Resource Constrained Distributed Constraint Optimization with Virtual Variables

Toshihiro Matsui and Hiroshi Matsuo
Nagoya Institute of Technology
{matsui.t | matsuo}@nitech.ac.jp

Katsutoshi Hirayama
Kobe University
hirayama@maritime.kobe-u.ac.jp

Marius Silaghi
Florida Institute of Technology
msilaghi@fit.edu

Makoto Yokoo
Kyusyu University
yokoo@is.kyushu-u.ac.jp

Abstract
Cooperative problem solving with resource constraints are important in practical multi-agent systems. Resource constraints are necessary to handle practical problems including distributed task scheduling with limited resource availability. A dedicated framework called Resource Constrained DCOP (RCDCOP) has recently been proposed. RCDCOP models objective functions and resource constraints separately. A resource constraint is an n-ary constraint that represents the limit on the number of resources of a given type available to agents. Previous research addressing RCDCOPs employs the Adopt algorithm, which is an efficient solver for DCOPs. An important graph structure for Adopt is the pseudo-tree for constraint networks. A pseudo-tree implies a partial ordering of variables. In this variable ordering, n-ary constrained variables are placed on a single path of the tree. Therefore, resource constraints that have large arity augment the depth of the pseudo-tree. This also reduces the parallelism, and therefore the efficiency of Adopt. In this paper we propose another version of the Adopt algorithm for RCDCOP using a pseudo-tree that is generated ignoring resource constraints. The proposed method reduces the previous limitations in the construction of RCDCOP pseudo-trees. The key ideas of our work are as follows: (i) The pseudo-tree is generated ignoring resource constraints. (ii) Virtual variables are introduced, representing the usage of resources. These virtual variables are used to share resources among sub-trees. However, the addition of virtual variables increases the search space. To handle this problem, influence of placement of virtual variables/resources constraints in the pseudo tree is considered. Moreover the search is pruned using the bounds defined by the resource constraints if possible. These ideas are used to extend Adopt. The efficiency of our technique depends on the class of problems being considered, and we describe the obtained experimental results.

Introduction
Cooperative problem solving with resource constraint is important in practical multi-agent systems. Resource constraints are necessary to handle practical problems including distributed task scheduling with limited resource availability. As a fundamental formalism for multi-agent cooperation the Distributed Constraint Optimization Problem (DCOP)
Adopt (Modi et al. 2005) is an efficient distributed constraint optimization algorithm. The processing of Adopt consists of two phases as follows.

- **Computation of global optimal cost:** Each node computes the boundary of the global optimal cost according to the pseudo-tree.
- **Termination:** After computation of global optimal cost, the boundary of the cost is converged to the optimal value in the root node. Then the optimal solution is decided according to the pseudo-tree in a top-down manner.

In this paper, important modifications for Adopt are applied to computation of the global optimal cost. Agent $i$ computes the cost using information as follows.

- $x_i$: variable of agent $i$. Value $d_i$ of $x_i$ is sent to lower neighbor nodes of $x_i$ using VALUE message.
- **current_context:** current partial solution of ancestor nodes of $x_i$. **current_context** is updated by VALUE message and context of COST messages.
- **threshold:** total amount of cost that is shared with subtree routed at $x_i$. **threshold** is received from parent node of $x_i$ using **threshold** message.
- **context**($x, d$), $lb_i(x, d)$, $ub_i(x, d)$: boundary of optimal cost for each value $d$ of variable $x_i$ and subtree routed at child node $x$. These elements are received from child node $x$ using COST message.

If **current_context** includes **context**($x, d$), upper and lower bounds of cost are $lb_i(x, d)$ and $ub_i(x, d)$ respectively. If **current_context** is incompatible with **context**($x, d$), **context**($x, d$), $lb_i(x, d)$, and $ub_i(x, d)$ are reset to $\{\}$, $0$, and $\infty$ respectively.

- $t_i(x, d)$: total amount of cost that is allocated to subtree routed at child node $x$ when $x_i$ takes value $d_i$. $t_i(x, d)$ is sent to $x$ using **threshold** message.

