A Study for Representations of Distributed Cooperative Search Algorithms based on Pseudo-trees

TOSHIHIRO MATSUI and HIROSHI MATSUO
Nagoya Institute of Technology
Gokiso-Cho Showa-ku Nagoya Aichi 466-8555
JAPAN
{matsui.t, mastuo}@nitech.ac.jp

Abstract: Distributed search is an important processing to solve problems in distributed cooperative systems. As fundamental research of distributed cooperative problem solving, distributed constraint optimization problems (DCOPs) have been studied. For the DCOPs, exact distributed search algorithms that are based on pseudo-trees have been proposed. In the search algorithms, tree-searches and dynamic programming methods are performed using an order of variables defined by the pseudo-trees. However, analyzing the behaviors of search algorithms is relatively difficult due to the distributed processing. In this study, we show a representation of the essential computation contained in search algorithms based on pseudo-trees. Using the representations of the computation, a basic algorithm that contains a relatively clear data-flow of the processing is composed. This representation can be considered a basis of analyzing the dependency of the processing and deriving actual distributed algorithms.

Key–Words: Distributed, Cooperative, Optimization, Search, Pseudo-tree

1 Introduction

Distributed search is an important processing to solve problems in distributed cooperative systems. As fundamental research of distributed cooperative problem solving, distributed constraint optimization problems (DCOPs) have been studied. In the formalization of DCOPs, the states of agents and the relationships between them are represented as a discrete optimization problem. Distributed search algorithms are applied to solve the problems. The main interest of DCOPs are fundamental optimization problems and distributed search algorithms in distributed cooperative systems.

For the DCOPs, exact search algorithms that are based on pseudo-trees have been proposed [2, 3, 4, 5, 7, 9, 8]. A pseudo-tree is a graph structure on constraint networks. In search algorithms, tree-searches and dynamic programming methods are performed using an order of variables that is defined by the pseudo-trees. These algorithms are designed as distributed processing exchanging messages.

Although most previous studies represent sophisticated algorithms, the algorithms are relatively complicated due to the distributed processing. Especially, analyzing the data-flow of asynchronous algorithms are not easy. Moreover, to handle asynchrony, the algorithms employ a number of techniques which are highly integrated with essential processing.

On the other hand, the algorithms are based on essential computation on pseudo-trees. Therefore, clarifying the computation as a basis of the algorithms is important. Such representation can suggest how to derive the actual algorithms.

In this study, we show a representation of the essential computation contained in search algorithms based on pseudo-trees. Using the representations of the computation, a basic algorithm that contains a relatively clear data-flow of the processing is composed. This representation can be considered as a basis of analyzing dependency in the processing and deriving actual distributed algorithms.

The rest of our paper is organized as follows. In Section 2, the background of the study is shown. The essential computation of search algorithms based on pseudo-trees is shown in Section 3. An example of a search algorithm with the computation is shown in Section 4. The algorithm is experimentally evaluated in Section 5. In Section 6, several considerations about related works are shown. In Section 7, we conclude our study.
2 Backgrounds

2.1 Distributed constraint optimization problems

The following is a fundamental definition of distributed constraint optimization problems. A problem is defined by set \( A \) of agents, set \( X \) of variables, set \( C \) of binary constraints, and set \( F \) of binary functions. Agent \( i \) has its own variable, \( x_i \), that 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 assignment \( \{(x_i, d_i), (x_j, d_j)\} \) is defined by binary function \( f_{i,j}(d_i, d_j) : D_i \times D_j \to \mathbb{N}_0 \). The goal is to find global optimal solution \( A \) that minimizes the global cost function: \( \sum_{f_{i,j} \in F} \{(x_i, d_i), (x_j, d_j)\} \subseteq A f_{i,j}(d_i, d_j) \).

Agent \( i \) knows the constraints and the cost functions that are related to \( x_i \). The search process to find the optimal solution is represented as a distributed algorithm based on message communication between agents.

2.2 Pseudo-trees

A pseudo-tree [3, 6], which is a graph structure that defines a partial order on variables, is based on a spanning tree of constraint networks. An example of a pseudo-tree is shown in Figure 1. The pseudo-tree (b) corresponds to the constraint network (a) in the figure. In the pseudo-tree, the edges of the constraint network are categorized into two types. The edges of the spanning tree are tree edges. The other edges are back edges. The tree edges represent the partial order relation between the two variables. We consider the tree edges of the pseudo-tree the edges of the corresponding spanning tree. Also, nodes, variables, and agents may not be strictly distinguished. The following notations are used.

- \( pr_i \): parent variable of \( x_i \).
- \( Ch_i \): set of child variables of \( x_i \).

\( Nbu_i \): partial set of ancestor variables of \( x_i \). The variables in \( Nbu_i \) are related to \( x_i \) by constraints.

\( PP_i \): partial set of ancestor variables of \( x_i \). Let \( x_k \) denote a variable in \( PP_i \). For at least one variable \( x_j \) that is contained in the pseudo-tree rooted at \( x_i \), \( x_k \) has relationship \( x_k \in Nbu_j \).

Since there are no back edges among different subtrees, search processing can be performed in parallel for them.

2.3 Distributed search based on pseudo trees

A number of distributed search algorithms based on the pseudo-trees have been proposed for DCOPs. These algorithms are generally based on dynamic programming and branch-and-bound. Dynamic programming [3] computes the optimal cost values of subtrees from leaf agents to a root agent. Then the optimal assignments are decided from a root agent to leaf agents. On the other hand, the size of the memory and the messages are exponential to the induced width of the pseudo-trees because each agent simultaneously computes the cost values for all assignments of the variables contained in \( PP_i \). In several algorithms [4, 5] computation of the optimal cost values is divided into iterative processing.

In memory-bounded methods based on branch-and-bound and on dynamic programming [2], one current assignment is processed in each agent. Such an algorithm performs iterative searches between agents. Several efficient methods reduce the iterations [7, 9].

3 Computation on pseudo-trees

In this section, we represent several computations on pseudo-trees. Those computations have been obviously or implicitly shown in previous studies. Our aim is to clarify parts of the algorithms.

3.1 Computation of partial solution

The computation of cost values depends on partial solutions. Because a pseudo-tree defines an order of the variables, partial solutions are built in a top-down manner. Here, we assume that each value \( d_i \) of variable \( x_i \) has already been determined. Using the assignment of each \( (i, d_i) \) of \( x_i \), partial solution \( t_i \) of \( i \)'s ancestor variables is computed in the node of \( pr_i \) as follows.

\[
t_i = \begin{cases} 
\emptyset & \text{if } x_i \text{ is the root variable} \\
pr_i \cup \{(pr_i, d_{pr_i})\} & \text{otherwise} 
\end{cases}
\]

(1)
Actually, several computations of cost values depend on partial solution $s_i$ of $PP_i$. The partial solution is computed as follows.

\[
s_i = \begin{cases} 
\phi & \text{if } x_i \text{ is the root variable} \\
\bigcup_{(j,d) \in s_{pr}, j \in PP_i \{ (j,d) \}} \bigcup_{(pr_i,d_{pr})} & \text{otherwise}
\end{cases}
\]

In both cases, if $x_i$ is the root variable, the partial solutions are empty.

### 3.2 Bottom-up computation of cost values

The essential computation of cost values is recursively represented as a bottom-up computation. Below, we assume that nodes have already received both variables’ values and cost values from other nodes. The computation in the node of $x_i$ is based on partial solution $s_i$ of $PP_i$.

Local cost $\delta_i(s_i \cup \{(i,d)\})$ for partial solution $s_i$ and value $d$ of variable $x_i$ is defined as follows.

\[
\delta_i(s_i \cup \{(i,d)\}) = \sum_{j \in N_{bu}, (j,d) \in s_i} f_{i,j}(d, d_j)
\]  

(3)

Optimal cost $g^*(s_i)$ for partial solution $s_i$ and the subtree rooted at $x_i$ is recursively defined as follows.

\[
g^*(s_i) = \min_{d \in D_i} g_i(s_i \cup \{(i,d)\})
\]  

(4)

\[
g_i(s_i \cup \{(i,d)\}) = \delta_i(s_i \cup \{(i,d)\}) + \sum_{j \in C_{hi}, s_j \subseteq (s_i \cup \{(i,d)\})} g_j^*(s_j)
\]  

(5)

### 3.3 Top-down computation of cost values

In addition to the bottom-up computation shown in 3.2, the cost value is computed in a top-down manner. The computation in node of $x_i$ is based on partial solution $t_i$ of $i$’s ancestor nodes. By the computation, the cost values for $i$’s child node of $x_k$ are computed in the node of $x_i$ as follows.

\[
h_k^*(t_i \cup \{(i,d)\}) = h_{pr_i}^*(t_i) + g^*(s_i) + \sum_{j \in C_{hi}\backslash \{k\}, s_j \subseteq (s_i \cup \{(i,d)\})} g_j^*(s_j)
\]  

(6)

where $s_i \subseteq t_i$

$h_k^*$ represents the total cost value, except for the subtree rooted at the node of $x_k$.

Using the top-down cost values, total cost value $e_i^*(t_i)$ for all variables is computed in the node of $x_i$ as follows.

\[
e_i^*(t_i) = \min_{d \in D_i} e_i(t_i \cup \{(i,d)\})
\]  

(7)

\[
e_i(t_i \cup \{(i,d)\}) = h_{pr_i}^*(t_i) + g_i(s_i \cup \{(i,d)\})
\]  

where $s_i \subseteq t_i$

(8)

If $x_i$ is the root node, $t_i$ and $s_i$ are empty. In that case, $h_{pr_i}^*(t_i) = 0$. Therefore, in the root node, $e_i^*$ and $g_i^*$ take the same value.

\[
e_i^*(\phi) = g_i^*(\phi)
\]  

(9)

They represent the globally optimal cost value.

Clearly the globally optimal cost value is computed by $g^*$.

### 3.4 Computation of the optimal solution

Using the globally optimal cost value, the optimal solution is computed. Here, we assume that the cost computation has been completed. When the optimal cost is computed in the root node, its optimal assignment is immediately determined.

The computation of the optimal solution is represented as a recursive computation in a top-down manner. When optimal assignment $s_i^*$ of $PP_i$ is determined, optimal value $d_i^*$ in node of $x_i$ is represented as follows.

\[
d_i^* = \arg\min_{d \in D_i} g_i(s_i^* \cup \{(i,d)\})
\]  

(10)

Assignment $s_i^*$ is computed using the optimal values of the variables in $PP_i$.

\[
s_i^* = (\cup_{(j,d) \in s_{pr}, j \in PP_i \{ (j,d) \}} \cup \{(pr_i, d_{pr})\})
\]  

(11)

That is based on the computation of partial solutions shown in 3.1.

The above computation is represented using the top-down costs as follows. Here, $t_i^*$ represents an optimal assignment of $x_i$’s ancestors.

\[
d_i^* = \arg\min_{d \in D_i} e_i(t_i^* \cup \{(i,d)\})
\]  

(12)

\[
d_i^* = \arg\min_{d \in D_i} (h_{pr_i}^*(t_i^*) + g_i(s_i^* \cup \{(i,d)\}))
\]  

(13)

\[
d_i^* = \arg\min_{d \in D_i} g_i(s_i^* \cup \{(i,d)\})
\]  

where $s_i^* \subseteq t_i^*$

(14)

\[
t_i^* = t_{pr_i}^* \cup \{(pr_i, d_{pr})\}
\]  

(15)
3.5 Boundaries of cost values

In actual computation, some cost values may remain unknown. In such cases, the upper and lower limit values are used as default values. In this work, we use 0 and \( \infty \) as the lower and upper limits. These default values separate a cost value into upper and lower bound values.

Let \( \underline{v} \) and \( \overline{v} \) denote the lower and upper bounds of \( v \). Clearly, the following relation exists between both values.

\[
\underline{v} \leq \overline{v}
\]

(14)

When \( \underline{v} = \overline{v} \), their values are the true value of \( v \). The comparison of two values \( v, v' \) is generalized as follows.

\[
\overline{v} \leq v' \Rightarrow v \leq v'
\]

(15)

In the following case, both values are equal. Therefore, the value is the true value.

\[
\overline{v} \leq v' \land \overline{v'} \leq \underline{v} \Rightarrow v = v'
\]

(16)

If the above conditions are not satisfied, the relation is indeterminable.

Minimizing is represented using a comparison for each pair of values.

\[
v^* = \min_{v \in V} v \iff \forall v' \in V \setminus \{v^*\}, v^* \leq v'
\]

(17)

That is generalized using boundaries as follows.

\[
v^* = \min_{v \in V} v \iff \forall v' \in V \setminus \{v^*\}, \overline{v} \leq v' \leq \underline{v}'
\]

(18)

If any pairs of values do not satisfy the condition in the comparison, the minimum value is indeterminable. On the other hand, some values may be determined that are not the minimum value.

Similarly, argmin is generalized using \( \min \) that is generalized for boundaries. When the minimum value is indeterminable, the value that gives the minimum value is also indeterminable.

3.6 Computation of cost values with boundaries

Using the boundaries of cost values, the computation shown in Sections 3.2 and 3.3 is separated to lower and upper bounds. \( g_i^*, g_i, \delta_i, \epsilon_i, \epsilon_i^* \) and \( h_i \) are replaced by \( g_i^*, g_i, \delta_i, \epsilon_i, \epsilon_i^*, h_i, g_i^*, \overline{g}_i, \overline{g}_i, \overline{g}_i, \epsilon_i^*, \overline{\epsilon}_i \) and \( \overline{h}_i \). The lower bound values and upper bound values are never added.

3.7 Best upper bound of global cost value

When an upper bound of a cost value is not \( \infty \), the value is the exact summation for all cost functions contained in the computation. In that case, if the lower bound of the cost value is simultaneously computed, both boundaries take the same value.

When the upper bound value \( g_i^*(\phi) \) of \( g_i^*(\phi) \) shown in Section 3.2 is not \( \infty \) in the root node, it represents a summation of the total cost values in the whole system for an assignment.

On the other hand, when the upper bound value \( e^*(t_i) \) of \( e^*(t_i) \) shown in Section 3.3 is not \( \infty \), it also represents a summation of the total cost values in the whole system for an assignment. This contains the case of the root node.

The total cost values may give the best upper bound of the globally optimal cost value. Let \( gub_i \) denote the best upper bound of the global cost value that is currently known by node of \( x_i \). When \( gub_i^*(\phi) \) and previous value \( gub_i^* \) of \( gub_i \) are given in the root node, \( gub_i \) is computed as

\[
gub_i = \min(gub_i^*, \overline{g}_i^*(\phi))
\]

(19)

Similarly, when \( e^*(t_i) \) and previous value \( gub_i^* \) of \( gub_i \) are given, \( gub_i \) is computed as

\[
gub_i = \min(gub_i^*, e^*(t_i))
\]

(20)

In the following, we prefer the case of \( e^*(t_i) \) for general representation.

3.8 Pruning

Because of the monotonicity in the summation of the cost values, the lower bound of the globally optimal cost values does not exceed the upper bound of the globally optimal cost values.

The best upper bound of the globally optimal cost is greater than or equal to the globally optimal cost. Therefore, partial solutions whose lower bounds of cost values exceed the best upper bound of the globally optimal cost are not part of the globally optimal solution. The condition of the pruning is shown in Section 3.12, because it applied when an assignment of the variable is determined.

3.9 Computation of optimal solution with boundaries

When the boundaries of cost values are applied, the optimal cost value may not be determined. In such cases, the computation of the optimal solution is not performed. On the other hand, search processing should be performed to obtain the optimal cost. Since
the search and termination strategies are related to the computation of the optimal solution, the details are discussed in later sections.

3.10 Top-down computation for pseudo parents

As a representation that simplifies part of the partial solution, the top-down computation of cost values in node of $x_i$ can be calculated for $PP_i$ instead of $i$’s ancestor nodes.

In this representation, partial solution $s_i$ may implicitly depend on more than one $t_i$. Therefore, the top-down cost values for $s_i$ may not be unique.

Although the simplification is slightly confusing, there are no influences on the bottom-up computation of cost values. Essentially, the bottom-up computation only depends on partial solutions for $PP_i$. As long as this characteristic is kept, the computation is correct.

A merit of the simplification is that the values of several ancestor nodes are not leaked to their descendant nodes.

The computation shown in Section 3.3 is modified as follows. Here $s^{'}_k \subseteq s_i \cup \{(i, d)\}$ represents a partial solution of $PP_k$.

$$h^{pr}_i(s^{'}_k) = h^{pr}_i(s_i) + \sum_{j \in Ch\setminus\{k\}, s_j \subseteq (s_i \cup \{(i, d)\})} g^*_j(s_j)$$

$$\delta_i(s_i \cup \{(i, d)\}) + \sum_{j \in Ch\setminus\{k\}, s_j \subseteq (s_i \cup \{(i, d)\})} g^*_j(s_j)$$

$$e^*_i(s_i) = \min_{d \in D_i} e^*_i(s_i \cup \{(i, d)\})$$

$$e^*_i(s_i \cup \{(i, d)\}) = h^{pr}_i(s_i) + g^*_i(s_i)$$

3.11 Contexts

The computation shown in the above sections is represented as a declarative description for the total problem. On the other hand, actual search processing is simultaneously performed for a part of the problem. The selection of the current partial solution is mainly composed by selecting an assignment in each node and distributing the assignments to other nodes.

In Section 3.1, the computation to build partial solutions is shown. Although the computation to distribute the assignments built partial solutions, several variations that are modifications of Expression 2 can be used. In this study, we simply use the definition shown in Section 3.1.

The current partial solution is called context. It is possible to handle multiple contexts at the same time using several additional methods. Below we focus on the case when a single context is used.

3.12 Selection of variable values

As shown in 3.11, each node decides its own assignment of the variable to build partial solutions. Essentially, the situation for selecting the assignment is categorized into two cases. One is the computation of the optimal assignment. The other is the computation of the cost values. If the optimal cost value is already known, the computation of the optimal assignment should be performed. Otherwise, the computation of the cost values should be performed. Therefore, both computations are integrated.

In the computation of the cost values, several strategies are available. In this study, a depth-first search is applied as a basic method. The total condition of the selections is shown as follows.

1. If $g^*(s_i) = \overline{g}^*(s_i)$ then optimal assignment for $s_i$ can be determined. Value $d$ such that $\overline{g}(s_i \cup \{(i, d)\}) = g^*(s_i)$ is selected.

2. Otherwise, value $d$ such that $g_i^*(s_i \cup \{(i, d)\}) = \overline{g}_i(s_i \cup \{(i, d)\})$ is selected to continue the search. Additionally, the search is pruned using two conditions. When value $d$ satisfies the condition $g_i^*(s_i \cup \{(i, d)\}) ≥ \overline{g}^*(s_i)$ or $h^*_i(s_i) + g^*_i(s_i) ≥ gub_i$, the value can be pruned.

When multiple values of the variable can be selected as the current assignment, the tie is broken using a value ordering.

4 Representation of Search Algorithm considering Data-flow

In this section, we compose a basic algorithm based on the computation shown in the previous section.

4.1 Pull type of distributed processing

Our purpose is to clarify the dependencies between parts of the computation. Therefore, we assume that each node can refer to the values of other nodes. It can be considered that the algorithm uses a pull type of distributed processing instead of push processing. Additionally, several values are cached using variables as shown below.
4.2 Consideration of logical time

The actual search processing is performed as sequences of computation in parallel. We model the processing with globally synchronized logical time. Values that are referred from other computations are tagged using the logical time.

Each node computes the current values. On the other hand, the nodes refer to previous values. In addition, if the values are contained in the same node, the current values can be referred to in appropriate order of the computation. These limitations are necessary to avoid wrong cycles in computation. The values are represented as single assignment variables.

Actually, the computations of the current values are represented using the current and previous values. Therefore, old values that are never referred to are eliminated. If values of assignments are not given in a time, those values are represented as \( \epsilon \). As shown in previous sections, unavailable cost values are represented using upper and lower bound values. In the node of \( x_i \), the following values are represented as single assignment variables.

- \( s_i \): a partial solution of \( PP_i \).
- \( g^*_i, \bar{g}^*_i \): lower and upper bounds of bottom-up cost values for \( s_i \).
- \( h_i, \bar{h}_i \): lower and upper bounds of top-down cost values for \( s_i \).
- \( s^i_{j,d} \): a cached value of \( s_j \). Here \( x_j \) is a child node of \( x_i \). \( s^i \) contains assignment \((i,d)\) and several assignments in \( s_i \) for \( PP_i \). This cache is necessary to avoid the infinite loop of the search. This method is used in [2] and similar solvers.
- \( g^{\bar{v}}_{j,d}, \bar{g}^{\bar{v}}_{j,d} \): cached values of \( g^*_j, \bar{g}^*_j \) that are related to \( s^i_{j,d} \).
- \( gub_i \): a upper bound of the globally optimal cost value.
- \( d_i \): a current assignment of \( x_i \).
- \( s_j \): a partial solution of \( PP_j \). Here \( x_j \) is a child node of \( x_i \). This value is set by the node of \( x_i \) and referred by the node of \( x_j \).
- \( h^i_j, \bar{h}^i_j \): top-down cost values that are related to \( s^i_j \).

\[
\text{figure 2: An example of algorithms}
\]

4.3 Example of algorithms

The computation of the current values in the node of \( x_i \) is shown in Figure 2. Here \( v^o \) denotes the previous value of \( v \). The computation is performed each time.

In the lines 3 and 4, partial solution \( s_i \) and top-down cost values are propagated from \( x_i \)’s parent node. In the lines from 6 to 13, bottom-up cost values are propagated from \( x_j \)’s child nodes. Moreover, those values are maintained as cached values. In the
The algorithm shown in Figure 2 is experimentally evaluated. The algorithm is applied to five ternary variables and seven constraints/functions that take cost values between 0 and 10. The pseudo tree of the problem is shown in Figure 4. Because one of our interest is the verification of the representation of the algorithms, we show an execution example in Figure 3 as the first result.

In the first steps, assignments and cost values are not computed. Therefore default cost values are mainly used. After partial solutions are propagated in a top-down manner, cost values are summed up. In step 35, boundaries of the globally optimal cost are closed in the root node. Then the optimal solution are determined. In the final steps several re-computations occur because of limitation on the number of cached values.

Eventually, the processing is converged. Boundaries of both top-down and bottom-up costs are closed in every nodes. The sum of the true values of top-down and bottom-up costs equal the globally optimal cost value. The example shows a complete execution of the computation of the optimal cost and the optimal solution.
6 Related works

In this section, we show several considerations of search algorithms that have been proposed in previous studies. To represent the following methods using the proposed approach of the representation, several generalizations are necessary.

In dynamic programming [3], the top-down computation of the cost values is not applied. On the other hand, the bottom-up computation of the cost values is simultaneously performed for all partial solutions of pseudo parents. In the computation of cost values no pruning is performed. The computation of contexts is implicitly performed in the preprocessing of the search. The modification is based on the knowledge that all contexts are selected at the same time.

In methods based on tree-search [2, 7], computations are performed for one context at the same time. However, the contexts are distributed using multiple paths. Therefore, additional techniques are applied to merge the duplicated contexts.

In several methods that are fully/partially based on dynamic programming [2, 3], the computation of the optimal solution is explicitly distinguished as the termination of the algorithm. In a different method [7], the computation of the optimal solution is implicitly performed. The search algorithm reaches quiescence with the optimal solution.

In the methods that apply a caching scheme [9], contexts and relative cost values that were previously computed are stored for future use.

Another important problems are the analysis of timings and conversion to push type algorithms for message passing.

7 Conclusion

This study showed the representation of the essential computation contained in search algorithms based on pseudo-trees. Using the representations of the computation, a basic algorithm that contains a relatively clear data-flow of the processing was composed. This representation can be considered as a basis of analyzing dependencies in the processing and deriving the actual distributed algorithms.

Our future works include more generic representations of the algorithms and methodologies to generate asynchronous distributed processing.

Acknowledgements: This work was supported in part by a Grant-in-Aid for Young Scientists (B), 22700144 and the Kayamori Foundation of Informational Science Advancement.

References:


