Distributed Cooperative Optimization on Cluster Trees

Toshihiro Matsui and Hiroshi Matsuo
Nagoya Institute of Technology
Gokiso-Cho, Showa-ku, Nagoya, Aichi, 466-8555, Japan
{*matsui.t, matsuo*}*@nitech.ac.jp*

***Abstract*—Resource allocation problems on resource supply networks are formalized with Distributed Constraint Optimization Problems. In previous studies, solution methods based on pseudo trees have been proposed. However, when the pseudo trees contain nodes of high degree and a large number of cycles, those methods are not applicable due to the large size of local problems in agents. Here, we employ cluster trees that hierarchically divide the problem into sub-problems. With optimistic approximation, solution methods on the cluster tree are applied to several large problems.**

***Keywords*-Distributed Constraint Optimization; resource allocation; resource supply network**

I. Introduction

Distributed resource allocation on networks including power supply networks is an important application of multiagent systems. Since the resource allocation is an optimization problem, distributed optimization methods are necessary to solve the problem. Distributed Constraint Optimization Problem (DCOP) is a basic framework of cooperative problem solving in multiagent systems [1], [2], [3], [4]. The states of agents and the relationships between agents are formalized into a constraint optimization problem, which is distributed on the multiagent systems. Several types of distributed search algorithms are employed for the DCOPs. The representation of DCOPs and corresponding solution methods can also be extended to meet particular problems.

Resource allocation problems motivated by the power supply networks of smart-grid systems have been studied as distributed optimization problems. In the supply networks, resources that are initially distributed among source nodes have to be shared among all nodes. In a related work [5], a dedicated class of DCOPs and a solver have been proposed to generate feeder trees under resource constraints. Other studies address the resource allocation on a feeder tree [6], [7]. These studies relate to Resource Constrained DCOP (RCDCOPs) [8], which is a dedicated class of problems where shared resources are represented as global constraints that can be decomposed into agents. Their solution methods are based on pseudo trees on the problems.

Since computational and memory costs of the above solution methods grow with several parameters related to the size of the problems, those methods are not directly applicable to large problems. In the case of this class of problems, approximation of the pseudo trees is not straightforward because of hard constraints on the resource allocation. On the other hand, if inexact methods are allowed, it is possible to employ other graph structures for solution methods. In this work, we investigate another type of method based on cluster trees on the resource supply networks.

II. Background

A. Problem Definition

As addressed in the previous section, there are several studies of distributed resource allocation problems on a network that are motivated by power supply networks containing distributed resources. We use a definition of a resource allocation problem similar to [6], [7]. Most of the definitions are inherited from [7].

In this network, amounts of the resource in several nodes are allocated to other nodes. The previous studies assume that the structure of the networks is limited to trees [6], [7]. Here, we address networks including cycles. While feeder-trees are common in actual power networks, we focus on more general cases.

The components of the network are as follows.

- Nodes: Each node in the network supplies or consumes an amount of resource. When the nodes supply resources, they are called sources. On the other hand, nodes that receive resources from other nodes are called sinks. There are limitations on the amounts of supply and consumption. The node also has a preference on the amount of the resource.
- Links: Each link connects two nodes. The links and nodes form paths to transfer an amount of resource. The capacity of the link limits the amount of resource that is transferred in a link. As a common assumption, the loss of the resource in the transfer is ignored.

In addition to the above limitations on the amount of resource, there is another type of constraint on the transferred resources. Namely, the total amount of the resource that is supplied and consumed has to be zero. The goal of the problem is to globally optimize an aggregation of preference under the constraints.

The problem is formally defined by $\langle N, L, R, F, \mathcal{L} \rangle$, where N, L, R, F, and $\mathcal{L}$ are a set of nodes, a set of links, a set of amounts of resource on nodes, a set of cost functions, and a set of amounts of resource on links, respectively.

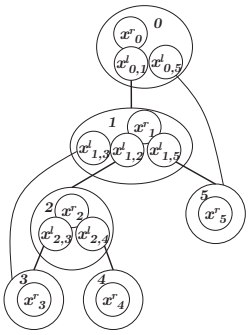


Figure 1. Pseudo tree for RCDCOP

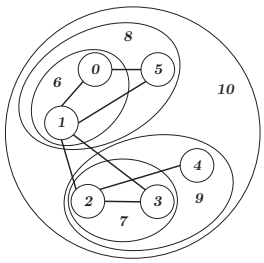


Figure 2. Hierarchical Decomposition

For node $i \in N$, the preference of the supply and the consumption on the amounts of the resource is defined as follows.

- R_i: $R_i \in R$ is a finite set of amounts of resource that are supplied or consumed by node i. Where amount $r \in R_i$ is a negative value, r represents an amount of the supplied resource. On the other hand, where amount r is a positive value, it represents an amount of consumed resource. Node i chooses a value of the amount from R_i.
- $f_i(r)$: $f_i(r) \in F$ is a cost function from amount $r \in R_i$ of the resource to a non-negative value. Here, we use cost functions to represent the preferences of the nodes because our solution methods are designed for minimizing problems.

Link (i, j) is defined for a pair $\langle i, j \rangle$ of nodes. For each node i, the set of neighborhood nodes that are directly connected with links is denoted as Nbr_i. The transfer of the resource on link $(i, j) \in L$ is represented as follows.

- $\mathcal{L}_{i,j}$: $\mathcal{L}_{i,j} \in \mathcal{L}$ is a finite set of amounts of resource transferred through link (i, j). Amount $l \in \mathcal{L}_{i,j}$ is a finite value such that $-l^c_{i,j} \leq l \leq l^c_{i,j}$, where $l^c_{i,j}$ is the capacity of link (i, j). The sign of value l represents the direction of the transfer. It is necessary to define a direction of a flow on the network. l takes a positive value when the corresponding link transfers an amount of resource in the same direction as the flow.

In each node $i \in N$, the summation of r_i and $l_{i,j}$ for all links (i, j) that connect node i must always be zero. The constraint is defined as $\sum_{(i,j)\in L^{in}_i} l_{i,j} = r_i + \sum_{(i,k)\in L^{out}_i} l_{i,k}$ with set L^{in}_i of input links and set L^{out}_i of output links of node i. L^{in}_i and L^{out}_i represent a flow on the network.

