Complete Distributed Search Algorithm for Cyclic Factor Graphs

Toshihiro Matsui and Hiroshi Matsuo
Department of Computer Science and Engineering, Nagoya Institute of Technology,
Gokiso-cho Showa-ku Nagoya Aichi 466-8555, Japan

Keywords: Max-Sum, Factor-graph, Pseudo-tree, Distributed Constraint Optimization, Multi-agent, Cooperation.

Abstract: Distributed Constraint Optimization Problems (DCOPs) have been studied as fundamental problems in multi-agent systems. The Max-Sum algorithm has been proposed as a solution method for DCOPs. The algorithm is based on factor graphs that consist of two types of nodes for variables and functions. While the Max-Sum is an exact method for acyclic factor-graphs, in the case that the factor graph contains cycles, it is an inexact method that may not converge. In this study, we propose a method that decomposes the cycles based on cross-edged pseudo-trees on factor-graphs. We also present a basic scheme of distributed search algorithms that generalizes complete search algorithms on the constraint graphs and Max-Sum algorithm.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) (Farinelli et al., 2008; Mailler and Lesser, 2004; Modi et al., 2005; Petcu and Faltings, 2005b; Zhang et al., 2005) have been studied as fundamental problems in multi-agent systems. The DCOP is a constraint optimization problem that consists of variables and constraint/objective functions that are distributed among the agents. Multi-agent cooperation problems including distributed resource scheduling, sensor networks and smart grids are represented as DCOPs (Maheswaran et al., 2004; Miller et al., 2012; Zhang et al., 2005).

The Max-Sum algorithm (Farinelli et al., 2008) is a solution method for DCOPs. The algorithm is based on factor graphs that consist of two types of nodes for variables and functions, while traditional methods are based on constraint graphs. The computation of the Max-Sum basically resembles DPOP (Petcu and Faltings, 2005b), which is a dynamic programming based on the pseudo-tree of the constraint graph.

When Max-Sum is applied to acyclic factor graphs, each agent performs as a root node of the tree, and hence the agent computes a global objective function for its variable. Namely, each agent individually (but cooperatively) computes a global objective function for its variable. This nature is considerable in an autonomy of agents since every agents observe the whole system. It also enables agents to determine the optimal assignments of their variables without explicit top-down controls of similar methods (Modi et al., 2005; Petcu and Faltings, 2005b). Therefore, how to employ the techniques of standard complete DCOP algorithms on the framework of factor graph and Max-Sum is an important issue as a basic study. However, in a case where the factor graph contains cycles, the Max-Sum is an inexact method that may not converge, since computation on different paths cannot be separated. For the cyclic factor-graphs, several approaches have been proposed. In bounded Max-Sum (Rogers et al., 2011), a graph is approximated to a minimum (maximum) spanning tree using preprocessing that eliminates the cycles. Then the Max-Sum is applied to the spanning tree as an exact solution method. In Max-sum_AD (Zivan and Peled, 2012), directed acyclic graphs (DAGs) are employed to reduce incorrect computation. In this method, the computation of the Max-Sum is limited in the same direction of the edges of the DAGs. The directions of the edges are alternatively inverted after the computation of Max-Sum.

We present a basic scheme of complete search algorithms that resembles Max-Sum. Since complete solution methods (Modi et al., 2005; Petcu and Faltings, 2005b) based on pseudo-trees have been developed for constraint graphs, similar approaches can be applied into factor-graphs. In this study, we propose a method that decomposes the cycles based on a cross-edged pseudo-tree (Atlas and Decker, 2007) on factor-graphs. With the proposed method, a modified acyclic graph that resembles the original factor-graph is generated. Moreover, we also present a basic scheme of distributed search algorithms that gen-

Matsui T. and Matsuo H..
Complete Distributed Search Algorithm for Cyclic Factor Graphs.
DOI: 10.5220/0004824301840192
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 184-192
ISBN: 978-989-758-016-1
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

eralizes complete search algorithms on the constraint graphs (Modi et al., 2005) and Max-Sum algorithm. While the complete search method is generally inefficient in complex problems with many cycles, our current interest is the generalization of complete search methods on pseudo-trees (Modi et al., 2005) and Max-Sum algorithm.

As shown above, we apply the method based on pseudo-trees to decompose cycles in factor-graphs. In most studies, pseudo-trees including cross-edged pseudo-trees (Atlas and Decker, 2007) are applied to constraint graphs. While the pseudo-trees on factor-graphs resemble the cases on traditional constraint networks, there are several important differences. First, in the case of factor graphs, there is the same separator in both directions on edges of the pseudo-tree. Here, the separator is a set of variables which are shared by two different components of a graph. Therefore, separators are simply computed by a single bottom-up computation on a pseudo-tree. In contrast, in the case of constraint graphs, there are different separators in different directions on edges. Second, since functions are still separated as single nodes in the decomposed tree, it inherently avoids double counting of a function in any path of aggregation of evaluation values. Those properties easily enable the decomposition of cycles. In addition, after the transformation, each agent node simply has its original variable/function. As a result, only neighborhood nodes and separators are modified. A different type of method based on junction trees (Vinyals et al., 2011) has been proposed to transform the graphs of problems. The main purpose of this method is the manipulation to replace variables and functions into different agents.

