{i,j} \in F,\ \{(x_i,d_i),(x_j,d_j)\} \subseteq \mathcal{A}} f_{i,j}(d_i, d_j)$.

2.2 Pseudo-tree

A pseudo-tree [5, 12] is a graph structure that defines a partial order on variables. The pseudo-tree is generated using a depth first traversal of the constraint network. For example, the pseudo-tree in Figure 1 (b) is generated from the constraint network in Figure 1 (a). The edges of the original constraint network are categorized into either the tree or the back edges of the pseudo-tree. The tree edges represent the partial order relation between the two variables.

There is no back edge between different sub-trees. Therefore, a divide-and-conquer strategy can be applied to the search processing for different sub-trees. By employing this property, search processing can be performed in parallel.

2.3 Computation of cost value in pseudo-trees

In this work, how search algorithms compute cost values using pseudo-trees is important. We show an outline of cost computation in ADOPT [9], an efficient distributed constraint optimization algorithm. Agent i knows the following information.

- current context: current partial solution of the ancestor nodes of x_i. Assignments in the current context are received from other agents.
- $lb_i(x_j, d)$, $ub_i(x_j, d)$: boundary of optimal cost for each value d of variable x_i and sub-tree routed at x_i's child variable node x_j. $lb_i(x_j, d)$ and $ub_i(x_j, d)$ are received from child node x_j. Each boundary value is related to an assignment used in the computation. When the assignment for (x_j, d) is incompatible with i's current context, $lb_i(x_j, d)$ and $ub_i(x_j, d)$ are reset to 0 and ∞ respectively.

The computation in agent i is shown as follows. Local cost $\delta_i(d)$ for value d of variable x_i and i's current context is defined as follows.

$$\delta_i(d) = \sum_{(x_j,d_j) \in i\text{'s current context},\ j \in \text{upper neighborhood nodes of } i} f_{i,j}(d, d_j) \tag{1}$$

Upper bound $UB_i(d)$ and lower bound $LB_i(d)$ for value d of variable x_i and the sub-tree routed at x_i are defined as follows.

$$LB_i(d) = \delta_i(d) + \sum_{j \in \text{child nodes of } i} lb_i(x_j, d) \tag{2}$$

$$UB_i(d) = \delta_i(d) + \sum_{j \in \text{child nodes of } i} ub_i(x_j, d) \tag{3}$$

Upper bound UB_i and lower bound LB_i for the sub-tree routed at x_i are defined as follows.

$$LB_i = \min_{d \in D_i} LB_i(d) \tag{4}$$

$$UB_i = \min_{d \in D_i} UB_i(d) \tag{5}$$

ADOPT performs distributed asynchronous processing based on the branch-and-bound and A* algorithms. Message paths in ADOPT are shown in Figure 1 (c). In this example, messages are sent based on the pseudo tree in Figure 1 (b). VALUE, COST, and THRESHOLD messages are exchanged between agents. Current value d_i of x_i is sent to the lower neighborhood nodes of x_i using a VALUE message. $lb_i(x_j, d)$ and $ub_j(x, d)$ are received from child node x_j of x_i using a COST message. A THRESHOLD message is used to allocate costs among sub-trees. As a result of search processing, at root node r, LB_r and UB_r converge into the global optimal cost. The global optimal solution is decided based on the optimal cost. The details of the ADOPT are shown in [9]. Other search algorithms that employ the pseudo-tree use similar computation for costs. LB_i and UB_i immediately reach their true values if the search algorithm performs effectively. In the case of minimization problems, optimistic search takes many iterations to compute the true lower bound. Therefore, improving accuracy of the lower bounds is important. A previous work employed lower bounds that are estimated in preprocessing, which is a subset of dynamic programming.

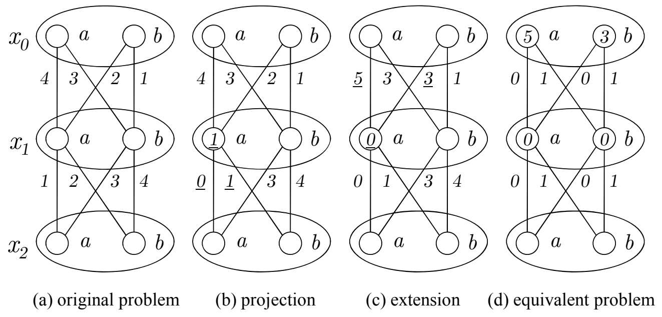


Figure 2: directed soft arc consistency enforcing

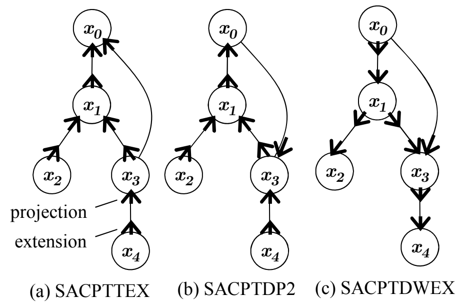


Figure 3: pseudo-tree based directed soft arc consistency enforcing

2.4 Soft arc consistency

Soft arc consistency enforcement [3, 13] generalizes arc consistency. The soft AC can be applied to an extended class of constraint satisfaction problems including DCOPs. Using soft AC, the original problem is converted into an equivalent problem that is more effectively solved. In addition to set F of binary cost functions, soft AC employs set F^1 of unary functions for variables. A unary function for x_i is denoted as f_i. Moreover, we use the following notations. $t \in f$ denotes a tuple of assignments for binary function f. $t_{\downarrow\{i\}}$ = (d) denotes that tuple t contains an assignment (x_i, d). $X_f \subseteq X$ denotes a set of variables that are related to a function f.

The process of enforcing soft arc consistency is shown in Algorithm 1[1]. In soft AC, both projection and extension operation are iteratively applied to the constraint network. Projection moves costs from binary constraint f to unary constraint f_i based on the lower bound of f for the assignment (x_i, d). Extension moves costs from unary constraint f_i to binary constraint f based on the lower bound of f_i for the assignment (x_i, d). In this algorithm, arc consistency operations are simply repeated until the equivalent problem converges. In the general case of constraint optimization problems, soft arc consistency may not converge, because a projection is an inverse of an extension. In line 13 of Algorithm 1, extension is allowed when the unary cost is absorbing. This limitation avoids the infinite loop. However, if cost functions take non-absorbing values, extension operations are not applied. Therefore it is necessary to apply extension for non-absorbing costs. On the other hand, additional methods are also necessary for convergence.

2.5 Directed soft arc consistency

Directed soft Arc Consistency (directed soft AC) [3] sequentially applies soft AC operations based on a direction on the constraint network. Using the direction, directed soft AC converges even if it performs the extension for non-absorbing costs.

An example of directed soft AC is shown in Figure 2. (a) is the original problem, which consists of three variables and two binary constraints/functions. Each variable takes a value from its domain $\{a, b\}$. Each label for an edge represents a cost for a tuple of assignments. The initial values of unary functions are set to zero. The initial unary costs are omitted in the figure. In this example, directed soft AC is performed from x_2 to x_0. (b) shows a projection that moves the cost of $f_{1,2}$ to f_1. (c) shows a extension that moves the cost of f_1 to $f_{0,1}$. The modified costs are underlined in the figure. Similarly, projection and extension are applied for all assignments of all functions. (d) is an equivalent problem obtained as a result. Note that global lower bound values are passed up to upper variable nodes. Finally, the global lower bound is summed into f_0. The costs for other unary functions are decreased to zero. In addition, separating the global lower bound value of 3 in the root node is possible. However we prefer the problem shown as (d).

[1]The original pseudo code is shown in [13]. We modified some notations.

3. DIRECTED SOFT AC IN PSEUDO-TREE

In this work, we propose a preprocessing method that applies directed soft AC based on pseudo-trees. In the following we explain the proposed method.

3.1 Applying directed soft AC

As shown in subsection 2.3, some DCOP solvers compute costs based on the pseudo-trees in a bottom up manner. Therefore it is reasonable to apply directed soft AC based on pseudo-trees in a similar manner. The pseudo-tree-based directed soft AC gives an equivalent problem that reduces the iteration of search processing for the lower bounds of costs. Figure 3 (a) shows an example of directed soft AC based on pseudo-tree. The directed soft AC processing is applied from the leaf nodes to the root node. The projection and extension procedures are the same as the ones of Algorithm 1 except that extension is also applied to the non-absorbing costs. Each node performs the following two processing phases:

1. projection between its own unary function and binary functions for all lower neighborhood nodes.
2. extension between its own unary function and a binary function for its parent node.

In the extension phase, the whole cost of its own unary function is moved to upper tree edge. No extension operations are performed for upper back edges. It is necessary to sum up cost values correctly. Projection in the leaf nodes is obviously unnecessary. If a leaf node has an initial unary function, extension can be performed. Non-leaf nodes perform the directed soft AC after all child nodes complete the processing.

Note that projection for a back edge is applied to its upper variable node. That is different from the cost evaluation for the back edges in subsection 2.3. In the DCOP solvers, cost evaluation for a back edge is performed by its lower variable node. Then the evaluated cost is sent to its ancestor nodes by tree edges. For example, in Figure 3, the cost for $f_{0,3}$ is evaluated by x_3. The cost is necessary in x_1 and x_0. On the other hand, the direction of the soft AC is not originally restricted. Therefore it can be applied in a different way. For example, the following method is also possible. (1) The costs of back edge are moved to lower variable node. (2) Then the costs of the lower variable node are moved to its upper tree edge. That resembles the computation of DCOP solvers. The example is shown in Figure 3 (b). Moreover, moving a part of costs in the projection and the extension is possible. For example, a cost for a variable can be distributed equally to all upper edges.

If a pseudo-tree has no back edges, the equivalent problem directly gives a global optimal solution. In root node r, its optimal assignment is $(x_r, \text{argmin}_d f_r(d))$. The global optimal cost

Algorithm 1: soft arc consistency enforcing

```
Projection($f$, $i$, $d$) begin
  $m \leftarrow 0$; $v \leftarrow \infty$;
  foreach $t \in f$ s.t. $t_{\downarrow\{i\}}$ = (d) do $v \leftarrow \min(v, f(t))$;
  if $v$ affects $f_i(d)$ then begin
    $f_i(d) \leftarrow f_i(d) + v$; $m \leftarrow 1$;
    foreach $t \in f$ s.t. $t_{\downarrow\{i\}}$ = (d) do $f(t) \leftarrow f(t) - v$;
  end;
  return $m$;
end.

Extension($i$, $d$, $f$) begin
  $m \leftarrow 0$; $v \leftarrow f_i(d)$;
  if $v$ is $\infty$ then begin                // remove for directed soft AC
    foreach $t \in f$ s.t. $t_{\downarrow\{i\}}$ = (d) do
      if $v$ affects $f(t)$ then begin
        $f(t) \leftarrow f(t) + v$; $m \leftarrow 1$;
      end;
    $f_i(d) \leftarrow f_i(d) - v$;
  end;
  return $m$;
end.

SAC($X$, $D$, $F^1 \cup F$) begin
  $m \leftarrow 1$;
  while ($m$) do begin
    $m_p \leftarrow m_e \leftarrow 0$;
    foreach $f \in F$ do
      foreach $i \in X_f$ do
        foreach $d \in D_i$ do $m_p \leftarrow m_p \vee$ Projection($f$, $i$, $d$);
    foreach $i \in X$ do
      foreach $f \in F$ s.t. $i \in X_f$ do
        foreach $d \in D_i$ do $m_e \leftarrow m_e \vee$ Extension($i$, $d$, $f$);
    $m \leftarrow m_p \vee m_e$;
  end;
end.
```

Algorithm 2: pseudo-tree based directed soft AC enforcing

```
1  // initialize node i
2  let $N_i$ denote neighborhood nodes of $i$;
3  $p_i \leftarrow \phi$;                     // parent node of i
4  $L_i \leftarrow \{\}$;                    // descendant nodes of i
5  $U_i \leftarrow N_i$;                     // unvisited neighborhood nodes of i
6  foreach $j \in N_i$ do $m_{i,j} \leftarrow 0$;    // flags
7  if $i$ is the initiator node then send loop-back DISCOVER message to $i$;
8  process messages as follows;

10 // arrive from i's parent node j
11 for DISCOVER message from $j$ do begin
12   $p_i \leftarrow j$;                    // $p_i \leftarrow i$ if i is the initiator node
13   foreach $k \in N_i \backslash \{j\}$ do begin   // notify that i has been visited
14     send VISITED message to $k$; $m_{i,k} \leftarrow 1$;
15   end;
16   if $N_i = \{j\} \wedge j \neq i$ then begin   // j is the only one neighbor of i
17     foreach $d \in D_i$ do Extension($i$,$d$,$f^i_{j,i}$);
18     send RETURN($\{i\}$, $f^i_{j,i}$) to $j$;          // backtracking
19     foreach $d \in D_j$ do Projection($f^i_{j,i}$,$j$,$d$);   // use dummy $f_j$
20   end;
21 end;

23 // return from i's child node j/resume depth first search
24 for RETURN(L,$f$) message from $j$ do begin
25   if $j \neq i$ then begin $L_i \leftarrow L_i \cup L$; replace $f^i_{i,j}$ by $f$; end;
26   if there exists $k \in U_i$ then begin          // visit next node
27     send DISCOVER to the most prior $k$; $U_i \leftarrow U_i \backslash \{k\}$;
28   end else begin
29     foreach $l \in N_i \cap L_i$ do foreach $d \in D_i$ do Projection($f^i_{i,l}$,$i$,$d$);
30     if $p_i \neq i$ then begin                // return to parent node
31       foreach $d \in D_i$ do Extension($i$,$d$,$f^i_{p_i,i}$);
32       send RETURN($L_i \cup \{i\}$,$f^i_{p_i,i}$) to $p_i$;   // backtracking
33       foreach $h$ s.t. $h \in N_i \wedge h \notin L_i$ do
34         foreach $d \in D_h$ do Projection($f^i_{h,i}$,$h$,$d$);   // use dummy $f_h$
35     end else the algorithm has terminated;
36   end;
37 end;

39 // remove i's neighborhood node j from unvisited nodes
40 for VISITED message from $j$ do begin
41   $U_i \leftarrow U_i \backslash \{j\}$; send ACK message to $j$;
42 end;

44 // receive acknowledge of VISITED message
45 for ACK message from $j$ do begin
46   $m_{i,j} \leftarrow 0$;
47   if $m_{i,k}$ = 0 for all $k \in N_i$ then
48     send loop-back RETURN({},{}) to $i$;   // resume depth first search
49 end;
```

is $\min_d f_r(d)$. In non-root node i, when its parent node j has an optimal assignment (x_j, d_j), i's optimal assignment is d_i such that $f_{i,j}(d_i, d_j) = 0$. If a pseudo-tree has back edges, DCOP solvers are necessary to find a global optimal solution. Equivalent problems obtained using the soft AC are solved using common DCOP solvers. Equivalent problems contain unary functions. For most DCOP solvers, adding evaluations for unary functions is easy. In fact, pseudo-tree-based directed soft AC passes up the global lower bound to the root node of the pseudo-tree. Therefore only the root node has a non-zero unary function if there are no absorbing costs.

A pseudo-tree-based directed soft AC is shown in Algorithm 2. The algorithm is based on a distributed depth-first search [2]. Basically, the algorithm performs a depth first search traversal using four messages: DISCOVER (arrive to next node), RETURN (backtrack), VISITED (disable neighborhood) and ACK (synchronize). As a result, a pseudo-tree for a constraint network is built. There are a few minor modifications to embed soft AC processing. The soft AC is performed in the backtracking steps (lines 17-19, 25, 29-34). Note that each projection/extension must be performed in two nodes. In the pseudo code, $f^i_{j,k}$ denotes i's copy of $f_{j,k}$. In bottom side node of a edge, a dummy unary function is used in a projection (lines 19 and 34). RETURN messages are also modified to handle information of soft AC.

3.2 Preprocessing overheads

In arc consistency enforcing, projection and extension are applied to a couple of a variable's value and tuples of assignments that contain the variable's value. Their computational complexity is linear with the number of tuples. The proposed method applies projection and extension based on the pseudo-tree. Projection is applied for each constraint and for each value of a variable that is the upside node of the constraint edge. Extension is also applied for each constraint and for each value of a variable. Therefore, the complexity of the total computation is linear with the number of tuples for all constraints. The space complexity is linear with the total number of values for all variables. Computational overhead of the preprocessing is clearly less than the corresponding iterative processing of ADOPT.

3.3 Correctness

In equivalent problems obtained using soft AC, each cost value for a complete solution equals the one in the original problem. Therefore the global optimal cost and solution are the same as the original ones. No modification of search algorithms is necessary except for the evaluation of unary constraints that are generated by soft arc consistency. Therefore correctness and termination hold in most search algorithms.

3.4 Using bounded errors

Even if soft AC reduces the redundancy of the original problem, exact search methods requires many iterations for difficult prob-

Table 1: calculation of DP0, DP1 and DP2

DP0	$h_i(d_i) := \sum_{j\in \text{ child nodes of } i} \sum_{k\in \text{ upper neighborhood nodes of } j} \min_{d_j\in D_j} \min_{d_k\in D_k} f_{j,k}(d_j, d_k)$
DP1	$h_i(d_i) := \sum_{j\in \text{ child nodes of } i} \min_{d_j\in D_j}(h_j(d_j) + f_{i,j}(d_i, d_j))$
DP2	$h_i(d_i) := \sum_{j\in \text{ child nodes of } i} \min_{d_j\in D_j}(h_j(d_j) + f_{i,j}(d_i, d_j) + \sum_{k\in \text{ upper neighborhood nodes of } j\setminus\{i\}} \min_{d_k\in D_k} f_{j,k}(d_j, d_k))$

lems that consist of a large number of variables and dense constraints. In such cases, approximation with bounded errors is useful [9]. Bounded error assures solution quality and reduces search iterations. Basically, the approximation method terminates when the difference between the upper and lower bounds reaches the parameter value of the bounded error [2] in the root node. Pseudo-tree based directed soft AC preprocessing is also efficient for the approximation method. Global lower bound values in the root node immediately push up the lower bound. Moreover, the reduced constraint costs improves convergence of the lower and upper bounds into the bounded errors.

4. EVALUATION

We evaluate the proposed method by the results of experiments. The efficiency of the proposed method and previous method was compared. We also analyzed and considered their efficiency.

4.1 Settings of experiment

We used graph coloring problems with three colors. Each problem consists of n ternary variables and $d \times n$ binary constraints. d is a parameter for the density of constraints. The costs of constraints are randomly set from integer values between 1 and 100 with uniform probability. The results are averaged for fifty problem instances. ADOPT and preprocessing methods were applied to each problem. The preprocessing methods are as follows:

- no preprocessing (ORIG)
- dynamic programming based methods(DP0, DP1, and DP2)
- directed soft AC based methods (SACND, SACPTDP2, SACPTP, and SACPTDTEX)

DP0, DP1, and DP2 are dynamic programming based methods proposed in [1]. They estimate a lower bound value for each value d_i of a variable x_i. Table 1 shows the lower bound calculation that is performed based on the pseudo-tree in a bottom up manner. This computation is rather simple. However, modifications of ADOPT are necessary to exploit the estimated lower bound. We modified ADOPT as follows.

- backtracking threshold: in ADOPT, each variable node maintains values called *backtracking threshold*. Backtracking threshold thr_i of node i represents a cost allocated to the sub-tree rooted at i. In the original version of ADOPT, thr_i takes a value between LB_i and UB_i. In the modified version, the lower bound of thr_i is limited to $h_i(d_i)$. d_i denotes the current assignment of x_i.
- $lb_i(x_j, d)$: as shown in subsection 2.3, $lb_i(x_j, d)$ is initialized/reset to zero in the original version. In the modified version, $lb_i(x_j, d)$ is initialized/reset to $min_{d'\in D_j} h_j(d')$. It is assumed that j's parent node i knows $h_j(d')$. However, in ADOPT, each node has no information about current assignments of its child nodes. Therefore the minimum value of estimated lower bounds is always used.
- initial assignment of x_i: x_i is initialized to $\text{argmin}_d h_i(d)$.

[2]In fact, ADOPT with error bound always keeps a margin between the backtracking threshold and the lower bounds based on the bounded error [9].

Refer to [1, 9] for details.