For allocation $\mathcal{R}$ of amounts of the resource for all nodes, the global cost $f(\mathcal{R})$ is defined as $f(\mathcal{R}) = \sum_{i\in N} f_i(r_i)$. Here, r_i takes a corresponding value in allocation $\mathcal{R}$. The goal of the problem is to find the optimal allocation $\mathcal{R}^*$ that minimizes $f(\mathcal{R})$ under the constraints.

B. Representation as Resource Constrained DCOP

DCOPs have been studied as a framework of multiagent cooperation. With the representation of DCOPs, an optimization problem in a multiagent system is defined as a constraint optimization problem whose variables, constraints, and evaluation functions are distributed among agents. The problem is solved using distributed cooperative search algorithms that are based on message communication.

RCDCOP [8] is an extended class of DCOPs that contains dedicated representations of resources and constraints related to the resources. The resource allocation problem shown in Subsection II-A is formalized as the RCDCOPs [6], [7]. Here, we show an RCDCOP that represents the resource allocation problem.

The RCDCOP for the resource allocation on the network is defined by $\langle A, X^r, X^l, D^r, D^l, F, C \rangle$. Here, A represents a set of agents. X^r is a set of variables that represent amounts of supplied or consumed resource in the nodes. X^l is a set of variables that represent amounts of transferred resources in the links. D^r and D^l are sets of finite domains of variables in X^r and X^l, respectively. F is a set of cost functions and C is a set of resource constraints. Each agent $i \in A$ corresponds to a node in the resource allocation problem. For the sake of simplicity, we use the notation of an agent and its corresponding node interchangeably.

Additionally, a partial order on a set of agents is defined based on the pseudo tree [3] rooted at a node. Figure 1 shows a pseudo tree for a RCDCOP. In the figure, six agents/nodes are represented as nodes of the pseudo tree. The pseudo tree corresponds to a spanning tree of the original network. Edges of the spanning tree are called tree-edges. Other edges are called back-edges. Based on the pseudo tree, notations of parent agent p_i, set of child agents Ch_i and set of lower neighborhood agents $Nbrs^l_i$ (the child and pseudo child nodes) are defined for each agent i.

Agent i has a variable $x^r_i \in X^r$ that represents the amount of supplied or consumed resource. i also has a set $X^l_i \subset X^l$ of variables that represent the amount of transferred resource from i. $x^l_{i,j} \in X^l_i$ represents the amount of resource transferred from agent i to its lower neighborhood agents $j \in Nbrs^l_i$. Similarly, $x^l_{p_i,i} \in X^l_{p_i}$ represents the amount of resources transferred from i's parent agent p_i to i. Agent i decides the values of the variables except for $x^l_{p_i,i}$, whose value is determined by p_i.

$D^r_i \in D^r$ defines the domain of variable x^r_i for agent i. $D^l_{i,j} \in D^l$ defines the domain of variable $x^l_{i,j}$ for link (i, j). D^r_i and $D^l_{i,j}$ correspond to R_i and $\mathcal{L}_{i,j}$, respectively.

$f_i(x^r_i) \in F$ is a cost function that corresponds to $f_i(r_i)$ for node i in the resource allocation problem. Similarly, $c_i \in C$ is a resource constraint for node i. The resource constraint c_i is defined as $c_i : x^l_{p_i,i} = x^r_i + \sum_{x^l_{i,j}\in X^l_i} x^l_{i,j}$.

The global cost function $f(\mathcal{X})$ for assignment $\mathcal{X}$ for all variables in $X^r \cup X^l$ is defined as $f(\mathcal{X}) = \sum_{i\in N} f_i(x^r_i)$ Here, x^r_i takes a corresponding value in $\mathcal{X}$. The optimal allocation $\mathcal{X}^*$ minimizes $f(\mathcal{X})$ under the constraints.

C. Computation based on Pseudo Trees

Several exact solution methods can be applied to the RCDCOP defined in Subsection II-B. In [6], [7], solution methods are shown for the case where the networks are trees. Those methods are generalized for pseudo trees. Here, we show the most important computation on the pseudo tree. For more details, please see [2], [6], [7], [8].

The computation of the cost value is recursively defined. The optimal cost $g_i^*(\mathcal{X}_i^u)$ for assignment $\mathcal{X}_i^u$ of resource from i's ancestor nodes and the sub-tree rooted at agent i is represented as follows:

$$g_i^*(\mathcal{X}_i^u) = \min_{\mathcal{X}_i} g_i(\mathcal{X}_i^u \cup \mathcal{X}_i) \tag{1}$$

$$g_i(\mathcal{X}_i^u \cup \mathcal{X}_i) = \delta_i(\mathcal{X}_i^u \cup \mathcal{X}_i) + \sum_{j \in Ch_i, \mathcal{X}_j^u \subseteq \mathcal{X}_i^u \cup \mathcal{X}_i} g_j^*(\mathcal{X}_j^u) \tag{2}$$

$$\delta_i(\mathcal{X}_i^u \cup \mathcal{X}_i) = \begin{cases} f_i(d_i) \text{ s.t. } (x_i^r, d_i) \in \mathcal{X}_i & c_i \text{ is satisfied.} \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

Here, $\mathcal{X}_i$ denotes an assignment such that $\{(x_i^r, d_i)\} \cup \bigcup_{j \in Nbr_i^l} \{(x_{i,j}^l, d_{i,j})\}, d_i \in D_i^r, d_{i,j} \in D_{i,j}^l$.

In Equations (1), (2), and (3), for the sake of simplicity, it assumed that each agent is able to refer to cost values and assignments of other agents. However, in actual computation, the values are unknown until they are received from other agents as messages. To represent the unknown cost values, a lower limit value 0 and an upper limit value ∞ are employed. As a result of the aggregation with limit values, a cost value is separated into a lower bound and an upper bound of the true value.

Here, we prefer an algorithm similar to ADOPTs [2], [8], [7] that is based on tree-search and partial dynamic-programming since its memory use is relatively lower than that of pure dynamic programming [3], [6]. In the processing, a single solution is constructed at a time based on the pseudo tree. The processing consists of two phases: the computation of the cost values and the decision of the optimal assignments of the variables. In the computation of the cost values, the globally optimal cost value is computed. The tree-search repeats the computation for each solution. In the search processing, an assignment (*context*) $\mathcal{X}_i^u$ is sent in a top-down manner using VALUE messages that are sent along tree edges. For the current context, cost values are aggregated in a bottom-up manner using COST messages. Unknown assignments are evaluated using upper and lower bounds of cost values. In the root agent, the boundaries eventually converge to the optimal value. Then, the optimal assignments of the root agent's variables are determined. Similarly, optimal assignments of other variables are recursively determined in a top-down manner.

In the case where the pseudo trees contain nodes of high degree and a large number of cycles, those methods are not applicable due to the large size of local problems in agents. The size of the local problem exponentially grows with the degree of the node and the induced width [3] of the pseudo tree. An approximation method eliminate cycles of pseudo trees [9]. In the case of the resource allocation problem on resource supply networks, the elimination means that the amount of resource is fixed to a constant value for the corresponding link because the resource constraint has to be satisfied for the link. It seems not to be easy to choose such constant values. Moreover, the size of the local problem also grows with the number of lower neighborhood nodes in Nbr_i^l. To reduce the size of combinations, the size $|\mathcal{L}_{i,j}|$ of link (i, j) and the domain of the corresponding variable can be reduced. This also reduces feasibility of the original problem.

III. Distributed resource allocation on cluster tree

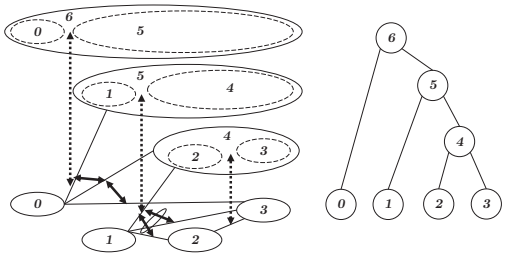
A. Hierarchical Decomposition of Problems

As shown above, exact solution methods based on pseudo trees are not directly applicable to large and complex problems. Approximation methods are therefore necessary.

To approximate the resource allocation problems, we employ a different method that hierarchically decomposes the problems. Namely, the network is recursively divided into multiple parts. Links between the parts are then approximated as a single link. This kind of method may decrease feasibility of the original problem since resources are approximately allocated using the hierarchical structure. On the other hand, if there are sufficient paths in the divided parts, the method takes advantage of decreasing the size of the problems. Since each region is represented as an abstract node, we assume that an overlay communication link between the abstract nodes is available.

As a simple method, a region is decomposed into two regions in the hierarchical structure. The regions are hence represented as a binary tree. The tree is basically the same as the well-known cluster tree. Based on the cluster tree, notations of parent node $cPrnt_i$ and set of child nodes $CChild_i$ are defined for node i. Since a cluster consists of two nodes, another node in the same cluster is referred to as peer node $cPeer_i$. The set of neighborhood nodes is denoted as $CNbrs_i$. Similar to the representation of RCDCOP, each node i corresponds to an agent. Figure 2 shows a hierarchical decomposition of the network. Below, we mainly use notations of nodes and links shown in Subsection II-A instead of RDCDOP because we also need a horizontal view of the network.

In the actual computation, the hierarchical structure is constructed in a bottom up manner that repeats clustering of two neighboring nodes. In the clustering, clusters with fewer nodes have priority. When a new node k is generated by integrating two nodes i and j, link (i, j) is eliminated in k's view. For other links that connect to i and j, corresponding links that connect to k are generated. If there are two links



(Unbalanced tree for explanation)

Figure 3. Hierarchical Decomposition and Cluster Tree

(i, o) and (j, o) from the two nodes i and j to the same node o, those links are integrated into new link (k, o). In that case, capacity $l^c_{k,o}$ of new link (k, o) is defined as $l^c_{k,o} = l^c_{i,o} + l^c_{j,o}$. If there is link (i, p) from only one node i of k's child nodes to non-child node p, new link (k, p) directly corresponds to the original link. The capacity of (k, p) equals $l^c_{i,p}$. In Figure 3, link $(2, 3)$ is eliminated in node 4. Links $(1, 2)$ and $(1, 3)$ are integrated into $(1, 4)$. Link $(0, 3)$ corresponds to $(0, 4)$.

In addition, cost functions of child nodes i and j are aggregated into a new cost function of k. Each value of the new cost function is the summation of corresponding values of original cost functions. If there are multiple summation values for the same amount of resource, the minimum value is chosen. Moreover, there are resource constraints among links connecting to nodes i and j. In the case of violating the constraint, the cost value is ∞. Let $l^c_{i,j}$ denote the capacity of link (i, j). For all links connecting i except (i, j), let l^c_i denote the summation of capacity for those links. Similarly, l^c_j denotes the summation of capacity for all links connecting j except (i, j). New cost function $f_k(r_k)$ is defined as follows.

$$f_k(r_k) = \min f(r_i, r_j) \text{ where } r_k = r_i + r_j \tag{4}$$

$$f(r_i, r_j) = \begin{cases} f_i(r_i) + f_j(r_j) & c_{i,j} \text{ is satisfied} \\ \infty & \text{otherwise} \end{cases} \tag{5}$$

$$\begin{aligned} c_{i,j}: \quad & r_i + r_j = r_k \wedge (\exists l, (-l^c_{i,j} \le l \le l^c_{i,j}) \wedge \\ & (-l^c_i \le r_i - l \le l^c_i) \wedge (-l^c_j \le r_j + l \le l^c_j)) \end{aligned} \tag{6}$$

The above integration is repeated until there are no neighboring nodes that are not integrated.

The pseudo code of the distributed clustering algorithm is shown in Figure 4. flg^{trm}_i is a flag that represents the termination of the processing. $cSize_i$ represents the size of i's cluster. Variables whose name begin with '$cLock$' are used to maintain the locking status as shown below. Because the integration affects neighboring nodes, the information of nodes is propagated to nodes in two hops (lines 32-38, 40-43) using CLSTST messages. [1] Here, we use a simple lock-and-commit protocol for mutual exclusion of integrations. Due to limitation of space, procedure MaintainLocking (line 18) is not shown. In the procedure, agent i that has no parent

[1]It is possible to eliminate extra information for non-peer nodes.

```
1  Main procedure:
2  flg^trm_i ← false. CNbrs_i ← Nbrs_i.
3  CLinks_i ← information of links connecting to i.
4  cSize_i ← 1. cPeer_i ← ϵ. cPrnt_i ← ϵ. CChld_i ← ∅.
5  cLockReq_i ← false. cLockAck_i ← ϵ. cLockAckCSize_i ← ϵ.
6  cLockRdy_i ← false. cLockWait_i ← false.
7  for each j ∈ Nbrs_i{
8   CNbrs_j ← ∅. CLinks_j ← ∅. cSize_j ← ϵ. cPeer_j ← ϵ.
9   cPrnt_j ← ϵ. cLockReq_j ← false. cLockAck_j ← ϵ.
10  cLockAckCSize_j ← ϵ. f_j ← ϵ. }
11 Similarly, initialize the information of nodes in just two hops.
12 Maintain. // initiation.
13 repeat{ receive messages until a break condition. Maintain.}

15 Maintain:
16 if(¬flg^trm_i){
17  if(information of all nodes in two hops has been received){
18   if(¬cLockRdy_i){ MaintainLocking. } // cLockRdy_i is set.
19   if(cLockRdy_i ∧ ¬cLockWait_i){
20    Clustering. cLockWait_i ← true. }
21   if(cLockRdy_i ∧ cLockWait_i){
22    if(cPrnt_{cPeer_i} = cPrnt_i∧
23     (cPrnt_i received CLSTST messages from all nodes in
24      CNbr_{cPrnt_i})){ // i can refer cPrnt_i's status.
25      cLockReq_i ← false. cLockRdy_i ← false.
26      cLockWait_i ← false. } }
27   if(cPrnt_i = ϵ ∧ cLockAck_i ≠ ϵ){
28    // reflect information of peer node of cluster
29    if(cPeer_{cLockAck_i} = i){
30     cPeer_i ← cLockAck_i. cPrnt_i ← cPrnt_{cPeer_i}. } }
31  }
32  send (CLSTST, CNbrs_i, CLinks_i, cSize_i, cPeer_i,
33   cPrnt_i, cLockReq_i, cLockAck_i, cLockAckCSize_i, f_i,
34   {(CNbrs_j, CLinks_j, cSize_j, cPeer_j, cPrnt_j, cLockReq_j,
35   cLockAck_j, cLockAckCSize_j)|j ∈ Nbrs_i})
36   to j ∈ CNbrs_i.
37  if(¬cLockReq_i ∧ cLockAck_i = ϵ ∧ cPrnt_i ≠ ϵ){
38   flg^trm_i ← true.} }

40 Receive (CLSTST, CNbrs, CLinks, cSize, cPeer, cPrnt,
41  cLockReq, cLockAc, cLockAckCSize, f,
42  set of information of neighborhood nodes of node j)
43  from j ∈ CNbrs_i: update information of each node.
```

Figure 4. Distributed Clustering (Procedures of Node i) (1)

tries to get a lock by assigning $true$ to $cLockReq_i$ while ties are broken based on the size of clusters and identifiers of nodes. On the other hand, agent i that has no parent acknowledges the locking of higher priority node j by assigning j and $cSize_j$ to $cLockAck_i$ and $cLockAckCSize_i$, respectively. If agent i receives acknowledgements from nodes in two hops, i clusters i and one of the neighborhood nodes (lines 19-20). After the update is propagated to related nodes, the locking is released (lines 21-26).

Figure 5 shows the procedure of clustering. Node i generates and initializes its parent node k (lines 1-17). Then, new links are generated based on links that relate i and $cPeer_i$ (lines 18-30). New cost function $f_k(r)$ is generated (lines 31-39). Finally, node k starts its processing (line 40).

Clustering:
choose j s.t. argmin$_{j \in Nbr}$ $cSize_j$ where
  $Nbr = \{m | m \in Nbr_i \wedge cPrnt_m = \epsilon\}$. $cPeer_i \leftarrow j$.
generate new node k. $cPrnt_i \leftarrow k$. $k\_flg_k^{term} \leftarrow$ false.
$k\_cSize_k \leftarrow cSize_i + cSize_j$. $k\_cPrnt_k \leftarrow \epsilon$.
$k\_cPeer_k \leftarrow \epsilon$. $k\_cLockReq_k \leftarrow$ false. $k\_cLockAck_k \leftarrow \epsilon$.
$k\_cLockAckCSize_k \leftarrow \epsilon$. $k\_cLockRdy_k \leftarrow$ false.
$k\_cLockWait_k \leftarrow$ false. $k\_CChld_k \leftarrow \{i, j\}$.
store information of i and j to node k in the same manner of
  CLSTST messages.
let $CNbr = \{n | n \in (CNbr_i \backslash \{j\}) \cup (CNbr_j \backslash \{i\}) \wedge$
  $cPrnt_n = \epsilon\}$.
$k\_CNbr_k \leftarrow CNbr$.
store information of $j \in CNbr$ to node k
  in the same manner of CLSTST messages
  but use default (empty) information for nodes in $CNbr_j$.
if($|k\_CNbr_k| = 0$){ $k\_cPrnt_k \leftarrow$ noparent. }
// integration of links
for each pair of links (i, p) and (j, p) s.t. $cPrnt_p = \epsilon$ {
  generate new link (k, p). $l^c_{k,p} \leftarrow l^c_{i,p} + l^c_{j,p}$.
  record that (k, p) originates from (i, p) and (j, p).
  store information of (k, p) into k_CLinks_k. }
for each link (i, p) s.t. there are no (j, p), $cPrnt_p = \epsilon$ {
  generate new link (k, p). $l^c_{k,p} \leftarrow l^c_{i,p}$.
  record that (k, p) originates from (i, p).
  store information of (k, p) into k_CLinks_k. }
for each link (j, p) s.t. there are no (i, p), $cPrnt_p = \epsilon$ {
  generate new link (k, p). $l^c_{k,p} \leftarrow l^c_{j,p}$.
  record that (k, p) originates from (j, p).
  store information of (k, p) into k_CLinks_k. }
// integration of cost functions
$l^c_i \leftarrow \sum_{l^c_{i,p}}$ in $_{CLinks_i}$ except $_{l^c_{i,j}}$ $l^c_{i,p}$.
$l^c_j \leftarrow \sum_{l^c_{j,q}}$ in $_{CLinks_j}$ except $_{l^c_{i,j}}$ $l^c_{j,q}$.
for all r s.t. $r = r_i + r_j, r_i \in R_i, r_j \in R_i$ { $w^*_r \leftarrow \infty$. }
for each $r_i \in R_i$ { **for** each $r_j \in R_i$ {
  if($\exists l \in \mathcal{L}_{i,j}, -l^c_i \le r_i - l \le l^c_i \wedge -l^c_j \le r_j + l \le l^c_j$){
    $r \leftarrow r_i + r_j$. $w \leftarrow f_i(r_i) + f_j(r_j)$.
    if($w < w^*_r$){ $w^*_r \leftarrow w$. } } } }
generate $k\_f_k(r)$ from w^*_k.
launch node k.

$k\_$ denotes k's variable.

Figure 5. Distributed Clustering (Procedures of Node i) (2)

B. Resource Allocation based on Hierarchical Structure

Based on the hierarchical structure, the resource allocation is computed in a top-down manner. In this phase, node k allocates amount $\tilde{r}_i$ and $\tilde{r}_j$ of resource for child nodes i and j that have been integrated into k.

When node k is the root node, $\tilde{r}_k$ is zero. In the allocation, constraint $\tilde{r}_k$=$\tilde{r}_i + \tilde{r}_j$ has to be satisfied. With $\tilde{r}_i$ and $\tilde{r}_j$, amount $\tilde{l}_{i \to j}$ of resource transferred between node i and link (i, j) is determined. Similarly, amount $\tilde{l}_{j \to i}$ is determined for node j. Here, $\tilde{l}_{i \to j} = -\tilde{l}_{j \to i}$.

When node k is not the root node, there are links between k and its neighborhood nodes. Such links are categorized into two types:

1) A link between k and peer node $cPeer_k$ that has been integrated with k. The allocation of resource for this link is determined by parent node $cPrnt_k$.

2) Links between node k and other nodes except peer node $cPeer_k$. The allocation of resource for these links is originally determined in one of the higher layers.

The amount of resource is possibly decomposed in the top-down processing. Node k maintains the correspondence among each link (k, p) and links that connect to its child nodes i and j. The amount of resource transferred via link (k, p) has been allocated in higher layers. Node k therefore determines the amount of resource transferred via the corresponding links for i and j.

When link (k, p) corresponds to two links (i, p) and (j, p), the shares of resource have to be determined. The optimal allocation of these shares requires search processing. In the case that capacities $l^c_{i,p}$ and $l^c_{j,p}$ of links (i, p) and (j, p) are large, the search processing is expensive since there are $(2l^c_{i,p} + 1)(2l^c_{j,p} + 1)$ allocations. To avoid this computation, we optimistically determine the allocation. Namely, the allocation is determined in proportion to the ratio between $l^c_{i,p}$ and $l^c_{j,p}$. When amount $\tilde{l}_{k,p}$ of resource is allocated to link (k, p), allocations $\tilde{l}_{i,p}$ and $\tilde{l}_{j,p}$ for links (i, p) and (j, p) are represented as $\tilde{l}_{k,p} = \tilde{l}_{i,p} + \tilde{l}_{j,p} \wedge \tilde{l}_{i,p}/l^c_{i,p} \approx \tilde{l}_{j,p}/l^c_{j,p}$.

As shown above, for all links, node k decomposes the allocation of resources into child nodes i and j. Then, node k determines allocations $\tilde{r}_i$, $\tilde{r}_j$, $\tilde{l}_{i \to j}$, and $\tilde{l}_{j \to i}$ such that the value of cost function $f_k(r_k)$ is minimized. In the minimization, constraints of (a) $\tilde{r}_k$=$\tilde{r}_i$+$\tilde{r}_j$, (b) $\tilde{l}_{i \to j} = -\tilde{l}_{j \to i}$ and (c) one similar to Equation (6) have to be satisfied. In constraint (c), the shares of resource that are determined in higher layers are used instead of boundaries of l^c_i and l^c_j in Equation (6). The allocation is totally represented as follows.

$$(\tilde{r}_i, \tilde{r}_j) = \text{argmin} f'(r_i, r_j) \tag{7}$$

$$\tilde{l}_{i \to j} = -(\tilde{l}_i + r_i), \tilde{l}_{j \to i} = -(\tilde{l}_j + r_j) \tag{8}$$

$$f'(r_i, r_j) = \begin{cases} f_i(r_i) + f_j(r_j) & c'_{i,j} \text{ is satisfied} \\ \infty & \text{otherwise} \end{cases} \tag{9}$$

$$\begin{aligned} c'_{i,j} \quad & : r_i + r_j = \tilde{r}_k \wedge \\ & r_i + l_{i \to j} = -\tilde{l}_i \wedge r_j + l_{j \to i} = -\tilde{l}_j \wedge \\ & l_{i \to j} = -l_{j \to i} \wedge -l^c_{i,j} \le l_{i \to j} \wedge l_{i \to j} \le l^c_{i,j} \end{aligned} \tag{10}$$

Here, $\tilde{l}_i$ and $\tilde{l}_i$ denote the total amounts of resource allocated to links connecting nodes i and j except (i, j), respectively. The allocations of resource are notified to nodes i and j. Then, similar processing is repeated in lower layers. Since the method obviously decreases the size of the solution space, feasible solutions are possibly lost.