Our current contribution is that we present a variation of Max-Sum algorithms that is a complete solution method based on pseudo-trees on cyclic factor graphs. Basic effects and overheads of this type of complete algorithms are almost clear since those resemble the cases of algorithms based on constraint networks and pseudo-trees. While the algorithm is complete, its computational cost, memory use and/or message size exponentially grow with the size of separators, also known as the induced width. On the other hand, there are several issues on search algorithms based on Max-Sum since all nodes perform as root nodes and their computation is partially integrated. Therefore, as the first study, we evaluate the possibilities of pruning instead of relatively obvious comparisons with other Max-Sum algorithms. Since sophisticated pruning methods need more works for the case of multiple root nodes, here the proposed method cannot be easily compared with the state-of-art of efficient search methods with a single root node (Yeoh et al., 2008). From another point of view, the proposed generalization will be a base of new approximate algorithms where agents have their own controls as root nodes.

The rest of the paper is organized as follows. In Section 2, we give background for the study. Then, in Section 3, we propose the method to decompose cycles in factor-graphs. Next, we introduce a scheme for distributed search on the decomposed graphs in Section 4. The proposed methods are experimentally evaluated in Section 5. We conclude our study in Section 6.

2 PRELIMINARY

2.1 DCOPs

A distributed constraint optimization problem (DCOP) is defined as follows. A DCOP is defined by (A, X, D, F), where A is a set of agents, X is a set of variables, D is a set of domains of variables, and F is a set of objective functions. A variable $x_n \in X$ takes values from its domain defined by the discrete finite set $D_n \in D$. A function $f_m \in F$ is an objective function defining valuations of a constraint among several variables. Here, f_m represents utility (or cost) values that are maximized (or minimized).

$X_m \subset X$ defines a set of variables that are included in the scope of f_m. $F_n \subset F$ similarly defines a set of functions that include x_n in its scope. f_m is formally defined as $f_m(d_i, \cdots, d_k) : D_i \times \cdots \times D_k \to \mathbb{N}_0$, where $\{x_i, \cdots, x_k\} = X_m$. $f_m(d_i, \cdots, d_k)$ is also simply denoted by $f_m(x_i, \cdots, x_k)$ or $f_m(X_m)$.

The aggregation $F(X)$ of all the objective functions is defined as follows: $F(X) = \sum_m \text{s.t. } {}_{f_m \in F, X_m \subset X} f_m(X_m)$. The goal is to find a globally optimal assignment that maximizes (or minimizes) the value of $F(X)$.

The variables and functions are distributed among the agents in A. Each agent locally knows the set of variables and the set of functions in the initial state. A distributed optimization algorithm is performed to compute the globally optimal solution.

2.2 Factor Graph

The factor graph is a representation of DCOPs. The factor graph consists of variable nodes, function nodes and edges. An edge represents a relationship between a variable and function. Figure 1 shows a traditional constraint network and factor graphs. In the constraint network (a), nodes represent variables and

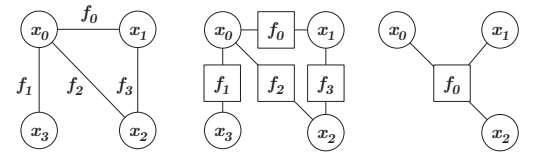

Figure 1: Factor graph.

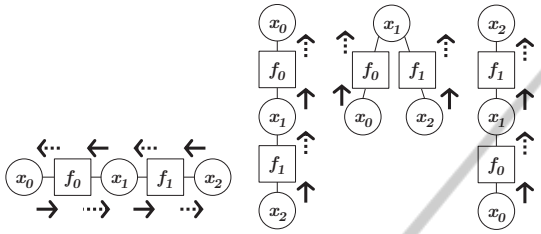

Figure 2: Symmetric computation on factor graph.

edges represent binary functions. The factor graph (b) represents the same problem. As shown in the case of a ternary function (c), the factor graph directly represents n-ary functions. Each agent has at least a function node or a variable node. In the following, we simply use a model in which a node corresponds to an agent. The Max-Sum algorithm is based on the factor graph.

2.3 Max-Sum Algorithm

The Max-Sum algorithm is a solution method for DCOPs. The computation of the method is performed on a factor graph. Each node of the factor graph corresponds to an agent that is also called a variable node or function node. Each node communicates with neighborhood nodes using messages, and computes globally optimal solutions.