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

$$\delta_i(d) = \sum_{(x_i, d_i) \in \text{current_context}_i, j \in \text{upper neighbor nodes of } i} f_{i,j}(d_i, d_j)$$

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

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

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

**Upper bound** $UB_i$ and **lower bound** $LB_i$ for the subtree routed at $x_i$ are defined as follows.

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

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

Each agent $i$ exchanges messages, and updates local information. Eventually, at root node $r$, global optimal cost converges as $LB_r = \text{threshold}_r = UB_r$. The global optimal solution is decided according to the optimal cost. Details of the Adopt algorithm are shown in (Modi et al. 2005).
Serialization of resource constrained variables

In previous works, a version of the Adopt algorithm using a basic approach, which serializes resource constrained variables, is proposed. The pseudo-tree is generated considering resource constraints. Variables, which are related to an n-ary constraint, are placed in a single path of a pseudo-tree. For example, the pseudo-tree shown in Figure 2(a) is generated ignoring resource constraints. For example, the pseudo-tree shown in Figure 2(a) is generated from the RDCOP shown in Figure 1. In this example, $x_0$, $x_2$, and $x_3$, which are related to resource $r_0$, are placed on a single path of a pseudo-tree. $x_0$, $x_3$, and $x_4$, which are related to resource $r_1$, are also placed on a single path. If it is necessary to serialize variables, extra tree edges are inserted between nodes. In the example of Figure 2(a), tree edges $(x_2,x_3)$ and $(x_1,x_4)$ are inserted.

In the Adopt algorithm, Resource evaluation nodes, which evaluate resource constraints, are introduced. A resource evaluation node is added as a child node of the lowest node of serialized nodes. For example, in Figure 2(b), extra nodes $r_0$ and $r_1$ are added as child nodes of $x_3$ and $x_4$ respectively. Each agent sends its value of variable to resource evaluation nodes using the VALUE message. Then the resource evaluation node evaluates the total amount of resource requirement for its resource. If the resource constraint is satisfied, the resource evaluation node notifies its parent node using the COST message. The violation of the resource constraint is represented by infinity cost. In addition, it is possible to integrate the resource evaluation node into its parent node.

In this approach, no modification of the Adopt algorithm is necessary except adding resource evaluation nodes and handling infinity cost. However, large arity of resource constraint increases the depth of the tree, and reduces parallelism in search processing.

Solving RDCOP with Resource constraint free pseudo-tree

In this work, we propose a novel version of the Adopt algorithm for RDCOP. The proposed algorithm allows resource constraints related to nodes in different subtrees. The pseudo-tree is generated ignoring resource constraints. For example, the pseudo-tree shown in Figure 3 is generated from the RDCOP shown in Figure 1. In this example, there is a constraint edge of $r_0$ between two different subtrees, which contain $x_2$ and $x_3$ respectively. Similarly, there is a constraint edge of $r_1$ between $x_3$ and $x_4$.

In the original Adopt, constraint edges, which are placed among different subtrees, are not allowed. In such case, it is not possible to generate a COST message that notifies parent nodes of the violated solution correctly.

Introduction of virtual variables

The main idea of the proposed method is the introduction of virtual variables, which represent usage of resources. Each node shares resources with its parent node and child nodes using the virtual variables.

Virtual variable $vr_{a,i}$ is defined for resource $r_a$ and node $x_i$, which requires resource $r_a$ in the subtree rooted at $x_i$. $vr_{a,i}$ is owned by the parent node of $x_i$. $vr_{a,i}$ takes a value from its discrete domain $\{0, 1, \cdots, C(r_a)\}$.

As a simple example, a pseudo-tree, which is related to a single resource constraint, is shown in Figure 4. In this example, resource $r_0$ is related to variables $x_0$, $x_1$, $x_2$ and $x_3$. For these resources and variables, virtual variables $vr_{0,1}$, $vr_{0,2}$ and $vr_{0,3}$ are introduced. Each virtual variable $vr_{a,i}$ is owned by the parent node of $x_i$. The value of $vr_{a,i}$ is controlled by the parent node. Note that root node $x_0$ does not have a parent node. Therefore, it is assumed that the value of $vr_{0,0}$ is given from the virtual parent node. In this case, $vr_{0,0}$ takes a constant value that is equal to capacity $C(r_0)$ of resource $r_0$.

Value $dr_{a,j}$ of virtual variable $vr_{a,j}$, which is owned by agent $i$, is sent to $i$'s child node $j$ using the VALUE message. Therefore, the VALUE message is modified to contain $(x_i,d_i)$ and additional assignment $(vr_{a,j},dr_{a,j})$. When node $j$ receives the VALUE that contains $(vr_{a,j},dr_{a,j})$, node $j$ updates its current_context of node $j$ with new $(vr_{a,j},dr_{a,j})$.

In node $i$, assignments of virtual variables for resource $r_a$ should satisfy a constraint $c_{a,i}$ as follows.

$$c_{a,i} : dr_{a,i} \geq u_i(r_a,d_i) + \sum_{j \in \text{child nodes of } i \text{ which requires } r_a} dr_{a,j}$$ (6)

Here $dr_{a,i}$ denotes the value of $vr_{a,i}$, which is received from the parent node of $i$. The assignment $(vr_{a,i},dr_{a,i})$ is contained in current_context. If an assignment does not satisfy the resource constraint $c_{a,i}$, the violation of the resource constraint is represented by infinity cost.

Each node $i$ evaluates the boundary of optimal cost for current_context. Then the cost information is sent to the parent node of $i$ using the COST message. The context of the COST message is modified to contain additional assignment for virtual variables of $i$'s parent node.

The modification using virtual variables allows pseudo-trees, which are generated ignoring resource constraints. However, the additional virtual variables increase the search space.
Generating virtual variables

In a general case, variables are related to one or more resources. Moreover, variables are related to a subset of whole resources. Virtual variables are generated according to rules as follows.

1. Basically, if a subtree routed at node $i$’s child node $j$ requires resource $r_a$, then node $i$ owns virtual variable $vr_{a,j}$. However, the following cases are prioritized as special cases.

2. If node $i$ or multiple subtrees routed at $i$’s child nodes require $r_a$, then current context, contains assignment $(vr_{a,i}, dr_{a,i})$. In this case, $dr_{a,i}$ is decided as follows.
   (a) If no $i$’s ancestor node requires $r_a$, then $i$ is the root node for $r_a$. In this case, $dr_{a,i}$ is initialized as a constant that takes a value equal to capacity $C(r_a)$ of $r_a$.
   (b) If node $i$ is not the root node for $r_a$, then $i$’s parent node $h$ owns virtual variable $vr_{a,i}$. Therefore, VALUE messages, which are received from $h$, contain assignment $(vr_{a,i}, dr_{a,i})$.

3. If node $i$ requires resource $r_a$ and no subtree routed at $i$’s child node requires $r_a$, then $i$ is a leaf node for $r_a$. In this case, node $i$ has no virtual variables for $r_a$. Therefore, the resource constraint is defined by $dr_{a,i} ≥ u_i(r_a, d_i)$.

4. If multiple subtrees routed at $i$’s child nodes $j ∈ A'$ require $r_a$, then $i$ must share $r_a$ among child nodes $j ∈ A'$, even if node $i$ does not require $r_a$. Therefore, node $i$ owns virtual variables $(vr_{a,i,j} | j ∈ A')$.

An algorithm to generate virtual variables is shown in Algorithm 1. In this algorithm, it is assumed that a pseudo-tree has been generated. For the sake of simplicity, the algorithm consists of two phases of processing. In the first phase, each node $i$ computes a set $R_i^-\{j \mid \text{resources that are required by nodes in the subtree routed at node } i\}$. In the second phase, each node $i$ computes a set $R_i^+$ of resources that are shared from node $i$ or $i$’s ancestor nodes. According to these results, node $i$ generates set $X_i$ of own variables. This preprocessing is performed during or after construction of the pseudo-tree.

Growth of search space and efficient methods for search processing

Additional virtual variables increase the search space. Node $i$ selects an assignment for a set of variables $X_i = \{x_j\} \cup \{vr_{a,j} | j ∈ \text{Children}_a, r_a ∈ R_j\}$. Here $R_j$ denotes a subset of resources that are required in the subtree routed at node $j$. Cost evaluations in node $i$ are modified to $\delta_i(D_i), LB_i(D_i)$ and $UB_i(D_i)$ respectively. Here $D_i$ denotes a total set of assignments for $X_i$. Moreover, cost information of node $i$’s child node $j$ is evaluated for $X_i,j = \{x_i\} ∪ \{vr_{a,j} | r_a ∈ R_j\}$. Therefore, they are modified to $lb_i(j, D_{i,j}), ub_i(j, D_{i,j})$, $t_i(j, D_{i,j})$, and $context_i(j, D_{i,j})$ respectively.

As a result of these modifications, the size of the search space increases exponentially with the number of virtual variables. To reduce this drawback, additional efficient methods are necessary.

Pruning for partial solution
In node $i$, search processing for $X_i$ is necessary to calculate boundaries $LB_i$ and $UB_i$ for optimal cost. The search space increases exponentially with the number of virtual variables that are contained in $X_i$. However, it is possible to prune the search processing using a boundary defined by a resource constraint. If an assignment does not satisfy Equation 6, the cost of the assignment is $∞$. Therefore, the assignment is pruned.

A violation of a resource constraint does not depend on the evaluation of other resource constraints. If an assignment violates a resource constraint for $r_a$, the assignment is a violated assignment even if other resource constraints are satisfied.

Cost information of child nodes
Cost information of node $i$’s child node $j$ is modified to $lb_i(j, D_{i,j}), ub_i(j, D_{i,j})$, $t_i(j, D_{i,j})$ and $context_i(j, D_{i,j})$ respectively. The memory space for this information increases exponentially with the number of virtual variables that are contained in $X_{i,j}$. However, in the Adopt algorithm, default initial cost information is used when the cost information has not been received from the child nodes. Moreover, when $current_context$ is incompatible with $context_{i,j}(j, X_{i,j})$, the cost information is reset to the initial value. Therefore, it is unnecessary to store the cost information that takes the initial value.

Algorithm 1: Generate virtual variables

1. Initiation, $\{$
2. Generate pseudo-tree ignoring resource constraint.
3. if ($i$ is not root node) $p_i ←$ parent node of node $i$.
4. $C_i ←$ a set of child nodes of node $i$.
5. $R_i ←$ a set of resources required by node $i$.
6. $X_i ← \{x_i\}$.
7. if ($i$ is root node) $\{ \text{call Rootward}(), \text{call Leafward}(\phi). \}$
8. $\text{Rootward}()$ $\{$
9. $R_i ← R_i$.
10. for each $j$ in $C_i$ $\{$
11. $\text{call Rootward}()$ and receive $R_i^\sim$, $R_i^- ← R_i^- ∪ R_i^\sim$. $\}$
12. $\text{Leafward}(R_{p_i})$ $\{$
13. $R_i^- ← \phi$.
14. for each $r$ in $R_i^\sim$ $\{$
15. $n ←$ number of nodes $j$ s.t. $r ∈ R_j^\sim$.
16. if ($n ≥ 2$ or ($n = 1$ and $(r ∈ R_i$ or $r ∈ R_{p_i})$))$\{$
17. $R_i^+ ← R_i^+ ∪ \{r\}$. $\}$
18. for each $j$ in $C_i$ $\{$
19. for each $r$ in $R_i^\sim$ $\{$
20. if ($r$ is contained in $R_i^+ \} X_i ← X_i ∪ \{vr_{a,j}\}$. $\}$$
21. $\text{call Leafward}(R_i^+)$. $\}$
}
Upper limit of resource usage The proposed method allocates resources in a top down manner. This is similar to the maintenance of Threshold in the original Adopt. However this processing is speculative. To reduce overestimation in the allocation, an upper limit of resource usage is considered. As a part of preprocessing, each node computes its maximum usage for each resource, and notifies its descendants in a bottom up manner. As a result, each node obtain upper limits of resource usage for each resource and subtree. Each node limits resource allocation using the upper limits.

Correctness and complexity of the algorithm The proposed method uses additional virtual variables. This modification straightforwardly extends Adopt. In each node, the original variable and virtual variables can be considered as one integrated variable. The cost evaluation and invariants for the integrated variable are the same as the original definition of Adopt. Therefore, the optimality, soundness, and termination are the same as Adopt. Proposed method can detect unsatisfiability (i.e., it reports an infinity cost).

Additional virtual variables exponentially increase search space. In each node, the original variable and virtual variables can be considered as one integrated variable. Then the growth of search space can be considered as the growth of the domain of the integrated variable.

Evaluation The efficiency of the proposed method is evaluated by experiments. We used a modified graph coloring problem with three colors. Resource constraints are added to the original problem. The problems are generated using parameters \((n, d, r, k, c, l, u)\). The total number of nodes \(n\) and link density \(d\) are the basic parameters of the graph coloring problem. The link density \(d\) is set to 1 or 2. In original graph coloring problems, this setting of parameters is used to generate a low constrained problem. However, the problem contains additional resource constraints as follows.

Parameter \(r\) determines the number of resources. \(c = \lceil n \times k \rceil\) determines the capacity of a resource. \(l\) determines the arity of a resource constraint. In this problem setting, each variable is related to at least one resource constraint. For the sake of simplicity, the usage of a resource, which is required by an agent, is limited to 0 or 1. This means that each agent requires a unit amount of a resource or does not require one at all. Parameter \(u\) represents the ratio of a variable’s values that require a resource. In these experiments \(u\) is set to \(\frac{3}{4}\). Each problem instance is generated so that at least one assignment globally satisfies the resource constraint. The experiment is performed for 10 instances for each setting. We evaluated three versions of Adopt as follows: Local serialization of resource constrained variables (N), virtual variable (V) and virtual variable with upper limit of resource usage (VU). Each experiment is terminated at 9999 cycles. In that case, the cycle is considered as total number of message cycles.

Total number of message cycles is shown in Figure 5. In these results, the shapes of the graphs are not monotonic. The reason for the non-monotonicity is that the difficulty of the problem cannot be completely controlled.

In the case of \(r = 1\), message cycles of the competing method are greater than the proposed methods. In this case, the competing method generates a linear graph as a pseudo-tree. The linear pseudo-tree causes a delay in the processing of Adopt. On the other hand, the proposed method generates a pseudo-tree ignoring resource constraints. Therefore, the processing of Adopt is performed in parallel.

However, in the case of \(r = 4, k = 0.25\) and \(0.5\), the proposed method takes a larger number of cycles than the competing method. In this problem, the proposed method generates multiple virtual variables for each node of a pseudo-
Table 1: Size of pseudo-trees and dimension of assignments (n=20)

<table>
<thead>
<tr>
<th>d</th>
<th>r</th>
<th>l</th>
<th>avg.max. depth of pseudo tree</th>
<th>avg. branch. factor</th>
<th>avg.max. dim. of assign.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>N</td>
<td>V</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>20</td>
<td>20.0</td>
<td>5.3</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>10.8</td>
<td>5.3</td>
<td>1.2</td>
<td>3.5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>20</td>
<td>20.0</td>
<td>11.2</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>15.2</td>
<td>11.2</td>
<td>1.2</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Table 2: execution time (n=20)

<table>
<thead>
<tr>
<th>k</th>
<th>d</th>
<th>r</th>
<th>c</th>
<th>l</th>
<th>execution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>1.786</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
<td>0.021</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>2.010</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
<td>0.944</td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>0.507</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>5</td>
<td></td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>1.089</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>5</td>
<td></td>
<td>0.073</td>
</tr>
</tbody>
</table>

Conclusion

We proposed a distributed constraint optimization method for RCDCOP using a pseudo-tree that is generated ignoring resource constraints. The proposed method allows resource constraints related to different subtrees in the pseudo-tree. The main idea is to introduce a special set of virtual variables that represents the usage of resources. The addition of virtual variables increases the search space. To handle this problem, influence of placement of virtual variables/resources constraints in the pseudo tree is considered. Moreover, the search is pruned using the bounds defined by the resource constraints, if possible. The proposed method reduces the previous limitations in the construction of RCD-COP pseudo-trees. The efficiency of our technique depends on the class of problems being considered, and we described the obtained experimental results.

Virtual variables increase the search space of the internal processing of agents. In this paper, only basic boundary is used to prune the search. Additional variable ordering, forward checking and branch-and-bound methods (Fruder and Wallace 1992) are necessary for more efficiency. The proposed approach using virtual variables can be applied to another pseudo-tree based DPOP algorithms (Petcu and Faltings 2005; 2006).

Analysis of pseudo-trees to improve the efficiency of the proposed method and better representation of boundaries to prune the search processing, will be included in future work.

Acknowledgments

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B), 19300048, 2007.

References