SACND is a non-directional soft AC shown in Algorithm 1. Note that SACND does not perform any extension operations. The extension is blocked by the condition in line 13 of Algorithm 1. That is necessary to avoid infinite loop of projections and extensions. SACPTDTEX is the pseudo-tree based directed soft AC shown in subsection 3.1. SACPTP is a subset of SACPTDTEX. SACPTP only performs projection and resembles SACND except for the direction of the soft AC. SACPTDP2 is a different version of the pseudo-tree-based directed soft AC that is similar to DP2. An example of SACPTDP2 is shown in Figure 3 (b). The difference between SACPTDP2 and SACPTDTEX is the direction of projection for the back edges. SACPTDTP2 performs the projection for a back edge and its downside node. In the root node, the estimated lower bounds of SACPTDP2 equal those of DP2. However, in SACPTDP2, cost values of the binary constraints are reduced. Moreover, the non-absorbing values of the unary cost functions of non-root nodes are reduced to zero.

Pseudo-trees are generated using depth first traversal with most-constrained order. In our experiments, preprocessing overhead was ignored because it is sufficiently small compared to search processing. We used simulation programs that iterate message cycles. In a message cycle, each agent reads the messages from its receiving queue. Then the agent writes messages for the sending queue. The messages in each queue are exchanged at the end of the cycle. The number of message cycles was limited to 10^6. The experiment was aborted at the limit number of message cycles. In that case, the number of message cycles of the instance is considered the limit number.

4.2 Efficiency for ADOPT

We evaluated the performance of the combination of ADOPT with preprocessing. The number of message cycles at convergence and the ratio of instances that correctly terminated are shown in Figure 4. The result shows that SACPTDTEX is most efficient to reduce message cycles. In the case of $n = 25$, some instances of SACPTDTEX, SACPTDP2 and DP2 correctly terminated. SACPTP reduces more message cycles than SACND does. Both of them only apply projection. The result shows that SACPTP is more efficient than SACND because SACPTP exploits the pseudo-tree. Another difference between them is the direction of soft arc consistency. Note that the effect of DP2 is less significant than the one of SACPTDP2, although they estimate the same global lower bounds in the root node.

4.3 Accuracy of lower bound

Each preprocessing method estimates lower bound values of costs. Accuracy of the lower bound for the optimal cost is shown in Figure 5. DP0 avg., DP1 avg. and DP2 avg. shows averaged accuracy for all nodes except leaf nodes. Others show accuracy at root nodes[3]. The results are averaged for all instances that correctly terminated. Note that a lower bound is estimated for each value of a variable. Figure 5 (a) shows the accuracy of estimated lower bound for a variable's value in an optimal assignment. DP2, SACPTDP2 and SACPTDTEX estimates relatively higher lower bounds. The

[3]While averaged accuracies are evaluated in the study of DP0, DP1 and DP2 [1], we mainly focus on the accuracy at root nodes.

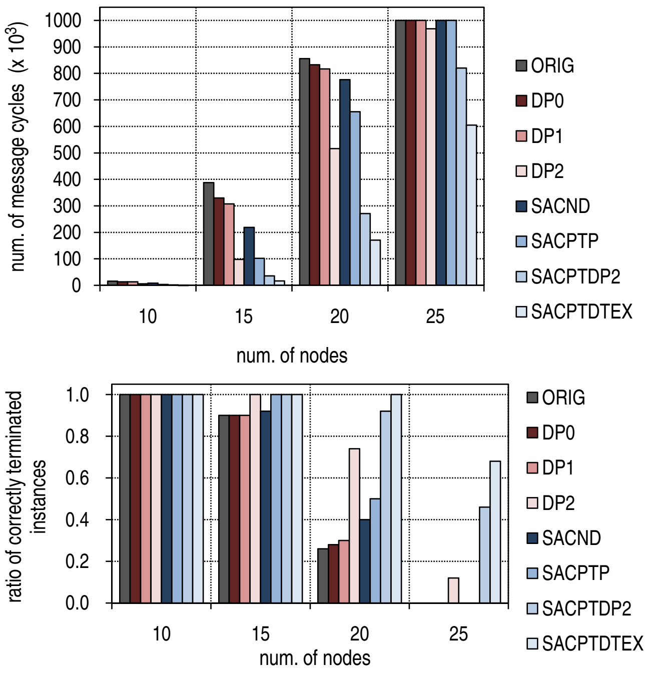
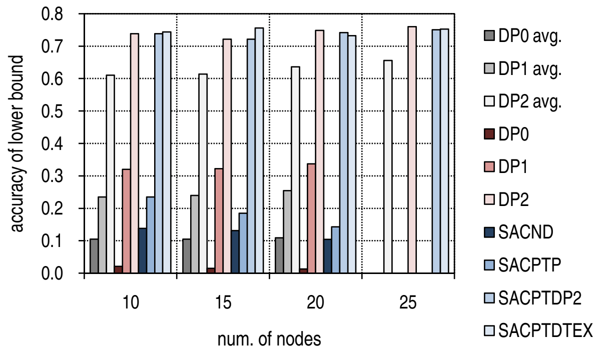
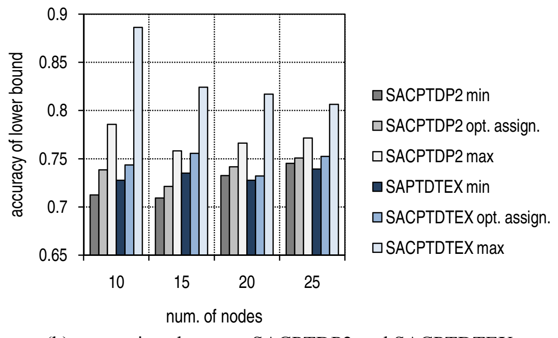


Figure 4: ADOPT (d=2)



(a) accuracy of estimated lower bound for a variable's value in optimal assignment



(b) comparison between SACPTDP2 and SACPTDTEX

Figure 5: accuracy (d=2)

higher lower bound value can be considered as a main reason of better effects of these method. Although accuracies of DP2 and SACPTDP2 are equal in the root node[4], SACPTDP2 is more effective to reduce search iterations.