The information of a message is an evaluation function for a variable. The function is represented as a table of evaluation values for the variable's values. A node computes a table for each variable that corresponds to each neighborhood node. The table is then sent to the corresponding neighborhood node. Therefore, a node knows evaluation functions for all neighborhood nodes. The evaluation function that is sent from variable node x_n to function node f_m is denoted by $q_{x_n \to f_m}(x_n)$. Similarly, $r_{f_m \to x_n}(x_n)$ denotes the evaluation function sent from function node f_m to variable node x_n. An evaluation function $q_{x_n \to f_m}(x_n)$ that is sent from variable node x_n to function node f_m is represented as follows.

$$q_{x_n \to f_m}(x_n) = 0 + \sum_{m' \text{ s.t. } f_{m'} \in F_n \setminus \{f_m\}} r_{f_{m'} \to x_n}(x_n) \quad (1)$$

Here, F_n denotes the set of neighborhood function nodes of variable node x_n. An evaluation function $r_{f_m \to x_n}(x_n)$ that is sent from function node f_m to variable node x_n is represented as follows.

$$r_{f_m \to x_n}(x_n) = \max_{X_m \setminus \{x_n\}} \left(f_m(X_m) + \sum_{n' \text{ s.t. } x_{n'} \in X_m \setminus \{x_n\}} q_{x_{n'} \to f_m}(x_{n'}) \right) \quad (2)$$

Here, X_m denotes the set of neighborhood variable nodes of function node f_m. $\max_{X_m \setminus \{x_n\}}$ denotes the maximization for all assignments of variables in $X_m \setminus \{x_n\}$. A variable node x_n computes a marginal function that is an evaluation function of x_n. The marginal function $z_n(x_n)$ is represented as $z_n(x_n) = \sum_{m \text{ s.t. } f_m \in F_n} r_{f_m \to x_n}(x_n)$. $z_n(x_n)$ corresponds to global objective values for variable x_n. The value of x_n that maximizes $z_n(x_n)$ is therefore the globally optimal assignment. Each variable node chooses such assignment as the optimal solution.

2.4 Issues on Max-Sum

If the factor-graph is acyclic, the computation of Max-Sum on the tree is considered a set of bottom-up computations that are simultaneously performed for different root nodes. The root node of each computation is a variable node. In each computation, dynamic programming is performed from leaf nodes to the root node. The computation shown in Figure 2(a) can be decomposed into the computations shown in Figure 2(b). There are three trees rooted at variable nodes. While all variable nodes individually compute the global optimal objective value, several nodes share common computation and communication.

This characteristic is interesting because each variable node has authority to determine its optimal assignment based on its knowledge of the global objective value. However, there are multiple optimal solutions, and some tie-break methods are necessary to avoid inconsistent decisions among multiple root nodes. A simple method for the tie-break adds small weight values for each value of variables.

In the case of cyclic factor-graphs, the computation of Max-Sum is incorrect. As shown in the Introduction, several studies address this issue (Rogers et al., 2011; Zivan and Peled, 2012). On the other hand, there are opportunities to construct exact solution methods based on a tree that is equivalent to the original factor graph. In the next section, we present a method to decompose the cycles into a tree.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

3 DECOMPOSITION OF CYCLES

3.1 Representations for Decomposition

In the Max-Sum algorithm, each variable/function node corresponds to an agent. To represent a network of the agents, we explicitly introduce another type of node, agent nodes. Here, we call the graph based on agent nodes the agent-graph. An initial agent-graph directly corresponds to the original factor-graph. On the agent-graph, cycles in the original factor-graph are decomposed. We denote the components of the agent graph as follows. a_i: an agent node. $\mathcal{X}_i$: set of variable nodes that belong to a_i. $\mathcal{F}_i$: set of function nodes that belong to a_i. Nbr_i: set of neighborhood nodes of a_i. In agent a_i of the variable node, $\mathcal{X}_i$ contains one variable while $\mathcal{F}_i$ is an empty set. Similarly, in agent a_i of the function node, $\mathcal{F}_i$ contains one function and $\mathcal{X}_i$ is an empty set. Nbr_i is defined as a set of agent nodes that have variables or functions relating to variables or functions of a_i. Formally, $Nbr_i = \{a_j | \exists x_n \in \mathcal{X}_i, \exists f_m \in \mathcal{F}_j, x_n \in X_m\} \cup \{a_j | \exists f_m \in \mathcal{F}_i, \exists x_n \in \mathcal{X}_j, x_n \in X_m\}$.

In the transformation of the graph, we employ a spanning tree on the agent-graph. Based on the spanning tree, agent nodes in Nbr_i are categorized as follows. a_{prnt_i}: parent node. $Chld_i$: child nodes. $Ancst_i$: ancestor nodes. $Dscnd_i$: descendant nodes.

3.2 Transformation of Graph

To eliminate cycles, graph decomposition methods based on spanning trees (pseudo-trees) are applied. A well-known pseudo-tree is based on depth-first search trees on graphs. With a depth-first search tree, edges in the original graph are categorized into tree-edges, which are edges of the spanning tree, and other back-edges. Such a pseudo-tree is preferable since it does not contain cross-edges between any two nodes in different subtrees. However, the depth of the tree is relatively large. Another problem is that the condition of an exact pseudo-tree limits the types of spanning tree.

On the other hand, when there are edges between different subtrees, the edges are decomposed using the technique of cross-edged pseudo-tree (Atlas and Decker, 2007). In this case, a copy of variables relating to a cross-edge is added. With the copy of the variable, several edges including the edge between different subtrees are modified to meet the condition of the pseudo-tree. In previous studies, decompositions of cycles were mainly discussed on constraint graphs. Here, we apply the cross-edged pseudo-tree on factor-graphs.

