Distributed Constraint Optimization with Structured Resource Constraints

Akshat Kumar
Computer Science Dept.
University of Massachusetts
Amherst, MA 01003
akshat@cs.umass.edu

Boi Faltings
Artificial Intelligence Lab.
Swiss Federal Institute of Technology, Lausanne
boi.faltings@epfl.ch

Adrian Petcu
SAP Research
Zurich, Switzerland
apetcu@gmail.com

ABSTRACT

Distributed constraint optimization (DCOP) provides a framework for coordinated decision making by a team of agents. Often, during the decision making, capacity constraints on agents' resource consumption must be taken into account. To address such scenarios, an extension of DCOP- *Resource Constrained DCOP*- has been proposed. However, certain type of resources have an additional structure associated with them and exploiting it can result in more efficient algorithms than possible with a general framework. An example of these are distribution networks, where the flow of a commodity from sources to sinks is limited by the *flow capacity* of edges. We present a new model of structured resource constraints that exploits the acyclicity and the flow conservation property of distribution networks. We show how this model can be used in efficient algorithms for finding the optimal flow configuration in distribution networks, an essential problem in managing power distribution networks. Experiments demonstrate the efficiency and scalability of our approach on publicly available benchmarks and compare favorably against a specialized solver for this task. Our results extend significantly the effectiveness of distributed constraint optimization for practical multi-agent settings.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: [Multiagent systems, Intelligent agents]

General Terms

Algorithms, Theory

Keywords

Distributed constraint optimization, Power restoration

1. INTRODUCTION

Distributed constraint optimization (DCOP) provides a framework for multiple agents to coordinate and jointly compute the optimal choice over a set of alternatives encoded as a constraint network [13, 14]. DCOP has many practical applications such as distributed meeting scheduling [10], distributed planning and resource allocation, target tracking in sensor networks [13] etc. Most of the current algorithms in DCOP optimize a single objective function [13, 14, 6, 11, 3]. Often, agents consume resources which have limited capacity that must not be violated. For example, in the distributed meeting scheduling problem agents may have a travel budget which can not be exceeded, rooms for the meeting may have a certain capacity on the number of attendees. Consequently, the search for the optimal solution must also take into account these resource constraints.

To address optimization with resources, an extension of DCOP- Resource Constrained DCOP (RCDCOP)- has been proposed in [2, 12]. It models resources by introducing *virtual variables* corresponding to each resource and imposes n-ary constraints among virtual variables and agents to guide the search mechanism. Adding resources significantly increases the combinatorial aspect of the problem as agents must reason about the global optimality as well as the resources they consume. While RCDCOP is a general framework for handling resources, some problem domains associate additional structure with resources and exploiting it can yield better, efficient algorithms. We examine one such domain of distribution networks and show how its structure can be utilized to counter the additional complexity of resource constraints.

Distribution networks describe the flow of a *commodity* from a source, where it is produced, to sinks. Such networks can model many scenarios such as liquids flowing through a pipe, parts through assembly lines or current through a power network. Each edge in this network is a conduit for the commodity and has a certain *capacity* which caps the amount of flow through it. In addition, it may incur a *cost* when the commodity flows through it. Vertices are the conduit junctions or sinks which may consume some of the incoming flow and forward the rest on a *subset* of its incident edges. This property is also called *flow conservation* and is equivalent to Kirchhoff's current law when the commodity is electric current. Additionally, the flow from the source to sinks must take the form of a *tree* i.e. acyclic, also called *feeder tree* in the context of power distribution networks.

The optimization problem we address in such networks is to determine the configuration of the least cost feeder tree such that every sink in the network gets fed without violating the capacity of any edge, assuming there is at least one such tree. This problem is essential for reconfiguring the network after the network structure gets disrupted due to faults such as line failures in power distribution networks.

Cite as: Distributed Constraint Optimization with Structured Resource Constraints, Akshat Kumar, Boi Faltings and Adrian Petcu, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.
Copyright © 2009, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

The key source of complexity in modeling such networks are the resource constraints (capacity of edges). Our contribution in modeling such resources is twofold. First, we use the property of flow acyclicity and flow conservation to develop an abstract notion of a resource using flow trees. Based on this notion, we provide a DCOP model of the optimization problem introduced earlier. Second, we provide an efficient distributed dynamic programming based algorithm to solve this model using only linear number of message exchanges. Our approach is based on DPOP [14], an existing solver for distributed optimization. Finally, we test on publicly available realistic power restoration benchmarks and compare with an existing specialized solver to show the efficiency of our approach. Next section introduces the motivating power supply restoration problem.

1.1 Power supply restoration

A power distribution system is a network of electric lines connected by switching devices (SDs) and fed by circuit breakers (CBs) [1, 16]. Both SDs and CBs have two device positions: closed, open. SDs are analogous to sinks (transformer stations) which consume some power and forward the rest on other lines if closed. Open SD stops power flow. Circuit breakers, which are analogous to power sources, feed the network when closed. The positions of the devices are set such that the paths taken by the power of each CB forms a tree called *feeder*, and no sink is powered by more than one power line. In addition, Kirchhoff's law (or flow conservation) must hold for all devices and the current load for any line must not exceed its capacity.

The problem of power supply restoration (PSR) is that of reconfiguring the network (setting positions of devices) such that power supply is restored to all affected sinks after one or more power lines become faulty. Typically, restoration also optimizes certain parameters such as minimizing switching operations, line losses etc [15]. These objectives can be modeled using constraints in the DCOP framework.