As a result of DP2, better lower bounds are obtained. However,

[4]In Figure 5 (a), Averaged accuracies of DP2 and SACPTDP2 are slightly different due to the difference of solutions/instances that correctly terminated.

Table 2: Cost of tuples (d=2)

problem	n	weight of tuples			num. of tuples [%]			
		min.	max.	ave.	0	1-50	51-100	101-
ORIG	10	1.3	99.8	49.9	0	50.2	49.8	0
	15	1.1	99.9	50.0	0	49.9	50.1	0
	20	1.0	100.0	49.9	0	50.1	49.9	0
	25	1.0	100.0	49.9	0	50.3	49.7	0
SACPTDP2	10	0	147.3	29.3	33.6	40.9	22.0	3.5
	15	0	163.6	29.6	33.7	40.7	21.8	3.8
	20	0	172.1	29.0	33.8	41.7	21.1	3.5
	25	0	178.2	29.0	33.7	41.4	21.5	3.4
SACPTDTEX	10	0	143.9	27.8	33.8	42.3	21.7	2.2
	15	0	161.6	28.3	33.8	41.9	21.3	2.9
	20	0	170.8	28.2	33.8	42.2	21.3	2.7
	25	0	176.3	28.4	33.8	41.9	21.5	2.8

DP2 does not modify the original problem. Therefore, ADOPT has to be modified to exploit the estimated lower bound. In the modified version of ADOPT, the estimated lower bound is used to limit the original lower bound. As shown in subsection 4.1, each node i knows its estimated value $h_i(d)$ that limits i's lower bound. In addition, i knows the estimated value $h_j(d')$ of its child node j. $h_j(d')$ improves the accuracy of allocation of costs between i and j. However, it is considered that the improvement in the modified version of ADOPT is smaller than one that is obtained by SACPTDP2. More improvements of the modified ADOPT may be possible. However, more modifications of ADOPT are also necessary. On the other hand, the pseudo-tree based directed soft AC converts the original problem into an equivalent problem. The global lower bound value is moved into the root node of the pseudo-tree. Almost all cost values are reduced for all constraints. In particular, unary non-absorbing constraints are substantially removed except at the root node. Therefore redundancy of cost functions between a parent node and its child nodes is reduced as much as possible. In search algorithms, any complicated computation to exploit the difference of estimated values between a parent node and its child nodes are unnecessary.

Figure 5 (b) shows a comparison of the accuracy between SACPTDP2 and SACPTDTEX. In the figure, maximum and minimum values of the accuracy are also shown. The maximum value of the accuracy of SACPTDTEX is larger than the one of SACPTDP2. The effect of the highest lower bound value should be studied further. However, we can infer that the highest lower bound limits the flipping of values of the variable in the root node, because the variable's value is optimistically selected based on its lower bound cost. Cost of tuples that are contained in binary cost functions is shown in Table 2. The cost values of original problems have uniform distribution. In equivalent problems, the average cost value is less than ones of original problems. Especially, cost values of $|D_i|$ tuples in each cost function $f_{i,j}$ are decreased into zero. Almost all instances of SACPTDP2 perform $3dn$ projections and $3(n-1)$ extensions. SACPTDTEX performs $3dn$ projections and $3(n-1-(\text{number of leaf nodes}))$ extensions. In this problem setting, initial costs of unary constraints are zero. However, SACPTDP2 temporally increases the unary cost in leaf nodes. Therefore extension is necessary in leaf nodes.

4.4 ADOPT with bounded errors

We evaluated the effect of preprocessing methods for ADOPT with bounded errors. The number of message cycles at the convergence and the ratio of instances that correctly terminated are show in Figures 6 and 7. With bounded error, ADOPT can correctly terminate in densely constrained problems. In the case with bounded errors, the proposed method more efficiently reduces the number of message cycles. ADOPT immediately terminates when the estimated lower bound is sufficiently greater than the difference between optimal cost and bounded error in the root node. The pro-

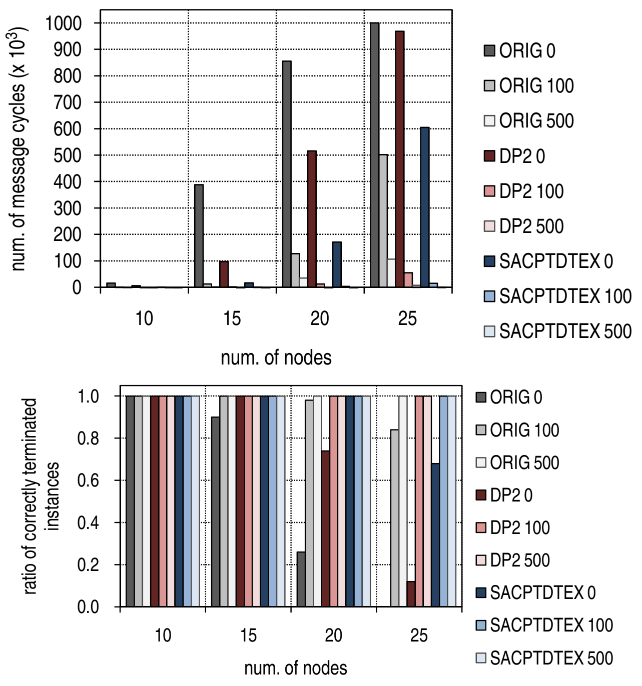


Figure 6: ADOPT with bounded errors (d=2)

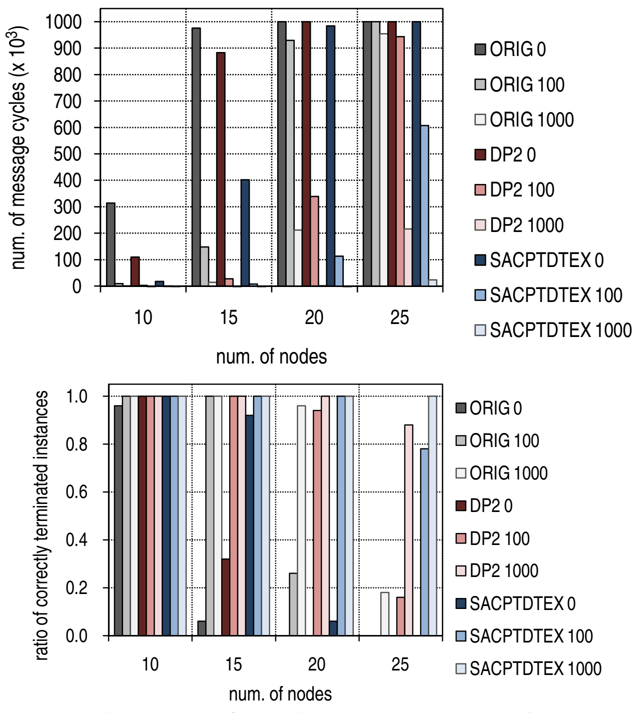
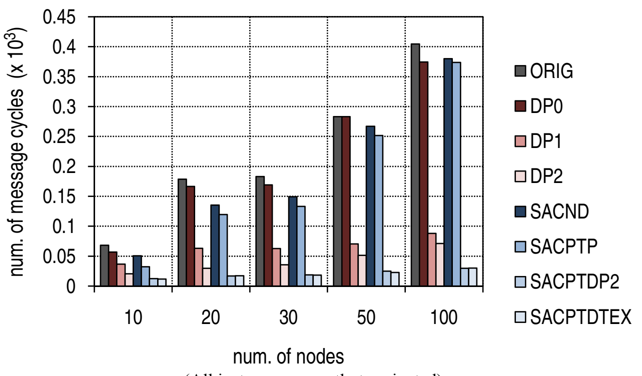


Figure 7: ADOPT with bounded errors (d=3)

posed method is more efficient than DP2 when the estimated lower bound is insufficient for the difference.

4.5 Other cases of problems

Figure 8 shows a result in the case of very low link density and relatively large number of variables. Figure 9 shows a result in the case of $d = 2$ and $|D_i| = 4$. Additionally, the problem also contains unary hard constraints that restrict some values of variables. The number of restricted values for each variable is between 0 and 2 with uniform probability. While the average number of non-restricted values of a variable is 3, the variance of the difficulty of problem instances is different from the one of the case of $|D_i| = 3$. The result shows that soft AC decreases number of message cycles of ADOPT in both cases.



(All instances correctly terminated)

Figure 8: ADOPT (d=1)

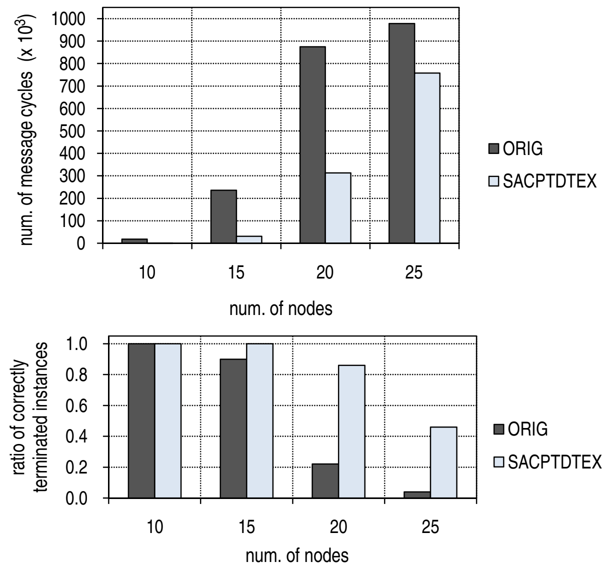


Figure 9: ADOPT (d=2, $|D_i|$=4, unary hard constraints)

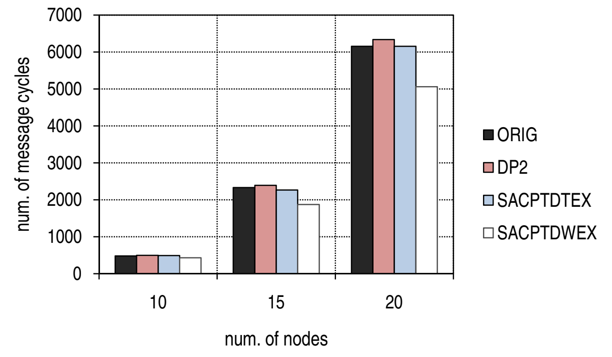


Figure 10: ODPOP (d=2)

4.6 Efficiency for bottom up optimistic search

The proposed method computes the global lower bound in a bottom up manner. Therefore it can be considered that the estimated lower bound is efficient for optimistic search algorithms that are mainly driven in a top down manner. Therefore different type of algorithms that are mainly driven in bottom up manner may not be appropriate for the estimated lower bounds.

We applied DP2 and SACPTDTEX to ODPOP [11] which is an exact search algorithm using the pseudo-tree. Basically, the ODPOP is a version of dynamic programming for the DCOPs [10]. However, it does not compute optimal cost at once. Each node iteratively sends pairs of a partial solution and its cost/utility value in the best first order. Each node has to store all partial solutions that are received from its child nodes. In the worst case, the space complexity is exponential in the induced width of the pseudo-tree [10]. That is in contrast to the memory bounded search of the ADOPT. In this experiment, we evaluated a subset of the problem instances that can be solved with acceptable amounts of computational resources, because our implementation of the ODPOP is not well optimized. Note that the purpose of this experiment is not the comparison between ADOPT and ODPOP. The number of message cycles is shown in Figure 10. The result shows that SACPTDTEX is not efficient in some cases. We evaluated another soft AC preprocessing SACPTDWEX that applies directed soft AC in top down manner. An example of the SACPTDWEX is shown in Figure 3 (c). In this preprocessing, projection and extension are applied to the downside variable/constraint. Each node applies extension for each tree edge. In the extension, cost values are evenly divided among child nodes. The result shows that SACPTDWEX is slightly efficient. However, it is still unclear whether other types of soft arc consistency methods will be more efficient for ODPOP.

5. RELATED WORKS

The proposed preprocessing methods is similar to dynamic programming based preprocessing [1] and search algorithm [10]. These methods compute the lower bounds of costs based on the pseudo-tree in a bottom up manner. All of them directly give the optimal solution in the case that the pseudo-tree has no back edges. In the case that pseudo-tree has back edges, preprocessing methods only estimate the lower bounds, because they are memory bounded algorithms. The search algorithm using dynamic programming always searches exact solutions. However its space complexity is exponential in the induced width of the pseudo-tree. The proposed method is similar to DP2 in [1]. They evaluate global lower bounds for all constraints. The global lower bound is passed up to the root node of the pseudo-tree. However, as shown in 3.1, the direction of the projection for back edges is different between the proposed method and the DP2. Therefore the estimated lower bound is different in some cases.

Unconditional soft AC is not useful when original lower bound values of costs are zero. In such cases of problems we might use some add-on conditional arc consistency techniques.

6. CONCLUSIONS

We proposed an efficient preprocessing method that applies directed soft arc consistency to distributed constraint optimization problems. In the proposed method, directed soft arc consistency is performed along the pseudo-tree in a bottom up manner. Using the pseudo-tree based directed arc consistency enforcemnt, the original problem is converted to an equivalent problem. The equivalent problem is efficiently solved using common search algorithms. The results of our experiments show that the proposed method is more efficient than previous methods that estimate lower bound of costs. Moreover, the proposed method is efficient for approximation algorithms that use bounded errors. Applying more efficient soft arc consistency, more detailed analysis of the effect of the proposed method for various types of algorithms and problems will be included in our future work.

7. ACKNOWLEDGMENTS

This work was supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B), 19300048.

8. REFERENCES

[1] S. M. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1041–1048, 2005.

[2] B. Awerbuch. A new distributed depth-first-search algorithm. *Information Processing Letters*, 20(3):147–150, 1985.

[3] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1–2):199–227, 2004.

[4] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 639–646, 2008.

[5] E. C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(14):755–761, 1985.

[6] R. Junges and A. L. C. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 599–606, 2008.

[7] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, 2004.

[8] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, 2004.

[9] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[10] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *9th International Joint Conferece on Artificial Intelligence*, pages 266–271, 2005.

[11] A. Petcu and B. Faltings. O-dpop: An algorithm for open/distributed constraint optimization. In *National Conference on Artificial Intelligence*, pages 703–708, 2006.

[12] T. Schiex. A note on csp graph parameters. *Technical report 1999/03, INRA*, 1999.

[13] T. Schiex. Arc consistency for soft constraints. In *CP*, pages 411–424, 2000.

[14] W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 591–598, 2008.

[15] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005.

[16] R. Zivan. Anytime local search for distributed constraint optimization. In *Twenty-Third AAAI Conference on Artificial Intelligence*, pages 393–398, 2008.