As a decomposition of the cycles, each edge of the spanning tree separates the graph into two parts. Both parts are related to a set of variables called separator. The separator is considered as an interface that is commonly used in those parts. Here, let $Sep_{i,j}$ denote the separators between agent-node i and j.

```
1  generate an agent-graph and a spanning tree
2   in a pre-processing.
3  wait for processing of all child nodes a_j in Chld_i.
4  if a_i corresponds variable node x_n in the factor graph
       {
5   let Nbr_i^n denote
6     {a_j | f_m ∈ F_j ∧ f_m ∈ F_n ∧ a_j ∉ (Ancst_i ∪ Dscnd_i)}.
7   if Nbr_i^n is not empty {
8    a_k ← the lowest node in Ancst_i ∩ (∩_{a_j ∈ Nbr_i^n} Ancst_j).
9    add x_n to X_k. store F_n to a_k.
10   store each (f_m, a_j) s.t. f_m ∈ F_j ∧ f_m ∈ F_n to a_k.
11   store end point (x_n, a_i) of equal constraint edge for
         x_n
12    to a_k.
13   for each (f_m, a_j) s.t. f_m ∈ F_j ∧ f_m ∈ F_n{
14    remove (x_n, a_i) from a_j. store (x_n, a_k) to a_j.
15    if a_j ∉ (Ancst_i ∪ Dscnd_i) {
16     remove a_i from Nbr_j. remove a_j from Nbr_i. } }
17   remove x_n from X_i. remove F_n from a_i.
18   erase each (f_m, a_j) s.t. f_m ∈ F_j ∧ f_m ∈ F_n ∧
19    ¬(∃n', n' ≠ n, f_m ∉ F_n') from a_i.
20   store end point (x_n, a_k) of equal constraint edge for
         x_n
21    to a_i. } }
22 for each a_j s.t. a_j ∈ Nbr_i ∧ a_j ∈ Ancst_i \ {a_prnt_i} {
23  remove a_j from Nbr_i. remove a_i from Nbr_j. }

25 Sep_{i,prnt_i} ← ∅. SepTrm_{i,prnt_i} ← ∅.
26 for each child node a_j in Chld_i{
27  Sep_{i,j} ← Sep_{j,i}. Sep_{i,prnt_i} ← Sep_{i,prnt_i} ∪ Sep_{j,i}.
28  SepTrm_{i,prnt_i} ← SepTrm_{i,prnt_i} ∪ SepTrm_{j,i}. }
29 for each (x_n', a_i) in SepTrm_{i,prnt_i} {
30  remove (x_n', a_i) from SepTrm_{i,prnt_i}.
31  remove x_n' from Sep_{i,prnt_i}. }
32 if the copy of x_n have been stored to a_k {
33  add (x_n, a_k) to SepTrm_{i,prnt_i}. add x_n to Sep_{i,prnt_i}. }
34 for each (x_n', a_j) s.t. x_n' ∈ X_i ∧ f_m ∈ X_n' ∧ f_m ∈ F_j ∧
35  a_j ∈ Ancst_i ∧ (a_j is the highest node for the same x_n')
       {
36  add (x_n', a_j) to SepTrm_{i,prnt_i}. add x_n' to Sep_{i,prnt_i}. }
37 for each (x_n', a_j) s.t. f_m ∈ F_i ∧ x_n' ∈ F_m ∧ x_n' ∈ X_j ∧
38  a_j ∈ Ancst_i ∧ (a_j is the highest node for the same x_n')
       {
39  add (x_n', a_j) to SepTrm_{i,prnt_i}. add x_n' to Sep_{i,prnt_i}. }

41 prepare a_i to manage each Sep_{i,j} and
42  a variable/function in the original factor graph.
```

Figure 3: Procedure of transformation (agent node a_i).

The procedure to transform a graph is shown below. The pseudo code and an example of transformation are shown in Figure 3 and 4, respectively.

1) generate an agent-graph based on the original factor-graph (lines 1-2). The graph in Figure 4(a) is an agent graph that corresponds to the original factor graph. Here, nodes are arranged based on a spanning tree rooted at node a_0 that corresponds to x_0.

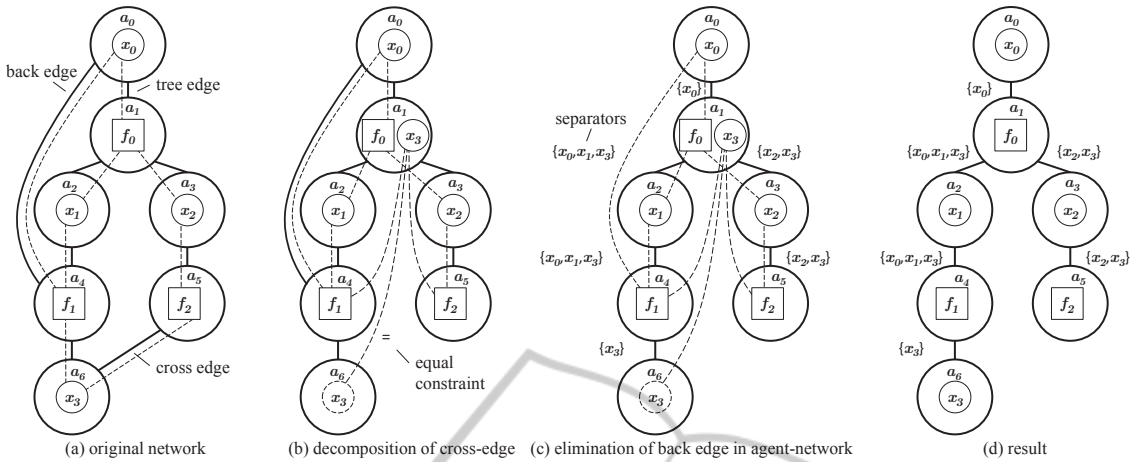

Figure 4: Transformation.

2) in a bottom-up manner, execute computation of agent node a_i as follows (lines 3-39).

3) if a_i has a variable $x_n \in \mathcal{X}_i$ that is an end point of each cross-edge, compute set Nbr_i^n of agent nodes that have another end point of each cross-edge (lines 4-6). Then, generate a copy of x_n into the lowest common ancestor node of nodes in Nbr_i^n (line 8). Eliminate the cross-edges from both agent-graph and the corresponding factor-graph (lines 14-19). Instead of the cross-edges, add edges between the copy of the variable and related function nodes (lines 9, 10 and 14). In addition, add a special edge that represents equal constraint between the copy and original variable x_n (lines 11 and 20). In Figure 4(a), since edge (x_3, f_2) connects the two nodes in different subtrees, this edge is decomposed. Here Nbr_6^n for agent node a_6 and variable x_3 contains a_5. In Figure 4(b), a copy of variable x_3 is placed in a_1 corresponding to f_0, which is the lowest common ancestor node of original nodes x_3 and f_2. Now, edge (x_3, f_2) and (f_1, x_3) is connected to the copy of x_3 by storing information of corresponding variables/functions into a_1, a_4 and a_5. In addition, a special edge that represents equal constraint between the copy and the original variable x_3 (i.e. between a_1 and a_6) is inserted.

4) eliminate back-edges in the agent-network by removing corresponding information of neighborhood nodes (lines 22 and 23). In Figure 4(c), a back-edge between agents a_0 and a_4 is eliminated. Note that this does not affect the computation of separators. Neighborhoods in the agent-network are referred in solution methods.

5) compute separators $Sep_{i,j}$ for each neighborhood node a_j of a_i. Also, information of variables $Sep_{i,prnt_i}$ and $SepTrm_{i,prnt_i}$ are computed for i's parent node $prnt_i$ (lines 25-). $SepTrm_{i,prnt_i}$ stores information of the highest agent node for each variable. Namely, $(x_j, a_k) \in SepTrm_{i,prnt_i}$ represents that agent nodes higher than a_k do not relate with variable x_j. Therefore, such x_j is removed from the separator at agent node a_k (lines 29-31). For child node a_j, $Sep_{i,j}$ is the same as $Sep_{j,i}$ (line 27). Additionally, $Sep_{i,prnt_i}$ and $SepTrm_{i,prnt_i}$ are aggregated for child nodes (line 27 and 28). For the parent, the separator is based on separators of child nodes (line 27) and nodes that have functions/variables neighboring the functions/-variables in a_i (lines 32-39). This computation is performed in a bottom up manner after the elimination of cross-edges and back-edges in the agent-network. Figure 4(c) shows the separators.

6) prepare each agent node so that it manages its original function/variable and separators (lines 41-42). After the generation of separators, each node is modified to have the original variable or function. Figure 4(d) shows the result of the transformation.

In the pseudo code, we assumed that each node can access variables of neighborhood agent nodes for the sake of simplicity. In actual computation, the communication is performed using messages. In addition, a lock-and-commit protocol is necessary for mutual execution between nodes in different subtrees. The resulting agent-graph resembles the original graph, except for cross-edges and separators. An important point is that a separator is common in both directions on an edge of spanning trees. Therefore, each node can simultaneously perform as a root node of this tree, similar to the Max-Sum in the case of trees.

3.2.1 Generalization of Max-Sum

With the separators, computation of Max-Sum is generalized to agent-nodes as follows. Here, $X_{m,n}$ denotes

assignments for variables in separator $Sep_{m,n}$.

$$q_{a_n \to a_m}(X_{n,m}) = \max_{X' \setminus X_{n,m}} \left(0 + \sum_{m' \text{ s.t. } a_{m'} \in Nbr_n \setminus \{a_m\}} r_{a_{m'} \to a_n}(X_{n,m'}) \right) \quad (3)$$

where all $X_{n,m'}$ are compatible with $X_{n,m}$, $X' = \bigcup_{m' \text{ s.t. } a_{m'} \in Nbr_n} X_{n,m'}$.

$$r_{a_m \to a_n}(X_{m,n}) = \max_{X'' \setminus X_{m,n}} \left(f_m(X_m) + \sum_{n' \text{ s.t. } a_{n'} \in Nbr_m \setminus \{a_n\}} q_{a_{n'} \to a_m}(X_{m,n'}) \right) \quad (4)$$

where X_m and all $X_{m,n'}$ are compatible with $X_{m,n}$, $X'' = \bigcup_{n' \text{ s.t. } a_{n'} \in Nbr_m} X_{n,m'}$.

$$z_n(x_n) = \max_{X' \setminus \{x_n\}} \left(\sum_{m \text{ s.t. } a_m \in Nbr_n} r_{a_m \to a_n}(X_{n,m}) \right) \quad (5)$$

where all $X_{n,m}$ are compatible with x_n, $X' = \bigcup_{m' \text{ s.t. } a_{m'} \in Nbr_n} X_{n,m'}$.