C. Search on Cluster Trees

Since the method shown in Subsection III-B greedily allocates the amount of resource, it easily falls into infeasible solutions. To avoid such infeasible solutions, we employ distributed tree search on the cluster trees. When a node cannot satisfy the resource constraints, it notifies its parent node of the infeasibility. As a result of this backtracking,

```
1  Main procedure:
2  let t and s denote cPrnt_k and cPeer_k, respectively.
3  let i and j denote child nodes in CChld_k.
4  flg_t^trm ← false. flg_k^trm ← false.
5  flg_k^ok ← indeterm. flg_k^okall ← indeterm.
6  ∀b ∈ {i, j}, ∀r ∈ R_k, ∀l ∈ L_{i,j}, flg_{b,r,l}^ok ← indeterm.
7  ctx_t^node ← ∅. ctx_t^link ← ∅. r̃_k ← ϵ. alc_k^links ← ∅.
8  if(k is the root node){r̃_k ← 0. Maintain. } // initiation
9  repeat{ receive messages until a break condition. Maintain.}

11 Maintain:
12 if(((k is the root node) ∨ (CLSTTD has been received)) ∧
13  ¬flg_k^trm){ alc_i^links ← ∅. alc_j^links ← ∅.
14  for each allocation l_{p,o} ∈ alc_k^links of link (p, o) {
15   Decompose l_{p,o}. }
16  DetermineAllocation r̃_i, r̃_j, l̃_{i→j} and l̃_{j→i}.
17  store l̃_{i→j} into alc_i^links. store l̃_{j→i}. into alc_j^links.
18  if(there is no feasible allocation){
19   if(k is the root node){ execute abort sequence. }
20   flg^ok ← false. }else{ flg^ok ← true. }
21  flg_k^okall ← flg^ok ⊗ ⨂_{b,r,l} flg_{b,r,l}^ok.
22   // ⊗ represents the aggregation of tri-state values
23   // (false, indeterminate and true)
24  if(((k is the root node) ∨flg_t^trm) ∧flg_k^okall){
25   flg_k^trm ← true. }
26  if(flg_k^ok){
27   send (CLSTTD, ctx_k^node ∪ {r̃_k}, ctx_k^link ∪ {l_{k,s}}, r̃_b,
28    alc_b^links, flg_k^trm) to b ∈ {i, j}. }
29  if(¬flg_t^trm ∧ flg_k^okall ≠ indeterm){
30  send (CLSTBU, ctx_k^node, ctx_k^link, r̃_k, l_{k,s}, flg_k^okall) to t. } }

32 Receive (CLSTTD, ctx^node, ctx^link, r̃, alc^links, flg^trm)
33  from node t:
34 if(ctx^node ≠ ctx_t^node ∨ ctx^link ≠ ctx_t^link){
35  // clear cache data when context changes.
36  ∀r ∈ R_k, ∀l ∈ L_{i,j},
37   flg_{i,r,l}^ok ← indeterm,flg_{j,r,l}^ok ← indeterm. }
38 ctx_t^node ← ctx^node. ctx_t^link ← ctx^link. r̃_k ← r̃.
39 alc_k^links ← alc^links. flg_t^trm ← flg^trm.

41 Receive (CLSTBU, ctx^node, ctx^link, r̃, alc^link, flg^ok)
42  from node b ∈ {i, j}:
43 if(ctx^node = ctx_t^node ∧ ctx^link = ctx_t^link){
44  f_{b,r̃,alc^link}^ok ← flg^ok. }

46 Decompose l_{p,o}:
47 if(p = i ∨ o = i){ store l_{p,o} into alc_i^links. }
48 else if(p = j ∨ o = j){ store l_{p,o} into alc_j^links. }
49 else if((p, o) originates from single edge (p', q')){
50  l_{p',o'} ← l_{p,q}. Decompose l_{p',o'}. }
51 else { // (p, o) originates from two edges (p', q') and (p'', q'').
52  determine l_{p',q'} and l_{p'',q''} s.t.
53   l_{p,q} = l_{p',q'} + l_{p'',q''} ∧ l_{p',q'}/l^c_{p',q'} ≈ l_{p'',q''}/l^c_{p'',q''}.
54  Decompose l_{p',o'}. Decompose l_{p'',o''}. }
```

Figure 6. Distributed Search on Cluster Tree (Procedures of Node k) (1)

another allocation is chosen. While there are opportunities to employ other distributed backtracking algorithms including efficient backtracking mechanisms, here we apply simple synchronous tree search for the sake of simplicity.

The previous approach shown in Subsection II-C also

```
1  DetermineAllocation r̃_i, r̃_j, l̃_{i→j} and l̃_{j→i}:
2  l̃_i ← Σ_{l∈alc_i^links} l. l̃_j ← Σ_{l∈alc_j^links} l. w* ← ∞.
3  for each r_i ∈ R_i{ for each r_j ∈ R_i{
4   if(r̃_k = r_i + r_j){
5    l_{i→j} ← −(l̃_i + r_i). l_{j→i} ← −(l̃_j + r_j).
6    if(l_{i→j} = −l_{j→i} ∧ −l^c_{i,j} ≤ l_{i→j} ∧ l_{i→j} ≤ l^c_{i,j}∧
7     flg^ok_{i,r_i,l_{i→j}} ≠ false ∧ flg^ok_{i,r_j,l_{j→i}} ≠ false){
8      w ← f_i(r_i) + f_j(r_j).
9      if(w < w*){ w* ← w. r̃_i ← r_i. r̃_j ← r_j.
10      l̃_{i→j} ← l_{i→j}. l̃_{j→i} ← l_{j→i}. } } } } }
```

Figure 7. Distributed Search on Cluster Tree (Procedures of Node k) (2) Notations t, s, i and j are the same as Figure 6.

employs tree search. Note that the search on pseudo trees aggregates values of cost functions in the bottom-up computation. On the other hand, the search methods on the cluster tree find a satisfiable solution since the cost values are already aggregated in the preprocessing.

Figure 6 shows the pseudo code of the search algorithm on the cluster trees. Here, flg_k^{ok}, flg_k^{okall} and $flg_{b,r,l}^{ok}$ represent whether an allocation is satisfiable or not. *Contexts* ctx_t^{node} and ctx_t^{link} represent an allocation for nodes and links in the levels of ancestor and parent nodes. $\tilde{r}_k$ and alc_k^{links} represent an allocation for node k. The root node initiates the computation (line 8). The allocation is sent using CLSTTD messages (line 27). Node k updates the information from its parent node based on the messages (lines 32-39). Then, node k decomposes the allocation for edges excluding the edge between its child nodes (lines 14-15 and 46-54). Based on the allocation, node k determines allocation for $\tilde{r}_i$, $\tilde{r}_j$, $\tilde{l}_{i\to j}$ and $\tilde{l}_{j\to i}$ (line 16 and Figure 7). The satisfiability of the allocation is reported using CLSTBU messages (lines 29-30). The information of satisfiability is stored to $flg_{b,r,l}^{ok}$ (lines 41-44). Backtracking is performed to find other allocations if necessary.

IV. Correctness and Complexity

The proposed method shown above is an inexact method since there are two inexact computations. First, it optimistically aggregates cost functions in the clustering process. Although the minimization resembles dynamic programming, it does not consider consistency between the cluster and other nodes. Second, it optimistically determines shares of resource for two integrated links. The quality and the feasibility of solutions therefore decrease. However, when there are sufficient paths of links in each cluster, the feasibility is possibly preserved.

In the clustering process, each node enumerates $|R_i| \cdot |R_j| \cdot (2l^c_{i,j} + 1)$ combinations of allocations for child nodes i and j in the worst case. To find l of constraint $c_{i,j}$ in Equation (6), the combination contains the capacity of link (i, j). In the allocation process, each node enumerates $|R_i| \cdot |R_j|$ combinations. On the other hand, in the exact solution methods based on pseudo trees, the complexity of computation exponentially grows with the number of lower

neighborhood nodes in each node. Let $|R_k|$, l^c and n denote the size of the domain of cost function $f_k(r)$, the capacity of a link, and the number of lower neighborhood nodes in node k, respectively. Node k then enumerates $|R_k| \cdot (2l^c + 1)^n$ combinations.

V. Evaluation

The proposed method was experimentally evaluated using example problems. As the first evaluation, we designed example problems that consist of sources and sinks. The problems are designed with parameters $\langle n_p, n_l, n_e, l_c, r^l, r^u, p^l, p^u, c^l, c^u \rangle$. n_p, n_l and n_e are the number of sources, sinks and links, respectively. l_c defines the capacity of links. Here, all links have the same capacity. With r^l and r^u, amount of resource r_i that is required by sink node i is defined so that $r^l \leq r_i \leq r^u$. For the sake of simplicity, we used a constant amount r_i of the required resource. Similarly, p^l and p^u define the maximum amount $\overline{r}_j$ of resource that is supplied from source node j. Source node j chooses the amount r_j between 0 and $\overline{r}_j$. c^l and c^u define coefficient value c_j for cost function $f_j(r_j)$ of each source node j. The cost function $f_j(r_j)$ is defined as $c_j \cdot r_j$. Those parameters r_i, $\overline{r}_j$ and c_j were randomly determined based on the unique distribution. Table I shows parameters of problems. Each network is a single connected component. The problem instances contain both feasible and infeasible problems. The results are averaged for fifty instances.

We compared the following methods.

- clst: greedy method on cluster trees without search processing.
- clstsch and clstsch100: tree search methods on cluster trees. clstsch100 limits the maximum size of domain of each cost function $f_i(r)$ to 100 using uniform sampling.
- ptsdsch, psdschfbs, psdsch100: search methods on pseudo trees. psdschfbs returns the first best solution. psdsch100 limits the maximum number of combinations of allocations to 100 in each agent. For several links, constant amounts of resources are allocated in proportion to their capacities. The parameter is chosen so that the experiment of problem (h) can be performed in a practical time frame (about two minutes for each instance).

All methods employ a best-first strategy to choose allocations. We excluded the pre-processing to generate the pseudo trees from evaluations. On the other hand, the clustering processing of the proposed method is evaluated. The simulation is based on message cycles. The number of the cut-off cycle is 10,000. The experiments are performed on a computer with Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz, 6GiB memory, Linux and g++.

Table II shows the number of feasible solutions, the number of infeasible solutions and the number of instances that reached the cut-off cycle. In small problems (a), most search methods find the optimal solutions. While the search methods on cluster trees cause errors in the case of (b), they work relatively well. In the case of (c), exact method psdtree cannot solve most problems within the cut-off cycle. In the case of a larger size of problems, only inexact methods can be applied because psdtree needs huge execution time. While the quality of the results depends on the problems, all methods on cluster trees can execute for those problems. psdtree100 shows poor performance since the setting for large problems (h) is insufficient even in small problems (a).

Table I
Parameters of Problems

problem	n_p	n_l	n_e	l^c	r^l	r^u	p^l	p^u	c^l	c^u
(a)	10	10	19	2	1	1	1	10	1	10
(b)	10	10	21	2	1	1	1	10	1	10
(c)	10	10	23	2	1	1	1	10	1	10
(d)	10	90	99	20	1	2	20	40	1	10
(e)	10	90	120	20	1	2	20	30	1	10
(f)	50	50	99	10	1	2	10	20	1	10
(g)	50	50	120	10	1	2	5	10	1	10
(h)	50	50	250	5	1	2	5	10	1	10

Table II
Number of feasible instances (fsbl), number of infeasible instances (infsbl) and number of instances that reached the cut-off cycle (over)

problem	(a)			(b)			(c)			(d)		
algorithm	fsbl	infsbl	over	fsbl	infsbl	over	fsbl	infsbl	over	fsbl	infsbl	over
psdsch	38	12	0	41	8	1	9	1	40			
psdschfbs	38	12	0	42	8	0	48	1	1			
psdsch100	18	32	0	19	31	0	23	25	2	0	50	0
clst	30	20	0	21	29	0	24	26	0	6	44	0
clstsch100	38	12	0	41	9	0	46	4	0	23	23	4
clstsch	38	12	0	41	9	0	46	4	0	21	22	7
problem	(e)			(f)			(g)			(h)		
algorithm	fsbl	infsbl	over	fsbl	infsbl	over	fsbl	infsbl	over	fsbl	infsbl	over
psdsch100	0	40	10	0	50	0	0	43	7	0	1	49
clst	3	47	0	23	27	0	27	23	0	0	50	0
clstsch100	42	6	2	43	5	2	50	0	0	44	3	3
clstsch	42	4	4	47	0	3	50	0	0	45	1	4

Table III shows the cost value of solutions, the number of iterations (message cycles) and the execution time. Those results are averaged for the instances that are solved by all methods. While the methods on cluster trees return the first best solution, the quality is better than that of psdtreefbs. In the cases of (b) and (c), psdtree needs many iterations due to back edges in the pseudo trees. In all results, the methods on the cluster trees need less execution time than the methods on the pseudo trees.

Table IV shows several values of the methods. Those results are also summarized for the instances that are solved by all methods. The clustering processing requires a relatively larger number of iterations (itrc) than the search processing (itrs). In the case of psdtree, the maximum size of the domain of cost functions (maxr) is less than that of clsttree. However, the size of local problems in each agent (maxcmb) is large because of links for lower neighborhood nodes.

VI. Discussions

In this paper, we addressed basic solution methods based on both pseudo trees and cluster trees. While there are a

Table III
COST VALUES OF SOLUTIONS (COST), NUMBER OF ITERATIONS (ITER) AND EXECUTION TIME (SEC)

problem	(a)			(b)			(c)			(d)		
algorithm	cost	iter	sec	cost	iter	sec	cost	iter	sec	cost	iter	sec
psdsch	1.65	52	0.330	1.49	2623	47.31	1.28	4905	52.69			
psdschfbs	2.19	25	0.151	2.26	47	1.13	2.14	63	1.73			
clstsch100	1.75	147	0.040	1.71	167	0.05	1.61	173	0.07	5.96	855	1.67
clstsch	1.75	147	0.040	1.71	167	0.06	1.61	173	0.07	5.94	1062	1.55
problem	(e)			(f)			(g)			(h)		
algorithm	cost	iter	sec	cost	iter	sec	cost	iter	sec	cost	iter	sec
clstsch100	5.55	1068	2.95	1.52	954	1.54	1.38	653	1.20	1.88	1263	21.67
clstsch	5.55	1112	2.80	1.51	403	0.51	1.39	660	1.17	1.90	1289	21.70

Table IV
NUMBER OF ITERATIONS OF PRE-PROCESSING (ITERC), NUMBER OF ITERATIONS OF SEARCH PROCESSING (ITERS), MAXIMUM SIZE OF DOMAIN $|R_i|$ OF COST FUNCTIONS f_i (MAXR), MAXIMUM SIZE OF LOCAL PROBLEMS (MAXCMB), AND NUMBER OF MESSAGES PER ITERATION (CLSTST, CLSTBU, CLSTTD, VALUE, COST)

problem	(c)							
algorithm			maxr	maxcmb		value	cost	
psdsch			11	67292	(134500)	18	18	
psdschfbs			11	67292	(134500)	14	14	
problem	(c)							
algorithm	iterc	iters	maxr	maxcmb		clstst	clsttd	clstbu
clstsch100	148	24	54	5717	(9945)	26	27	21
clstsch	148	24	54	5717	(9945)	26	27	21
problem	(h)							
algorithm	iterc	iters	maxr	maxcmb		clstst	clsttd	clstbu
psdsch					(1.661E+9)			
clstsch100	998	265	100	1794018	(8.496E+6)	337	173	158
clstsch	998	290	374	5637838	(2.674E+7)	337	173	159

$(\cdot)$ in maxcmb is the theoretical size (without pruning) for all instances. Other results are averaged for solved instances.

number of efficient methods to improve those methods, we focused on the effects of the approximation using the cluster tree.

The methods based on the pseudo tree will also be approximated using similar approaches. However, the approximation needs relatively complex aggregations on the pseudo tree to determine an optimistic allocation of the amount of resource. On the other hand, the idea based on the cluster tree is intuitively simple.

While the hierarchical decomposition is useful, more sophisticated methods will be necessary for large, interconnected and dynamic networks. Such advanced methods should be developed considering topological theory and complex network theory.

The proposed method requires overlay networks. The evaluation of the communication delays on the networks is beyond the scope of this study because it will contain routing protocols.

The above issues should be evaluated in future works. Our current results show that the proposed method is applicable to large and complex problems where the previous methods are not directly applicable.

VII. CONCLUSION

In this work, we presented a solution method for a distributed resource allocation problem motivated by a power supply network containing distributed sources. Our method employs cluster trees on networks instead of pseudo trees. While the proposed method is an inexact method, it works with relatively large problems. The experimental results show the effectiveness of the proposed method.

Our future work will include improvements of search methods for our approach, theoretical analysis of the approximated problems, and more detailed comparison of the approximation methods for both the proposed methods and the methods of pseudo trees.

ACKNOWLEDGMENTS

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

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

[2] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," *Artificial Intelligence*, vol. 161, no. 1-2, pp. 149–180, 2005.

[3] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," in *19th International Joint Conference on Artificial Intelligence*, 2005, pp. 266–271.

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

[5] A. Kumar, B. Faltings, and A. Petcu, "Distributed constraint optimization with structured resource constraints," in *8th International Conference on Autonomous Agents and Multiagent Systems*, 2009, pp. 923–930.

[6] S. Miller, S. D. Ramchurn, and A. Rogers, "Optimal decentralised dispatch of embedded generation in the smart grid," in *11th International Conference on Autonomous Agents and Multiagent Systems*, vol. 1, 2012, pp. 281–288.

[7] T. Matsui and H. Matsuo, "Considering equality on distributed constraint optimization problem for resource supply network," in *2012 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, 2012, pp. 25–32.

[8] T. Matsui, M. Silaghi, K. Hirayama, M. Yokoo, and H. Matsuo, "Resource constrained distributed constraint optimization with virtual variables," in *23rd AAAI Conference on Artificial Intelligence*, 2008, pp. 120–125.

[9] T. Okimoto, Y. Joe, A. Iwasaki, M. Yokoo, and B. Faltings, "Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds." in *17th International Conference on Principles and Practice of. Constraint Programming*, 2011, pp. 660–674.