Resource Constrained Distributed Constraint Optimization with Virtual Variables

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

Marius Silaghi
Florida Institute of Technology
msilaghi@fit.edu

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

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, the 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) (Ali, Koenig, and Tambe 2005; Maheswaran et al. 2004; Mailler and Lesser 2004; Modi et al. 2005; Petcu and Faltings 2005) has been studied. With DCOPs, the agent states and the relationships between agents are formalized into a constraint optimization problem.

A dedicated framework called Resource Constrained DCOP (RCDCOP) has recently been proposed (Bowring, Tambe, and Yokoo 2006; Pecora, Modi, and Scerri 2006). 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. Multiply-constrained DCOP with privacy requirements is formalized in (Bowring, Tambe, and Yokoo 2006). Resource constrained distributed task scheduling modeled as n-ary constrained DCOPs, and the algorithm to solve such problems, are presented in (Pecora, Modi, and Scerri 2006). The previous research addressing RCDCOPs employs the Adopt algorithm (Modi et al. 2005), which is a basic solver for DCOPs. Adopt depends on a partial ordering of variables. The ordering is implied by a pseudo-tree for constraint networks. 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.

On the other hand, a basic resource constraint is a rather simple constraint that represents the limitation of the total usage of resources required by agents. Therefore, it is possible to allow resource constraints related to different subtrees in the pseudo-tree. 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

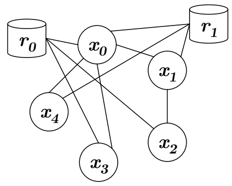


Figure 1: Resource constrained DCOP

depends on the class of problems being considered, and we describe the obtained experimental results.

Problem definition

Resource Constrained DCOP (RCDCOP)

A DCOP is defined by a set A of agents, a set X of variables 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. The cost of an assignment $\{(x_i, d_i), (x_j, d_j)\}$ is defined by a binary function $f_{i,j}(d_i, d_j) : D_i \times D_j \to \mathbb{N}$. The goal is to find a global optimal solution $\mathcal{A}$ that minimizes the global cost function: $\sum_{f_{i,j} \in F,\ \{(x_i,d_i),(x_j,d_j)\} \subseteq \mathcal{A}} f_{i,j}(d_i, d_j)$.

In RCDCOP resource constraints are added to DCOP. Resource constraints are defined by a set R of resources and a set U of resource requirements. A resource $r_a \in R$ has its capacity defined by $C(r_a) : R \to \mathbb{N}$. Each agent requires resources according to its assignment. For assignment (x_i, d_i) and resource r_a, a resource requirement is defined by $u_i(r_a, d_i) : R \times D_i \to \mathbb{N}$. For each resource, the total amount of requirements must not exceed its capacity. The global resource constraint is defined as follows: $\forall r \in R, \sum_{u_i \in U,\ \{(x_i,d_i)\} \subseteq \mathcal{A}} u_i(r, d_i) \le C(r)$. The resource constraint takes arbitral arity.

An example of RCDCOP that consists of 5 variables and 2 resources is shown Figure 1. In this example, x_0, x_2 and x_3 are constrained by resource R_0. x_0, x_1 and x_4 are constrained by resource R_1.

Background : Solving RCDCOP using Adopt

In previous work, the Adopt algorithm is employed to solve n-ary resource constrained DCOP. Adopt is a DCOP solver using a pseudo-tree for a constraint network. In this section, a brief description of pseudo-trees, Adopt and an extension of Adopt for n-ary constraints will be shown.

Pseudo-tree

The Adopt algorithm depends on a variable ordering defined by a pseudo-tree. The pseudo-tree is generated using a depth first search for the constraint network in the preprocessing of Adopt. The edges of the original constraint network are categorized into tree edges and back edges of the pseudo-tree. The tree edges represent the partial order relation between two variables. There is no edge between different subtrees. By employing this property, Adopt performs search processing in parallel.

Adopt

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.
- $current\_context_i$: current partial solution of ancestor nodes of x_i. $current\_context_i$ is updated by **VALUE** message and $context$ of **COST** messages.
- $threshold_i$: total amount of cost that is shared with subtree routed at x_i. $threshold_i$ is received from parent node of x_i using **THRESHOLD** message.
- $context_i(x,d)$, $lb_i(x,d)_i$, $ub_i(x,d)$: 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 $current\_context_i$ includes $context_i(x,d)$, upper and lower bounds of cost are $lb_i(x,d)$ and $ub_i(x,d)$ respectively. If $current\_context_i$ is incompatible with $context_i(x,d)$, $context_i(x,d)$, $lb_i(x,d)_i$ and $ub_i(x,d)$ are reset to $\{\}$, 0 and ∞ 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 $current\_context_i$ is defined as follows.

$$\delta_i(d) = \sum_{\substack{(x_j,d_j) \in current\_context_i, \\ j \in upper\ neighbor\ nodes\ of\ i}} f_{i,j}(d, d_j) \tag{1}$$

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

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

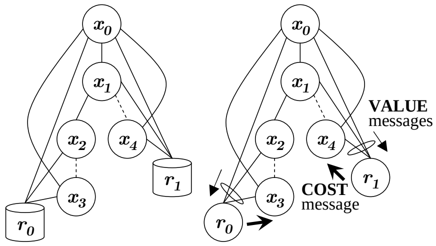
$$UB_i(d) = \delta_i(d) + \sum_{j \in child\ nodes\ of\ i} ub_i(x_j, d) \tag{3}$$

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

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

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

LB_i is initialized to 0, while UB_i can be initialized to $-\infty$.



(a) pseudo-tree (b) computation

Figure 2: Serializing of resource constrained variables

Each agent i exchanges messages, and updates local information. Eventually, at root node r, global optimal cost converges as $LB_r = threshold_r = UB_r$. 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 from the RCDCOP 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_1 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 not 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 RCDCOP with Resource constraint free pseudo-tree

In this work, we propose a novel version of the Adopt algorithm for RCDCOP. 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 RCDCOP 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_1 and x_4.

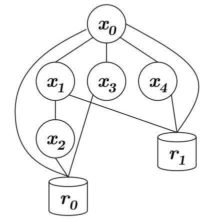


Figure 3: Resource constraint free pseudo-tree

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 routed 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.

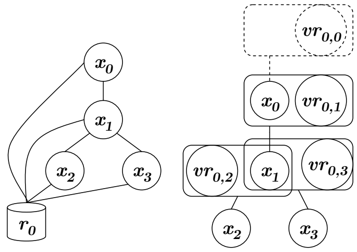
Value $dr_{a,j}$ of virtual variable $vr_{a,j}$, which is owned by agent i, is sent to i's child node j using the **VALUE** message. Therefore, the **VALUE** message is modified to contain (x_i, d_i) and additional assignments $(vr_{a,j}, dr_{a,j})$. When node j receives the **VALUE** that contains $(vr_{a,j},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

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

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 $current\_context_i$. 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 $current\_context_i$. Then the cost information is sent to the parent node of i using the **COST** message. The context of



(a) pseudo-tree (b) virtual variables

Figure 4: Virtual variables for resource constraint

the **COST** message is modified to contain additional assignments 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 $current\_context_i$ 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 $dr_{a,i} \geq u_i(r_a, d_i)$.
4. If multiple subtrees routed at i's child nodes $j \in A'$ require r_a, then i must share r_a among child nodes $j \in A'$, even if node i does not require r_a. Therefore, node i owns virtual variables $\{vr_{a,j} | j \in A'\}$.

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^- of 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 $\mathcal{X}_i$ of own variables. This preprocessing is performed during or after construction of the pseudo-tree.

Algorithm 1: Generate virtual variables

Initiation$_i${
  Generate pseudo−tree ignoring resource constraint.
  if(i is not root node) $p_i \leftarrow$ parent node of node i.
  $C_i \leftarrow$ a set of child nodes of node i.
  $R_i \leftarrow$ a set of resources required by node i.
  $\mathcal{X}_i \leftarrow \{x_i\}$.
  if (i is root node){ call Rootward$_i$(). call Leafward$_i(\phi)$. } }
Rootward$_i$(){
  $R_i^- \leftarrow R_i$.
  for each j in C_i{
    call Rootward$_j$() and receive R_j^-. $R_i^- \leftarrow R_i^- \cup R_j^-$. } }
Leafward$_i(R_{p_i}^+)${
  $R_i^+ \leftarrow \phi$.
  for each r in R_i^- {
    $n \leftarrow$ number of nodes j s.t. $r \in R_j^-$.
    if ($n \geq 2$ or ($n = 1$ and ($r \in R_i$ or $r \in R_{p_i}^+$))){
      $R_i^+ \leftarrow R_i^+ \cup \{r\}$. } }
  for each j in C_i{
    for each r in R_j^- {
      if(r is contained in R_i^+) $\mathcal{X}_i \leftarrow \mathcal{X}_i \cup \{vr_{r,j}\}$. }
    call Leafward$_j(R_i^+)$. } }

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 $\mathcal{X}_i = \{x_i\} \cup \{vr_{a,j} | j \in Children_i, r_a \in R_j\}$. 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 $\delta_i(\mathcal{D}_i)$, $LB_i(\mathcal{D}_i)$ and $UB_i(\mathcal{D}_i)$ respectively. Here $\mathcal{D}_i$ denotes a total set of assignments for $\mathcal{X}_i$. Moreover, cost information of node i's child node j is evaluated for $\mathcal{X}_{i,j} = \{x_i\} \cup \{vr_{a,j} | r_a \in R_j\}$. Therefore, they are modified to $lb_i(j, \mathcal{D}_{i,j})$, $ub_i(j, \mathcal{D}_{i,j})$, $t_i(j, \mathcal{D}_{i,j})$ and $context_i(j, \mathcal{D}_{i,j})$ 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 $\mathcal{X}_i$ 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 $\mathcal{X}_i$. 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 $lb_i(j, \mathcal{D}_{i,j})$, $ub_i(j, \mathcal{D}_{i,j})$, $t_i(j, \mathcal{D}_{i,j})$ and $context_i(j, \mathcal{D}_{i,j})$ respectively. The memory space for this information increases exponentially with the number of virtual variables that are contained in $\mathcal{X}_{i,j}$. How-

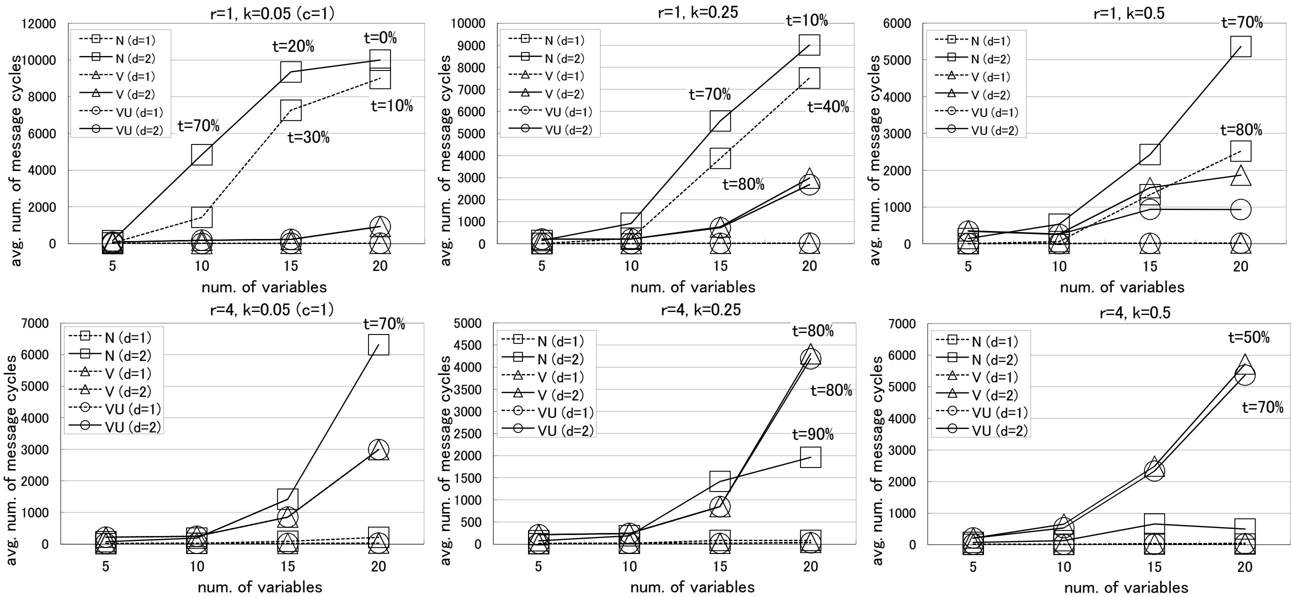


Figure 5: Message cycles (t: ratio of correctly terminated instances (others: 100%))

ever, in the Adopt algorithm, default initial cost information is used when the cost information has not been received from the child nodes. Moreover, when $current\_context_i$ is incompatible with $context_{i,j}(j, \mathcal{X}_{i,j})$, the cost information is reset to the initial value. Therefore, it is unnecessary to store the cost information that takes the initial value.

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 decendants 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 for 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 $\frac{2}{3}$. 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 9999 cycles 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

Table 1: Size of pseudo-trees and dimension of assignments (n=20)

d	r	l	avg.max. depth of pseudo tree		avg. branch. factor		avg.max. dim. of assign.
			N	V	N	V	
1	1	20	20.0	5.3	1.0	3.5	9.6
	4	5	10.8	5.3	1.2	3.5	13.0
2	1	20	20.0	11.2	1.0	1.5	3.7
	4	5	15.2	11.2	1.2	1.5	6.8

Table 2: execution time (n=20)

k	d	r	c	l	execution time (s)		
					N	V	VU
0.05	1	1	10	20	1.786	0.007	0.008
		4	1	5	0.021	0.242	0.253
	2	1	10	20	2.010	0.350	0.363
		4	1	5	0.944	3.885	4.167
0.5	1	1	10	20	0.507	32.524	0.940
		4	3	5	0.002	334.243	26.162
	2	1	10	20	1.089	5.656	1.491
		4	3	5	0.073	490.274	251.030

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-tree. Therefore, the search space of the proposed method is increased.

On the other hand, in the case of $r = 4, k = 0.05$, the proposed method takes smaller message cycles. In this case, resource constraints are rather tight. Therefore, local serialize version of Adopt generates large number of infinity cost messages. This also increases message cycles.

Results related to generated pseudo-trees and the dimension of assignments are shown in Table 1. In the competing method N, the depth of the pseudo-tree increases when the number of resources is small.

In the proposed method, the dimension of the assignment for each node increases with the number of resources. The dimension also depends on the branching factor. The total number of cost information that is recorded in each node increases with the dimension of assignment.

The total execution time is shown in Table 2. The experiment is performed on a machine with a 1.6GHz Itanium2 processor and 32GB memory. The execution time depends on the total number of message cycles and computation cost. This result includes instances which were teminated at 9999 cycle. The cost increases in the following order: N, V, VU. In the case of $r = 1, k = 0.5$, the efficient method of VU reduces execution time.

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 RCDCOP 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 a basic boundary is used to prune the search. Additional variable ordering, forward checking and branch-and-bound methods (Freuder and Wallace 1992) are necessary for more efficiency. The proposed approach using virtual variables can be applied to other pseudo-tree based DPOP algorithms (Petcu and Faltings 2005; 2006).

Analysys 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

Ali, S. M.; Koenig, S.; and Tambe, M. 2005. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *4th International Joint Conference on Autonomous Agents and Multiagent Systems*, 1041–1048.

Bowring, E.; Tambe, M.; and Yokoo, M. 2006. Multiply constrained distributed constraint optimization. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems*, 1413–1420.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58(1):21–70.

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 *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, 310–317.

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

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.

Pecora, F.; Modi, P.; and Scerri, P. 2006. Reasoning About and Dynamically Posting n-ary Constraints in ADOPT. In *7th International Workshop on Distributed Constraint Reasoning, at AAMAS, 2006*.

Petcu, A., and Faltings, B. 2005. A Scalable Method for Multiagent Constraint Optimization. In *9th International Joint Conferece on Artificial Intelligence*, 266–271.

Petcu, A., and Faltings, B. 2006. O-DPOP: An algorithm for Open/Distributed Constraint Optimization. In *National Conference on Artificial Intelligence*, 703–708.