As a result of the decomposition of cycles, there is a well-known issue of the induced width on the graph. Namely, when there are many cycles in the graph, the decomposition of the cycles needs a large size of separators. Such separators exponentially increase the number of combinations of assignments. While there are opportunities for efficient/approximate methods for this issue, in this study, we focus on the base-line scheme to decompose cycles in the factor graphs.

4 DISTRIBUTED SEARCH

4.1 Basic Idea

When factor-graphs are trees, the Max-Sum algorithm is exactly a dynamic programming method. For each message, evaluation values for all values of a variable are simultaneously computed and sent. In this computation, only bottom up messages are employed as shown in Figure 2(b). The size of a message depends on the size of the corresponding separator. When we decompose cycles into separators, the size of a message is exponential in the size of the separator.

To reduce the size of each message, tree search can be introduced into the scheme of Max-Sum. In this computation, with top-down messages, each agent node requests evaluation values for each assignment of a separator between a neighboring agent node. The neighboring node then returns the evaluation value for the assignment. If appropriate search strategies and pruning methods are available, there are opportunities to reduce the search processing. On the other hand, in the search processing, evaluation values for assignments that have not been expanded are necessary. Since such an unknown evaluation value is represented with its lower and upper bound values, those boundaries are introduced to the Max-Sum.

We propose a scheme that simultaneously performs solution methods based on tree search and dynamic programming for trees rooted at each agent node. This scheme generalizes complete search methods on pseudo-trees and the Max-Sum algorithm on trees. While the original Max-Sum is defined for maximizing problems, in the following, we prefer minimizing problems since it is easy to employ pruning based on the lower limit cost value 0. Namely, maximization operators in equations (3), (4) and (5) are replaced by minimization operators. While the pruning can be applied to maximization problems, it needs additional computation to estimate the upper limit of objective values. Max-Sum for minimization problems have been addressed in (Zivan and Peled, 2012).

To resolve symmetric solutions, in agent-node of variable x_n, we add a weight value $p_n(x_n)$ to evaluation values. $p_n(x_n)$ represents the preference of value x_n. The values of $p_n(x_n)$ must be sufficiently smaller than the values of original functions. Moreover, summations of the weight values must be different values for all assignments. Here, we use hierarchical values that are implemented additional digits. The summations in equations (3) and (5) are modified to include $p_n(x_n)$.

4.2 Boundary of Cost Values

As shown above, in the processing of tree search, boundaries of evaluation values are necessary to evaluate unknown assignments and to detect termination. Here, we introduce lower limit value 0 and upper limit value ∞ of cost values. Since unknown assignments are evaluated using a pair of lower and upper limit values, cost values are generally represented by lower and upper bound values. When both boundaries take the same value, it is the true value.

For evaluation value $q_{a_n \to a_m}$, its lower bound value $q^{\perp}_{a_n \to a_m}$ is defined as follows. $q^{\perp}_{a_n \to a_m}(X_{m,n}) = q_{a_n \to a_m}(X_{m,n})$ if $q_{a_n \to a_m}(X_{m,n})$ has been received. Otherwise $q^{\perp}_{a_n \to a_m}(X_{m,n}) = 0$. In the case of upper bound value $q^{\top}_{a_n \to a_m}$, ∞ is used as the default value. Similarly, $r^{\perp}_{a_m \to a_n}$ and $r^{\top}_{a_m \to a_n}$ are defined for $r_{a_m \to a_n}$. $z^{\perp}_n(x_n)$ and $z^{\top}_n(x_n)$ are also defined for $z_n(x_n)$. Boundaries of $q_{a_n \to a_m}$ and $r_{a_m \to a_n}$ immediately take the same value in the end nodes of trees. In other nodes, both boundary values are separately aggregated in each direction. Therefore, the boundaries take dif-

ferent values until the true cost value is aggregated. The boundaries basically resemble those shown in ADOPTs (Modi et al., 2005).

4.3 Distributed Search Algorithm

In the tree search processing, each agent node requests evaluation values for each assignment of a separator for a neighboring agent node. For each request, a message that contains the assignment is sent. On the other hand, the neighborhood node then returns the evaluation value for the assignment using a message that contains the assignment and the related cost value.

In the agent node a_n of variable x_n, when lower and upper bounds of $r_{a_m \to a_n}(s)$ take the same value, no more search for a_m and s is necessary. Namely, if $r^{\perp}_{a_m \to a_n}(s) = r^{\top}_{a_m \to a_n}(s)$, then a_n searches for other assignments of $Sep_{n,m}$. Similarly, if $q^{\perp}_{a_n \to a_m}(s) = q^{\top}_{a_n \to a_m}(s)$, then a_m terminates search for a_n and s. Each agent node can simultaneously send different assignments for different neighborhood agent nodes. That absorbs differences in multiple search processing. In the agent node a_n of variable x_n, when lower and upper bounds of marginal function $z^{\top}(x_n)$ take the same value, agent a_n can decide the optimal assignment of x_n. If $z^{\perp}_n(x_n) = z^{\top}_n(x_n)$, then $\mathrm{argmin}_{x_n} z^{\top}_n(x_n)$ is the optimal assignment.

The pseudo code of the distributed search in agent node a_n of variable x_n is shown in Figures 5. After initialization, each agent node requires cost values for assignments of separators sending CONTEXT messages to neighborhood nodes. On the other hand, each agent node sends cost values for the received assignment using COST messages. Here, node a_n stores the received assignments into s_m. Those assignments are called current context. While most of the processing in agent node a_m of function f_m is similar to that shown in Figure 5, only agent nodes a_n of variables x_n compute the optimal assignments x^*_n.

In the processing of lines 21-22 in Figure 5, appropriate search strategies are employed to choose assignment s. In the case of the best-first strategy, s of minimum $r^{\perp}_{a_m \to a_n}(s)$ or $q^{\perp}_{a_n \to a_m}(s)$ is preferred. For the depth-first strategy, an assignment whose boundaries are still open is selected based on an ordering.

4.4 Correctness and Complexity

The proposed method transforms cyclic factor-graphs into equivalent acyclic graphs. In addition, the search method can be considered as a simplified version of ADOPT (Modi et al., 2005) that is a complete algorithm on pseudo-trees while we applied it to factor-

```
1  Main{
2   Initialize.
3   until forever {
4    until receive loop is broken { receive messages. }
5    Maintenance. } }
6  Initialize{
7   for all a_m ∈ Nbr_n { s_m ← ε.
8    for all s ∈ ∏_{x_i ∈ Sep_{n,m}} D_i {
9     r^⊥_{a_m→a_n}(s) ← 0. r^⊤_{a_m→a_n}(s) ← ∞. } }
10  Maintenance. }
11 Receive(CONTEXT, s) from a_m { s_m ← s. }
12 Receive(COST, s, r'^⊥_{a_m→a_n}(s), r'^⊤_{a_m→a_n}(s)) from a_m {
13  r^⊥_{a_m→a_n}(s) ← r'^⊥_{a_m→a_n}(s). r^⊤_{a_m→a_n}(s) ← r'^⊤_{a_m→a_n}(s). }
14 Maintenance{
15  for all s ∈ ∏_{x_i ∈ Sep_{n,m}, a_m ∈ Nbr_n} D_i {
16   update z^⊥_n(s) and z^⊤_n(s). }
17  x*_n ← d s.t. (x_n, d) ∈ argmin_s z^⊤_n(s).
18  for all a_m ∈ Nbr_n {
19   if s_m ≠ ε {
20    send (COST, s_m, q^⊥_{a_n→a_m}(s_m), q^⊤_{a_n→a_m}(s_m)) to a_m.
      }
21   choose s s.t. r^⊥_{a_m→a_n}(s) ≠ r^⊤_{a_m→a_n}(s)
22    based on a search strategy.
23   send (CONTEXT, s) to a_m. }
```

Figure 5: Distributed search (agent node a_n of variable x_n).

graphs. The proposed method is therefore complete and sound.

Since the proposed method expands an assignment for each separator at the same time, in the worst case, its number of iterations depends on $(\prod_{Sep_{n,m}} \prod_{x_j \in Sep_{n,m}} D_j) \times (\prod_{Sep_{m,n}} \prod_{x_k \in Sep_{m,n}} D_k)$. Here $Sep_{n,m}$ and $Sep_{m,n}$ are separators (corresponding to Equations (3) and (4)) on the longest path of the pseudo-tree. While the size of messages is linear in the number of variables in the corresponding separator, in the worst case, the space complexity for stores of received cost values is still exponential in the number of variables in the corresponding separator. There are opportunities to reduce the size of the stores of cost values using additional methods that decompose the separator into iterative search.

5 EVALUATION

The proposed method was experimentally evaluated. There are a number of studies addressing solution methods based on search and dynamic programming with efficient methods on constraint graphs. It is obvious that the performances of the proposed method basically follow the results of previous works. We therefore focused on the characteristics of distributed search processes rooted at different nodes on the same decomposed factor graphs.

We employed factor graphs that consist of variables and ternary functions. The size of domain of each variable is three. Cost values take 0 or 1 based on (maximum) constraint satisfaction problems. For all functions, a ratio of inconsistent pairs of values was set. As spanning trees, we employed breadth-first search trees. The results were averaged for twenty instances. In the search processing, the following search strategies were applied. bst: best first. dpt: depth first. mix: half of agent nodes are bst and others are dpt. In addition, we applied a pruning based on global upper bound cost values (gub). For the processing, each agent propagate its best global upper bound using CONTEXT messages. When the lower bound cost value of an assignment exceeds the gub, the search for the assignment is pruned even if its boundaries are still open.