PSR has received much attention following a series of power blackouts in U.S. and Europe. Power companies like EDF in France [15] and NESA in Denmark [8] have shown much interest in automating this task. Further, DCOP provides an ideal solution to the demands of an increasingly deregulated power market from highly regulated monopolies [7]. Co-generation (several small power sources from wind, solar, thermal etc.) will mean many different interconnected power sources that switch on and off making distribution networks more complex. We need to localize decision making within the network so that it can scale, that it can be more robust when power links might fail, and that it can be safe against central manipulation. Such objectives are readily achievable using the DCOP framework.

2. BACKGROUND

This section briefly introduces the DCOP model and Resource Constrained or Multiply-Constrained DCOP [12, 2], which extend DCOP by adding support for resources.

A discrete distributed constraint optimization problem (DCOP) is a tuple $< \mathcal{X}, \mathcal{D}, \mathcal{F} >$. $\mathcal{X} = \{X_1, ..., X_n\}$ is a set of variables, and $D = \{D_1, ..., D_n\}$ is a set of finite variable domains. $\mathcal{F} = \{f_1, ..., f_m\}$ is a set of functions also called constraints, where each f_i is a function with scope $(X_{i_1}, \cdots, X_{i_k})$, $f_i : D_{i_1} \times .. \times D_{i_k} \rightarrow \Re$, which denotes how much *cost* is assigned to each possible combination of values of the involved variables. In a DCOP, each variable and constraint is owned by an agent. To simplify the notation, we use X_i to denote either the variable itself, or its agent. The goal is to find a complete instantiation $\mathcal{X}^*$ for the variables that *minimizes* the global cost or $\sum_i f_i$ under the assignment $\mathcal{X}^*$.

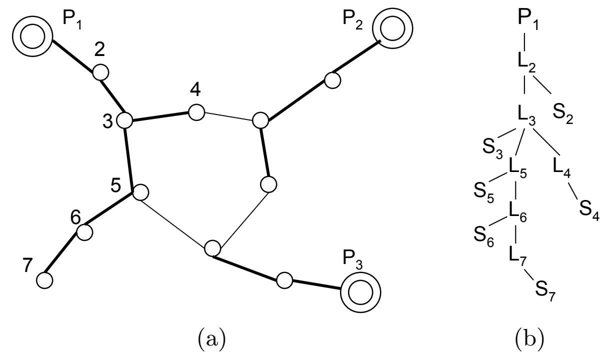


Figure 1: A power network and a feeder tree.

Resource Constrained DCOP (RCDCOP) adds support for resources to the above framework [2, 12]. They are defined by a set R of resources and a set U of requirements. Each resource $r_a \in R$ has a *capacity* $C(r_a) : R \rightarrow \Re$. The set U defines the quantity of resources required by agents i.e. $u_i(r_a, d_i) : R \times D_i \rightarrow \Re$ defines the amount of the resource r_a required by agent i under the assignment d_i. The global resource constraint requires that the capacity of each resource must not be exceeded i.e. $\forall r \in R, \sum_i u_i(r, d_i) \leq C(r)$ under the assignment $\mathcal{X}^*$. An important generalization is that each resource requirement u may take any arity leading to n-ary constraints among agents. This suggests the increased complexity of solving RCDCOP than DCOP.

3. MODELING THE DISTRIBUTION NETWORK

This section describes different aspects of modeling, especially resources, for the problem of finding optimal configuration of feeder trees. It has been shown that the PSR task without the capacity constraints is *easy* and with capacity constraints is NP-hard [8], which further emphasizes the importance of handling resources. The model we develop applies to any general distribution network, but for ease of exposition we will use the PSR problem introduced earlier.

We view a power distribution network as a graph $G = (V, E)$. Vertices represent power sources (CBs) P and Sinks S (SDs). A sink S_i consumes $|S_i|$ units of power. The edges are the electric lines. To remind the reader, the model must enforce the following constraints:

- Acyclicity: The path taken by the power from each source must be a tree. Each sink, line, source must be part of at most one feeder tree.
- Flow conservation: Kirchhoff's law must hold for each device.
- Resource constraints: For each line the current load must not exceed its capacity.

Figure 1(a) shows a power network. The double circled nodes are the CBs, rest are sinks. Thick lines supply power

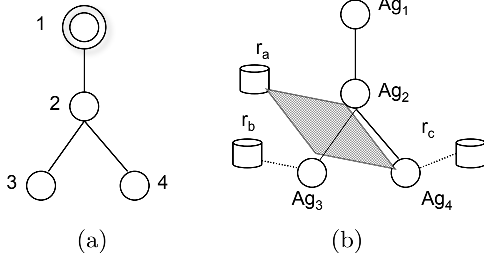


Figure 2: A power network and its equivalent RCD-COP formulation.

to the connected sinks, rest are open and do not propagate power. It is easy to see that the path of power for each source is a tree (Figure 1(b) shows the power path for source P_1), and each sink and line is part of at most one tree.

3.1 Modeling using RCDCOP

In the DCOP framework, we assume that each vertex of the distribution network is owned by an agent, which owns all variables, constraints corresponding to that vertex. Without going into details, let us focus on resource constraints.

Figure 2 shows a network graph and the corresponding RCDCOP formulation at the abstraction of agents. RCD-COP introduces three resources: r_a, r_b, r_c, to model the capacity constraint for each edge. Each agent consumes power equivalent to its sink requirement i.e. $|S_i|$. Kirchhoff's law requires that an edge must carry power for both, the directly connected sink and others which may receive power indirectly through it. In the configuration of Figure 2, edge $(1, 2)$ carries power for both sink 3 and 4 in addition to the directly connected sink 2.

Consequently, the resource for an edge is connected to *all* the agents which may receive power either directly or indirectly through it in *some* configuration, as it is not known in advance *exactly* which of those agents will receive power through it in the optimal configuration. For example, the resource r_a is connected via a 4-ary constraint to Agents 2, 3 and 4 in Figure 2(b). This n-ary constraint also allows to interrupt the search if the capacity of the resource gets violated.

One can easily imagine the complexity of such formulation as the network grows bigger. The resources for edges near the power source will be involved in prohibitively high arity constraints. This complexity arises because the RCD-COP framework cannot model the dependencies between resources introduced by the Kirchhoff's law other than through the introduction of constraints that involve all of them. In a connected network, this eventually leads to a constraint involving all agents and resources, which is prohibitively complex to represent and compute with.

3.2 Modeling using resource quantification

The RCDCOP model can be significantly simplified by realizing that in a power network, it is not the destination of the power but its amount that matters. Thus, from the resource consumption point of view, a configuration where a power line feeds a set of sinks X is equivalent to one where it feeds a set of sinks Y as long as the sum of power drawn by sinks in each set is the same. If all sinks draw about the same power, we quantize the capacity as the number of sinks that are fed through a line. This is the approach taken in [8], where it is assumed that at most 13 sinks can be fed through any line. It implies that we can discretize the resource consumption as a *Load variable* having domain $\{0, 1, \ldots, 13\}$.

After quantifying the resource constraints, the only other requirement is to model the direction of incoming flow for each sink i.e. determining the incident edge through which a sink gets power. We will encode this information in a variable called *Direction variable* for each node in the network. It also models the constraint that each sink gets fed from a single incident power line. Next, we give a formal definition for these variables and constraints.

3.2.1 Variables

Figure 3(a) shows a power network with three nodes. As mentioned earlier, we create two variables: a Load variable L_i, a Direction variable $\mathcal{D}_i$, for each node i in the network graph, as shown in Figure 3(b). For a sink, the Load variable models the amount of incoming flow i.e. the electric current. Since, any sink can be powered by only a single line, the Load variable also models the capacity for that line. The domain of the Load variable is the discretized resource. For a power source, the Load variable models the number of sinks it can feed i.e. $\{0, 1, \ldots, maxSinks\}$.

The Direction variable $\mathcal{D}_i$ models all the possibilities of feeding a node i i.e. the set of neighbors which can forward power to i in *some* configuration. This set can be determined by performing a DFS traversal of the network graph with each power source being the DFS root in turn. For example, in Figure 3(a), sink 2 can receive power either along the path $1 \to 2$ or $1 \to 3 \to 2$. Therefore, its domain becomes $\{1, 3\}$. Figure 3(b) shows the domain for all Direction variables. For the power source, the domain includes only a dummy d.

3.2.2 Constraints

Two types of constraints are created. Tree constraints model the acyclicity of the flow i.e. restrict the power path to be a tree, and also model the optimization criterion. The second type, KCL constraints, model the flow conservation or Kirchhoff's law.

A binary Tree constraint is created between the Direction variable for each node i and Direction variables for other nodes which can receive power from i i.e. have i in their domain set. For example, in Figure 3(a), nodes 2 and 3 can receive power from 1. Hence, $\mathcal{D}_1$ is connected to both $\mathcal{D}_2$ and $\mathcal{D}_3$ in Figure 3(c). Similar constraints are also created for both the nodes 2 and 3 (not shown in figure). Semantically, in the constraint $f_i : \mathcal{D}_i \times \mathcal{D}_j \to \Re$, the first variable corresponds to the sender node i.e. node 1 in Figure 3(c) and second variable corresponds to the receiver i.e. nodes 2 or 3. The valuations $f_i(d_i, d_j)$ are defined as:

- If $d_i = j$ i.e. node i receives power from node j and $d_j = i$ i.e. node j receives power from i, then $f_i(d_i, d_j) = \infty$ as it represents a *loop* which is not allowed.
- If $d_j \neq i$ i.e. node j does not receive power from i under the current assignment, then $f_i(d_i, d_j) = 0$ as the current constraint only models the flow from the node i.

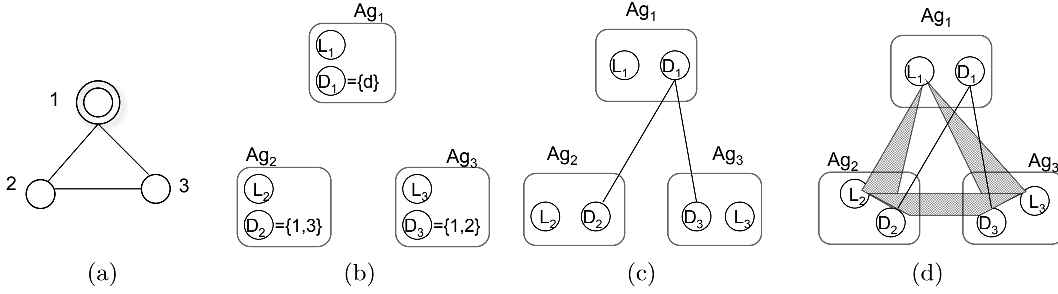


Figure 3: A power network and its quantification based model.

- The remaining case is $d_j = i$. $f_i(d_i, d_j) = cost(i, j)$, which gives the cost of flow along the edge (i, j). In a power network, it becomes the line loss, our optimization criterion.

An n-ary KCL constraint is created among the Load variable for each node i and the Load and Direction variables for other nodes which can receive power from i i.e. have i in their Direction variable domain. For example, in Figure 3(d), the shaded region represents the n-ary KCL constraint among the Load variable L_1 of the sender node 1 and the Load and Direction variables for nodes 2 and 3 which can receive power from 1. The purpose of KCL constraint is to enforce the Kirchhoff's law for each sender node i.

Let R_i be the set of all the Load and Direction variables for the nodes which can receive power from i. For example, $R_1 = \{L_2, \mathcal{D}_2, L_3, \mathcal{D}_3\}$. Let r denote an assignment to the variables in the set R_i, and r_X be the projection of r on the variable X. For example, $r = (2, 1, 3, 1)$ is an assignment for R_1 and $r_{L_3} = 3$. Semantically, in the KCL constraint $f_i : L_i \times R_i \rightarrow \{0, \infty\}$, the first variable corresponds to the Load variable of the sender, followed by the variables in the receiver set. The valuation $f_i(l_i, r)$ can take two values: $0, \infty$, depending upon if the assignment satisfies the Kirchhoff's law or not.

- $f_i(l_i, r) = 0$, if for all nodes k which receive power from i under the current assignment r i.e. $\forall k\ s.t.\ r_{\mathcal{D}_k} = i$, the sum of their corresponding Load variables respects the Kirchhoff's law equation. The equation is $l_i = \sum_k r_{L_k} + |S_i|$, which simply states that the amount of incoming power flow to the node i $(= l_i)$ must be equal to the sum of power consumed at i $(= |S_i|)$ and the amount of power forwarded to other nodes, which is $\sum_k r_{L_k}$.
- $f_i(l_i, r) = \infty$, if otherwise.

The advantage of the above model over the RCDCOP model is that it eliminates the need for very high arity constraints to model the Kirchhoff's law. For any node, the maximum arity of KCL constraint is the number of immediate neighbors, and Tree constraints are always binary. For the RCDCOP model, the n-ary constraint can include all the agents as opposed to just immediate neighbors. This simplification is made possible due to resource quantification which models all possible power flows through a line. However, the quantification also makes the complexity dependent on the problem instance. For example, the model in Figure 3(d) can be solved by DPOP algorithm [14], which has exponential complexity in the induced width of the graph. The induced width for our example is 3, and if the discretized domain for the Load variables is of size k, then the complexity becomes $O(k^3)$. For $k = 13$ it is solvable, but for a different instance, where k may be 1000, then the same model becomes very complex to solve. This limitation affects the scalability and applicability of this model for different instances and other kinds of distribution networks, where quantification may not be even possible.

In the following, we present a further refinement to this model, which uses feeder trees to model the capacity constraints and the flow conservation law, and alleviates the above limitations of the quantification technique.

3.3 Modeling using resource abstraction

To avoid the need for discretizing resources, we go back to modeling the set of sinks that are fed through a link, but in a more structured way. The solution to the flow configuration problem can be thought of as a collection of feeder trees which describe the power path from each power source. For example, the tree in Figure 1(b) describes the power flow from source P_1 in Figure 1(a). The root of this tree is the power source, P_1. The structure of this tree can be defined recursively. Each node L_i of this tree corresponds to the node i in the power network which receives power from the current source, P_1. The children of a node L_i include: the sink of the node i, and nodes L_js which receive power from i. For example, in Figure 1(a), node 3 forwards power to both the nodes 4 and 5. Therefore, the children of L_3 in the feeder tree include L_4 and L_5 as well as its own sink S_3. We call each L_i in such trees a *load abstraction* associated with the node i in the distribution network. L_i symbolizes the incoming flow to the node i and is used to describe feeder tree configurations.

The feeder tree structure offers three distinct advantages. First, the Kirchhoff's law or flow conservation is implicitly modeled. The children of each L_i in this tree model how the incoming power at node i is used: S_i models the consumption for the sink at i, and other children of L_i model where the rest of the power is forwarded. Further, feeder trees reflect that power path has to be acyclic. Second, the capacity constraint for each line can be modeled easily. For this, we evaluate each node of this tree bottom up. For example, in Figure 1(b), $|L_7| = |S_7|$ and $L_6 = |S_6| + |L_7|$, which simplifies to $|L_6| = |S_6| + |S_7|$. Thus, each L_i in this tree can be expressed as a sum of sinks, which can be evaluated easily. To enforce the capacity constraint, we ensure that $\forall i\ |L_i| \leq l_\tau$, where l_τ is the capacity of a line. If a tree does not satisfy this constraint, then it is deemed inconsistent and can not become part of the solution.

Finally, the search for the optimal solution can be modeled as a search for the best configuration in the *space of all possible feeder trees.* This type of search has a distinct advantage- it does not require to quantify the capacity of a line. Thus it offers a *generic model* for all distribution networks and the complexity does not depend on the granularity of the discretization. This change in the search methodology is reflected in the way we will define constraints for this model. The model we described below borrows many concepts from the quantification model, but models constraint valuations differently.

3.3.1 Variables

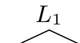
Only the Direction variables are created for each node in the network graph. These variables are analogous to the Direction variables in the quantification model and their domain is defined in the same way (see Section 3.2.1).

3.3.2 Constraints

Since, there are no Load variables, a single n-ary constraint is created, which models both the Tree constraint and the KCL constraint. Similar to the quantification model, this constraint is created among the *Direction* variable (instead of the Load variable, as there are none) for each node i and Direction variables for other nodes which can receive power from i. We define the receiver set R_i as in the quantification model, except that it only contains Direction variables. The semantics of this constraint: $f_i : \mathcal{D}_i \times R_i \rightarrow \Delta \times \Re$, has similar meaning as in the quantification model. The first variable corresponds to the sender, followed by the variables in receiver set. This constraint produces a mapping from the input assignment to the *feeder tree* it represents and the *cost* of this feeder tree. Δ is the space of all possible feeder trees and $\Re$ represents the real valued space of costs associated with such trees. This mapping is the result of change in search methodology, which is now performed in the space of feeder trees. $f_i(d_i, r)$ is defined as:

- $= (\phi, \infty)$, if the assignment represents a loop (for details see Tree constraint in Section 3.2.2).
- $= (\delta, c_\delta)$, where δ is a *depth* 1 feeder tree and c_δ is the cost of this tree. The root of δ is L_i, the load abstraction associated with the sender node i. The children of L_i include the sink S_i for the node i and all other nodes which receive power from i under the current assignment r i.e. $L_k \in \text{Children}(L_i)$ iff $r_{\mathcal{D}_k} = i$, and the cost is $c_\delta = \sum_k cost(i, k)$.

It is easy to see that the above constraint represents Kirchhoff's law. Root L_i denotes the incoming power flow to the node i. Its children describe how this power is consumed: S_i denotes the own consumption at i and L_ks denote the nodes where the rest is forwarded. For example, consider the constraint $f_1(\mathcal{D}_1 = d, \mathcal{D}_2 = 1, \mathcal{D}_3 = 2))$ corresponding to sender node 1 in Figure 3, which implies only $\mathcal{D}_2$ receives power from 1 under the current assignment. Hence, the feeder tree mapping is:



. The cost of this tree is $cost(1, 2)$.

In contrast to the RCDCOP model where an n-ary constraint can include all the nodes in the power network, the maximum arity of constraints in this model is the number of *immediate neighbors* for any node. The reduction in complexity is easily evident. This is made possible as the abstraction based model can handle Kirchhoff's law using partial feeder trees, whereas RCDCOP resorts to making n-ary constraints that involve all the nodes because it can not implicitly model the dependencies induced by Kirchhoff's law.

3.3.3 Inconsistent and Dominated tree pruning

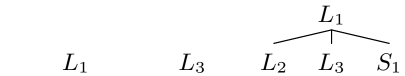
For the capacity constraints, the evaluation of each node in a tree must be smaller than the capacity threshold l_τ, otherwise the tree is inconsistent and can be pruned. However, for some trees this evaluation can not be performed. For our last example, $|L_1| = |L_2| + |S_1|$. Since we do not know L_2 at this point, L_1 can not be evaluated. Such trees are called *partial feeder trees*, as the complete power path from L_2 is not known, and are retained until they become *complete* i.e. all the leaves are sinks, implying we can evaluate all nodes. The pruning can still be accelerated by considering the bounds on the evaluation of such trees. For the last example, it holds that $|L_1| \geq |S_1|$ and if $|S_1| > l_\tau$, then this tree can be safely pruned. Next, we describe the domination based pruning which will become useful during the dynamic programming step for solving this model.

When there are two alternative *complete* feeder trees that a) have the same root and b) feed the same set of nodes, then the one with higher cost can be eliminated as it is dominated and will never be part of an optimal solution. The dominance based pruning holds only for *complete* trees, not for partial trees. We do not give a formal proof due to space constraints. However, the intuitive reason is that the only uncertainty in such trees is how the root gets fed, while the downstream path till the destination sinks is known. Out of various options of feeding an *isolated subnetwork* via a common node, the one which has the lower cost is always preferred and can be the part of an optimal solution.

For the optimization task i.e. minimizing $\sum_i f_i$, the $\sum$ operator has to be redefined for feeder trees, as now each f_i is a mapping to trees instead of real valued costs. This operator also has an intuitive meaning in the context of trees. Each f_i gives a *partial* feeder tree. The $\sum$ operator provides a way to combine two partial trees, and its repeated application will reduce the partial nature of trees until we get the solution tree, which describes the complete power path from source to destination sinks.

3.3.4 Combining feeder trees

Two trees δ_i and δ_j rooted at L_i and L_j are *combinable* if the root of one tree occurs as a leaf node in the other tree and other than that they do not have any common node. The former is called the *child* and the latter is called the *parent*. The combined tree, δ_k, is given by embedding the child tree in the parent. The cost of δ_k is the sum of the cost of combining trees. The $\sum$ (sum) operator applies to two feeder trees that are either combinable or disjoint, and produces their combination or a collection of disjoint trees as appropriate. For example, in Figure 3, the sum of $f(\mathcal{D}_1 = d, \mathcal{D}_2 = 1, \mathcal{D}_3 = 1)$ and $f(\mathcal{D}_3 = 1, \mathcal{D}_2 = 1)$ is given by



. In this example, the second tree, $L_3 \rightarrow S_3$, is the child, as its root L_3 occurs as a leaf in the first tree. It is easily observable that the $\sum$ operator reduces the partial nature of trees. For example, in the combined tree, the power path from L_1 till

the sink S_3 is known, which was known neither in the first tree nor in the second.

It turns out that any two feeder trees, δ_i, δ_j, that are selected for join by the $\sum$ operator are either disjoint or always satisfy the conditions for *combination*. Due to space constraints we give an informal proof sketch. Let us suppose that the two trees share a node L_k which violates the combination conditions. Since, the root node can not be shared (if it was, then it implies we are adding the same constraint twice, which is not possible), L_k must be an internal node or a leaf. Let $L_{k'}$ be the first common node on the path from L_k (including L_k) to the root in both the trees such that its parent in the both the trees is different i.e. $L^{\delta_i}_{P(k')} \neq L^{\delta_j}_{P(k')}$. It can be shown that such a node always exists. This condition implies that node k' receives power via different sources as implied in the first and second tree configuration. This is not possible, as the Direction variable for k' enforces that it should receive power from a single node in both the configurations. Hence, this is a contradiction, implying the two trees never violate the combination property.

In the next section, we describe a dynamic programming based approach to find the optimal flow configuration using the above model. It is based on DPOP [14], an efficient algorithm for solving DCOP.

4. DYNAMIC PROGRAMMING OPTIMIZATION

As constraints for the abstraction model give a mapping to the space of feeder trees, existing dynamic programming algorithm DPOP [14] can not be used. The algorithm we propose is based on DPOP, but changes its bottom up util propagation phase to deal with feeder trees.

The *primal graph* of a constraint network is an undirected graph with variables as vertices and an edge connects variables that appear in the scope of same constraint function. Our algorithm works on the DFS traversal of the primal graph for the given constraint network. DFS trees have already been investigated as a means to boost search [5, 4]. Due to the relative independence of nodes lying in different branches of the DFS tree, it is possible to perform search in parallel on independent branches, and then combine the results. In a DFS tree, $P(X_i)$ refers to the parent of the node X_i, $C(X_i)$ to the children, $PP(X_i)$ to the pseudo parents of X_i and Sep_i to the seperator of X_i. For DFS related definitions see [14].

As with DPOP, our algorithm has three phases. We will describe in detail only the UTIL phase, which is significantly different from DPOP.

Phase 1 - a *DFS traversal* of the graph is generated using a distributed algorithm (see [14]). The outcome of this protocol is that all nodes consistently label each other as parent/child or pseudoparent/pseudochild, and edges are identified as tree/back edges. The DFS tree thus obtained serves as a communication structure for the other 2 phases of the algorithm.

Phase 2 - *UTIL propagation*: This is a bottom up pass on the DFS arrangement in which the utility information is aggregated and propagated from the leaves towards the root (in the form of UTIL messages), from each node to its parent through tree edges but not back edges. A UTIL message sent by a node X_i to its parent $P(X_i)$ is a multidimensional matrix which informs $P(X_i)$ how much utility the subtree rooted at X_i provides for different assignments of values to the variables that define the separator Sep_i for the subtree. In our case, the utility for an assignment u is the feeder tree and its cost i.e. $util(u) = (\delta, -c_\delta)$ (costs can be treated as negative utilities). In more detail, the agents perform following steps:

1. Wait for UTIL messages from all their children, and store them.
2. Perform an *aggregation*: join messages from children, and also the relations they have with their parents and pseudoparents.
3. Perform an *optimization*: project themselves out of the resulting join by picking their optimal values for each combination of values of the other variables in the join.
4. Send the result to parent as a new UTIL message.

Aggregations apply the JOIN operator and optimization applies the PROJECT operator as described below. Let UTIL^j_i denote the message sent by X_i to its parent X_j and $dim(\text{UTIL}^j_i)$ denote its dimensions i.e. the set of variable defining this message.

Definition 1 (JOIN). *The $\oplus$ operator: $UTIL^j_i \oplus UTIL^j_k$ is the join of two UTIL matrices. It has dimensions as $dim(UTIL^j_i) \cup dim(UTIL^j_k)$. The value of each cell in the join is obtained by applying the $\sum$ operator to the corresponding cells in the source matrices.*

The above join operator easily extends to feeder trees as we have defined the $\sum$ operator which combines feeder trees (see Section 3.3.4).

Definition 2 (PROJECT). *The $\perp$ operator: if $X_i \in dim(UTIL^j_i)$, then $UTIL^j_i \perp_{X_i}$ is the projection through optimization of the $UTIL^j_i$ matrix along the X_i axis: for each instantiation u of the variables in $\{dim(UTIL^j_i) - X_i\}$, all the corresponding values from $UTIL^j_i$ (one for each value of X_i) are tried, and the one which gives maximal utility (or minimal cost) is chosen. That is, the utility $util_\perp(u)$ for the instantiation u in the matrix $UTIL^j_i \perp_{X_i}$ is given by $util_\perp(u) = \max_{X_i} util^j_i(u, x)$.*

However, we can apply the above projection operator only in the case of complete feeder trees (which have all leaves as sinks). This optimization is a case of domination based pruning as described in Section 3.3.3. In the following we define a new operator and highlight new features of our algorithm different from DPOP.

For partial feeder trees, the projection operator can not be used due to *capacity constraints*. The reason is that in partial trees some nodes can not be evaluated, thus it is not possible to check if the capacity constraint is violated or not. If we project the variable X and select the best assignment $X = x^*$ for each u, then we risk that the corresponding feeder tree δ of $util^*(u)$ will violate the capacity constraint as the message moves up in the DFS tree. This will force us to backtrack to the initial suboptimal choice of X. This problem does not occur for complete trees, as in complete trees each node, including the root can be exactly evaluated and capacity constraints can be enforced.

To avoid such backtracking, we provide a $CompositeProject$ operator which is applied when conditions for domination based pruning are not satisfied.

DEFINITION 3 (COMPOSITEPROJECT). *The $\perp^c$ operator: if $X_i \in dim(UTIL_i^j)$, then $UTIL_i^j \perp^c_{X_i}$ is obtained by the projection of X_i from $UTIL_i^j$ as follows: for each instantiation u of the variables in $\{dim(UTIL_i^j) - X_i\}$, a tuple of UTIL values is formed in which each value corresponds to an instantiation of X_i. That is, the utility $util_{\perp^c}(u)$ for the instantiation u in the matrix $UTIL_i^j \perp^c_{X_i}$ is given by $util_{\perp^c}(u) = \langle util_i^j(u, x_1), \ldots, util_i^j(u, x_k)\rangle$, where each x_i is an instantiation of X_i which has domain of size k.*

For applying any operator such as $\sum$ to such tuple of values, we apply it individually to each value in this tuple. The side effect of *CompositeProject* operator is that combining partial feeder trees gives rise to combinatorial explosion as messages pass up the DFS tree. We use a number of techniques to counter this effect. For example, a complete feeder tree affects the the global solution only through its root (the only uncertainty in such trees is how the root gets fed, rest of the power path is known). Thus, in a UTIL message we replace each such tree with a single node, its root, and the cost of the total tree. The rest of the tree is stores locally at the current agent and an indexing scheme is used to associate these stripped off trees with their root during the final VALUE phase. As the UTIL messages move up in the DFS tree more information is accumulated and many partial trees become complete due to the application of $\sum$ operator. Thus, this scheme can provide significant savings. Further pruning is achieved by inconsistent and dominated tree pruning to keep space requirements within manageable limits as shown in the experiments section.

Phase 3 - *VALUE propagation*: This phase is a top-to-bottom pass that assigns value to variables, with decisions made recursively from the root down to the leaves. This phase is initiated by the root agent once it has received all UTIL messages from all of its children. Based on these UTIL messages, the root assigns to its variable X_0, the value $v^\star$ that maximizes the sum of its own utility and that communicated by all its subtrees. In case of power networks, it selects the assignment to the Direction variable that provides the minimum cost feeder tree. It then sends the value message VALUE($X_0 \leftarrow v^\star$) to every child. The process continues recursively to leaves. At the end of this phase, the algorithm finishes, with all variables being assigned their optimal values. For a distribution network, these assignments define the minimum cost feeder tree. As per the assignment to Direction variables, the position of the corresponding devices are set.

5. EXPERIMENTS

We use the power network configuration benchmarks made publicly available by Hadzic *et al.* [8]. These are realistic power network instances provided by their contractor NESA, a power distribution company in the Copenhagen area, Denmark. We compare against their binary decision diagram (BDD) based solver. For these instances, it was possible to quantify the line load to a discrete range $[0, 13]$. We tested both, the quantification based model and the resource abstraction model on a windows workstation with Intel dual core 2.4 GHz cpu, 1GB RAM. The quantification based model had poor scalability, it could solve only the smallest instance with 200MB space in around 30 min. We did not implement the RCDCOP model because of its obvious complexity. The abstraction based model easily scaled to solve large instances within few seconds (the execution time is important, as in power networks when the lines fail, only a few minutes are available to the operator to reconfigure the network, otherwise a blackout will happen).

Instance Name	Time (sec)	Max Size (Kb)	Total Size (Kb)	BDD (Kb)
std-diagram	0.6	13	50	7.3
1-6+22-32	0.9	58	121	31
Complex-P1	1	75	300	252
Complex-P2	1.1	98	525	418
1-32	1.3	155	808	4154
Large	99	3900	36000	-
Complex	-	-	-	-

Table 1: Space and runtime statistics

We enlist the modeling advantage our abstraction provides over the BDD based approach. First, the BDD based solver is quantification based and thus, suffers from the limitations of such an approach. Their solver's complexity is problem domain dependent and can not extend to settings which require a large quantified domain for resources. Further, it is not clear that how BDDs can be extended to other kinds of distribution networks as they use power network domain knowledge. On the other hand, our approach searches in the space of feeder trees, which are generic to model any kind of distribution network. Feeder trees can model problems where quantification may not be even possible, and its complexity remains independent of the problem instances. Second, their approach is centralized and precomputes all the solutions offline without any optimality criterion. For evolving and dynamic power markets their approach is not suitable. Our model provides all the advantages of a distributed approach and can adapt readily to dynamic settings.

Experimentally, we will compare the space usage of our approach to compute the best configuration and the space complexity of their solver. However, it should be noted that their approach compiles the solutions offline, while our approach works in real time. Therefore, Hadzic *et al.* [8] do not report the runtime for their offline compilation.

Table 1 shows the maximum message size exchanged between any two agents, total message size and the execution time for our algorithm. The last column shows the results reported in [8]. A - means that the instance was too large for the algorithm to solve. For smaller and medium sized instances (till Complex-P2) our algorithm was fast with the largest message size a few Kbs. The instance Large which has two power sources, 66 lines and 56 sinks and many cyclic paths proved harder. The original BDD based approach too could not compile this instance. They used several domain specific ordering and scheduling heuristics to compile this instance to 8483Kb. We did not employ any special technique and still solved the task within a reasonable time. However, the instance Complex went beyond our reach. The next section describes some techniques that allowed us to nevertheless solve this instance.

5.1 Network Decomposition

We used a similar technique as in [8]. We divided the network graph into 5 overlapping regions such that every vertex belongs to exactly one region and each edge can be

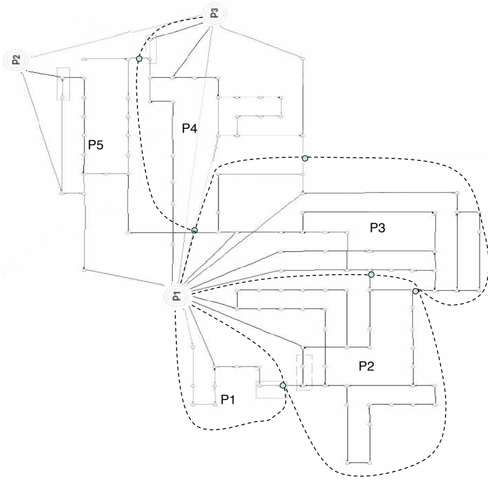


Figure 4: Complex: the largest test instance

part of at-most two regions, as shown in Figure 4. Next, we provided feedback to our algorithm by giving information about the shared edges's state i.e. determining the direction of flow along that edge. After this step we were able to run inference on all five regions (P1-P5) *independently.* The max message size for P1 was 8Kb (t=.4sec), for P2 1.9Mb (t=8sec), for P3 8Mb (t=102sec), for P4 2.6Mb (t=10sec) and for P5 1Mb (t=2sec). Note that, treating regions independent of each other sacrifices the global optimality in exchange for speedy network configuration.

6. CONCLUSION AND FUTURE WORK

In this work we addressed an important concept of resource constraints in distributed constraint optimization. We examined the domain of distribution networks, which associate a structure with resources in the form of Flow conservation law and flow acyclicity. RCDCOP, the general framework for handling resource constraints, can not handle such resources as it fails to utilize the structure of such resources. We provided two models which overcome this limitation of RCDCOP. One is based on resource quantification which discretizes the resources. This model is simple to implement, but fails to generalize and showed limited scalability in experiments. However, based on the insights from the quantification model, we provided a generic model for handling resources in distribution networks. Our approach, based on feeder trees, is generic enough to model any kind of distribution network and unlike quantification based approach, is problem domain independent. Using this model, we presented a distributed, dynamic programming based algorithm to solve the optimal flow configuration problem. Experimentally, we could solve industrial power network configuration instances and provided competitive performance with a specialized solver for this problem.

Our work advances the applicability of the DCOP framework and shows that by exploiting the problem structure, DCOP algorithms can scale well to solve real life problems. In our current and future work, we are working to further increase the scalability by applying state-of-the-art techniques in constraint networks such as AND/OR search and decision diagrams in DCOP algorithms [9].

7. REFERENCES

[1] P. Bertoli, A. Cimatti, J. Slanley, and S. Thiebaux. Solving power supply restoration problems with planning via symbolic model checking. In *ECAI*, pages 576–580, Lyon, France, 2002.

[2] E. Bowring, M. Tambe, and M. Yokoo. Multiply-constrained distributed constraint optimization. In *AAMAS*, pages 1413–1420, Hakodate, Japan, 2006.

[3] A. Chechetka and K. Sycara. No-Commitment Branch and Bound Search for Distributed Constraint Optimization. In *AAMAS*, Hakodate, Japan, May 2006.

[4] R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[5] E. C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(14):755–761, 1985.

[6] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward-bounding for distributed constraint optimization. In *ECAI*, pages 103–107, 2006.

[7] J. Griffin and S. Puller. *Electricity deregulation: choices and challenges.* University of Chicago Press, Chicago, 2005.

[8] T. Hadzic, A. Wasowski, and H. Andersen. Techniques for efficient interactive configuration of distribution networks. In *IJCAI*, pages 100–105, Hyderabad, India, 2007.

[9] A. Kumar, A. Petcu, and B. Faltings. H-DPOP: Using hard constraints for search space pruning in DCOP. In *AAAI*, pages 325–330, 2008.

[10] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS*, 2004.

[11] R. Mailler and V. Lesser. Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *JAIR*, 25:529–526, 2006.

[12] T. Matsui, H. Matsuo, M. Silaghi, K. Hirayama, and M. Yokoo. Resource constrained distributed constraint optimization with virtual variables. In *AAAI*, pages 120–125, 2008.

[13] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AI Journal*, 161:149–180, 2005.

[14] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, Edinburgh, Scotland, Aug 2005.

[15] S. Thiebaux and M. Cordier. Supply restoration in power distribution system - a benchmark for planning under uncertainty. In *ECP*, pages 85–96, 2001.

[16] S. Thiebaux, M. Cordier, O. Jehl, and J. Krivine. Supply restoration in power distribution system - a case study in integrating model-based diagnosis and repair planning. In *UAI*, pages 525–532, 1996.