Table 1 shows the results in the case of cyclic graphs. The result shows that the iteration of bst varies widely in comparison to that of dpt. While mix employs bst and dpt, it takes more iterations on the average. It reveals that there were a mismatch between different strategies. We also evaluated the number of assignment whose boundaries did not converge. The ratio of assignments with the open boundaries is relatively large in the case of sparsely constrained problems. Namely, there were effects of pruning. Additionally, gub increased the ratio of open assignments. It can be considered that several agents help other agents through the gub. Table 2 shows the case that factor graphs are trees. Compared with these results, we can see that the effect of pruning is relatively large in the case of cyclic graphs.

Table 1: Cyclic graph.
(21 variables, 13 3-ary functions, (max) CSP)

inconsistent pairs	alg.	iteration			#open asgn.	#closed asgn.	rate of open asgn.
		min	max	ave			
0	all	11	17	14	2089	166	92.64%
2014/01/03	bst	105	2188	1000	168	2086	7.47%
	bst-gub	105	2188	1000	491	1764	21.77%
	dpt	138	2165	986	18	2237	0.79%
	dpt-gub	112	2111	943	267	1987	11.86%
	mix	138	2557	1150	25	2230	1.12%
	mix-gub	112	2557	1071	337	1917	14.96%
2014/02/03	bst	95	2293	1007	160	2095	7.11%
	bst-gub	95	2293	1007	478	1777	21.20%
	dpt	146	2157	995	6	2248	0.29%
	dpt-gub	122	2151	968	166	2089	7.35%
	mix	134	2659	1238	20	2235	0.88%
	mix-gub	122	2609	1178	213	2042	9.46%
1	bst/bst-gub	164	2526	1190	3	2252	0.13%
	dpt/dpt-gub	148	2175	1006	0.5	2254	0.02%
	mix/mix-gub	176	2582	1289	3	2252	0.14%

Table 2: Tree.
(31 variables, 15 3-ary functions, (max) CSP).

inconsistent pairs	alg.	iteration			#open asgn.	#closed asgn.	rate of open asgn.
		min	max	ave			
0	all	15	17	16	42	228	15.67%
2014/01/03	bst	33	57	45	3	267	0.94%
	bst-gub	33	57	45	13	257	4.78%
	dpt	33	49	42	6	264	2.26%
	dpt-gub	33	49	42	10	260	3.54%
	mix	39	57	46	5	265	1.81%
	mix-gub	39	57	46	11	259	4.04%
2014/02/03	bst	35	51	42	2	268	0.78%
	bst-gub	35	51	42	35	235	12.87%
	dpt	35	49	42	3	268	0.93%
	dpt-gub	35	49	42	13	257	4.65%
	mix	39	53	47	2	268	0.72%
	mix-gub	39	53	46	20	250	7.46%
1	bst/bst-gub	44	62	51	0	270	0
	dpt/dpt-gub	39	45	42	0	270	0
	mix/mix-gub	43	59	51	0	270	0

6 CONCLUSIONS

We proposed a method that decomposes the cycles in factor-graphs based on a cross-edged pseudo-tree. The proposed method generates a modified acyclic graph that resembles the original factor-graph. In addition, we presented a scheme of distributed search algorithm that generalizes complete search algorithms on the constraint graphs and Max-Sum algorithm.

Detailed analysis of the distributed search on the modified graphs, and a more sophisticated solution method based on the proposed scheme will be included in future works. To improve the proposed method, there are several directions of the research. An approach applies efficient pruning methods in existing search methods for DCOPs (Yeoh et al., 2008). In this case, the property that all agents simultaneously perform as root nodes and intermediate/leaf nodes have to be considered to manage multiple pruning processes. Another approach simply employs inference methods instead of search. This approach will be technically easier than search methods. In the case, several approximation (Rogers et al., 2011; Petcu and Faltings, 2005a) and filtering methods reduce computation and the number of messages will be applied. In addition, there are opportunities to partially integrate search and inference methods.

ACKNOWLEDGEMENTS

This work was supported in part by KAKENHI, a Grant-in-Aid for Scientific Research (C), 25330257 and the Artificial Intelligence Research Promotion Foundation.

REFERENCES

Atlas, J. and Decker, K. (2007). A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In *AAMAS07*, pages 111:1–111:8.

Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS08*, pages 639–646.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., and Varakantham, P. (2004). Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *AAMAS04*, pages 310–317.

Mailler, R. and Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS04*, pages 438–445.

Miller, S., Ramchurn, S. D., and Rogers, A. (2012). Optimal decentralised dispatch of embedded generation in the smart grid. In *AAMAS12*, volume 1, pages 281–288.

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

Petcu, A. and Faltings, B. (2005a). A-DPOP: Approximations in distributed optimization. In *CP05 - workshop on Distributed and Speculative Constraint Processing, DSCP*.

Petcu, A. and Faltings, B. (2005b). A scalable method for multiagent constraint optimization. In *IJCAI05*, pages 266–271.

Rogers, A., Farinelli, A., Stranders, R., and Jennings, N. R. (2011). Bounded approximate decentralised coordination via the Max-Sum algorithm. *Artificial Intelligence*, 175(2):730–759.

Vinyals, M., Rodriguez-Aguilar, J. A., and Cerquides, J. (2011). Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, 22(3):439–464.

Yeoh, W., Felner, A., and Koenig, S. (2008). BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. In *AAMAS08*, pages 591–598.

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

Zivan, R. and Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *AAMAS12*, volume 1, pages 265–272.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS