Supply Chain Coordination through Distributed Constraint Optimisation *

David A. Burke[1], Kenneth N. Brown[1], Mustafa Dogru[2**], and Ben Lowe[2**]

[1] Centre for Telecommunications Value-chain Research and
Cork Constraint Computation Centre,
Department of Computer Science, University College Cork, Ireland
[2] Bell Labs Ireland, Alcatel-Lucent, Dublin, Ireland
d.burke@4c.ucc.ie,k.brown@cs.ucc.ie,{dogru,blowe}@alcatel-lucent.com

Abstract. Supply chain coordination involves planning and coordinating a range of activities across the supply chain, among several business units. These business units are naturally distributed and we assume that they operate autonomously, but they need to be coordinated in order to effectively meet end user demand and avoid waste. Distributed Constraint Optimisation (DisCOP) considers algorithms explicitly designed to handle such problems, providing support for coordinating agents (autonomous nodes or business units) in the supply chain. In this paper we describe the supply chain coordination problem and present it as a new benchmark problem suitable for modelling and solving using DisCOP. In this problem each agent has a large complex problem with many public variables (variables constrained with other agents), and this characteristic poses new challenges to DisCOP algorithms. To deal with this, we introduce a number of modelling and search techniques, which we use with the ADOPT DisCOP algorithm in order to solve the supply chain problem. We perform experiments on problem instances of varying complexity to demonstrate the effectiveness and limitations of our approach.

1 Introduction

Many combinatorial decision problems are naturally distributed over a set of agents: e.g. coordinating activities in a sensor network [1], or scheduling meetings among a number of participants [2]. Distributed Constraint Reasoning (DCR) [3] considers algorithms explicitly designed to handle such distributed problems. Algorithms search for globally acceptable solutions, balancing communication costs with processing time, while allowing private information to be kept local to each agent. Many different algorithms have been proposed that consider both satisfaction (DisCSP) [3, 4] and optimisation (DisCOP) [5–7].

In this paper we make two contributions. Our first contribution is to propose the Supply Chain Coordination (SCC) problem as a new benchmark for DisCOP. SCC involves the planning and coordinating of production and delivery

* This work is supported by Science Foundation Ireland under Grant No. 03/CE3/I405
** Supported partly by Industrial Development Agency (IDA), Ireland

schedules among several agents (business units). This benchmark problem is a useful addition as it considers problems where agents have large internal problems with many variables. This is in contrast to other existing benchmarks for DCR (e.g. SensorDCSP [1], Graph-colouring [7], Meeting Scheduling [2]) that consider single or small numbers of variables per agent. A characteristic of the problem is that each agent has many public variables (variables constrained with other agents), which leads to an extremely large distributed search space, posing significant challenges for existing DisCOP algorithms. In our second contribution, to address these challanges, we propose modelling techniques and search strategies that can effectively prune the initial search space of this problem and allow the remaining search space to be traversed more efficiently. Through implementation with the ADOPT DisCOP algorithm, we apply these techniques and perform a number of experiments demonstrating the strengths and limitations of our approach.

2 Distributed Constraint Optimisation Problem

A Distributed Constraint Optimisation Problem (DisCOP) consists of a set $A = \{a_1, a_2, ..., a_n\}$ of *agents*, and for each agent a_i, a set $X_i = \{x_{i1}, x_{i2}, \ldots, x_{im_i}\}$ of *variables* it controls, such that $\forall i \neq j \ X_i \cap X_j = \emptyset$. Each variable x_{ij} has a corresponding domain D_{ij} of possible values. $X = \bigcup X_i$ is the set of all variables in the problem. $C = \{c_1, c_2, \ldots, c_t\}$ is a set of *constraints*. Each c_k has a *scope* $s(c_k) \subseteq X$, and is a function $c_k : \prod_{ij:x_{ij} \in s(c_k)} D_{ij} \rightarrow I\!N$. The *agent scope*, $a(c_k)$, of c_k is the set of agents that c_k acts upon: $a(c_k) = \{a_i : X_i \cap s(c_k) \neq \emptyset\}$. An agent a_i is a *neighbour* of an agent a_j if $\exists c_k : a_i, a_j \in a(c_k)$. For each agent a_i, $p_i = \{x_{ij} : \forall c \ x_{ij} \in s(c) \rightarrow s(c) \subseteq X_i\}$ is its *private* variables – variables which are not constrained by other agents' variables – and $e_i = X_i \backslash p_i$ is its *public* variables – variables that have constraints with other agents. A *global assignment*, is the selection of one value for each variable in X. A *local assignment*, to an agent a_i, is an assignment to X_i. The global objective function, F, assigns a cost to each global assignment: $F : \prod_{ij} D_{ij} \rightarrow I\!N :: g \mapsto \sum_k c_k(g_{\downarrow s(c_k)})$. An optimal solution is one which minimises F. The solution process, however, is restricted: each agent is responsible for the assignment of its own variables, and thus agents must communicate with each other in order to find a global solution.

ADOPT [7] is a memory-bounded, complete DisCOP algorithm that allows agents to work asynchronously. Agents are first prioritised into a tree, whereby each agent maintains lower and upper bounds for the subtree rooted at that agent. The algorithm proceeds by higher priority agents choosing assignments to their variables and sending these to lower priority neighbours. Lower priority agents respond with lower and upper bound costs that their subtree will incur based on the proposed assignments. As the search progresses, the lower and upper bounds on costs will be tightened in each agent until the lower bound of the minimum cost solution is equal to its upper bound. If an agent detects this condition, and its parent has terminated, then an optimal solution is found and it may terminate also.

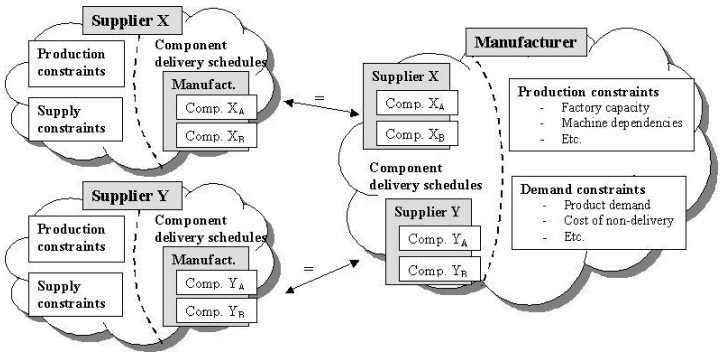


Fig. 1. Basic supply chain coordination scenario

3 The Supply-chain Coordination (SCC) Benchmark

3.1 Motivation

Supply chain management involves planning and coordinating a range of activities across the supply chain. The supply chain typically consists of several interdependent agents (organisations or business units – from the same or different companies), each holding the responsibility for provision of particular components that are combined in a final product. In order to reduce costs, each agent may try to optimise its internal processes (planning, scheduling, etc.); however by making decisions locally and independently, the actions may lead to inefficiencies in the wider supply chain. Thus, there is a need for agents to coordinate their actions: "as market pressures dictate that today's commercial supply chains provide rapid and efficient supply, the need to coordinate with all organisations within the supply chain is becoming increasingly critical" [8]. By coordinating the actions of all participants, more efficiencies can be gained, reducing inventories, lead times and costs and improving quality and service levels. While in some cases the competitive nature of business may restrict coordination possibilities, some environments will allow certain levels of coordination, e.g. (i) coordination of business units within the same company; (ii) an alliance of independent firms who wish to cooperate in order to compete better in the market; or (iii) a dominant firm using its position to encourage cooperation in the supply chain, e.g. the use of Vendor Managed Inventory by Wal-Mart [9].

To optimise the actions of all agents in the supply chain, it is often difficult to centralise the problem because of one or more of the following reasons: (i) each agent is a seperate business unit and may be unwilling to share local information; (ii) each agent's internal problem is large and complex, meaning that the cost involved in centralising this information is prohibitive and attempting to solve such a problem centrally leads to large monolithic models of great complexity; (iii) each agent's local low level problems (such as factory scheduling) are more suited to be solved at that agent's location where detailed information is readily available. For these reasons, a decentralised approach is necessary.

Problem parameters
D_{it} : the amount demanded for product i in period t,
B_{ij} : the number of components j needed to produce one unit of product i,
I'^{m}_{j} : the manufacturer's opening inventory level for component j,
Y'^{m}_{i} : the opening inventory level for product i,
l_i : the length of time required to build product i,
r_i : the time taken to deliver an order for product i to a customer,
C^m_t : the manufacturer's factory capacity (total time) available at period t,
M : the maximum quantity of products that can be built in any period,
k_j : the maximum number of batches for component j,
y_j : the cost of delivering a single order for component j,
s_j : the batch size for component j,
h^m_i : the holding cost for a single unit of product/component i for one period,
w_i : the setup cost for the manufacturer for product i,
v_i : the setup time for product i,
p_i : the cost of not meeting an order for product i

Public decision variables
O^m_{jt} : the number of batches delivered for component j in period t

Private decision variables
o_{jt} : a {0,1} variable indicating if there is an order for component j in period t,
b_{it} : a {0,1} variable indicating if any of product i will be built in period t,
m_{it} : the quantity of product i manufactured at time t, $m_{it} \in \mathbb{N}$

Auxiliary variables
ξ : the total cost,
A^m_{jt} : the number of component j arriving at period t, $A^m_{jt} \in \mathbb{N}$
q_{it} : the quantity of product i for dispatch in period t, $q_{it} \in \mathbb{N}$
I^m_{jt} : the closing inventory level for component j at period t, $I^m_{jt} \in \mathbb{N}$,
Y^m_{it} : the closing inventory level for product i at period t, $Y^m_{it} \in \mathbb{N}$

Fig. 2. Manufacturer model – input parameters and variables.

3.2 Scenario Description

Consider a simple 2-tier supply chain (Fig. 1). At the top of the chain is a manufacturer producing a number of products over a fixed planning horizon. In Figures 2 and 3, we give an example model for the manufacturers problem.[3] In each period t of the planning horizon, the manufacturer has a specific demand D_{it} from customers for each of its products i. To produce the products, the manufacturer must order components from its suppliers. Components are delivered in batches (e.g. pallet, container or truckload), where s_j is the quantity/batch size. A single order O^m_{jt}, incurring a fixed cost y_j, specifies the number of batches of a particular component j that must be delivered in period t. The manufacturer must coordinate with the suppliers to decide how to schedule the

[3] The superscripts m and s are used to distinguish between similar parameter and variable names in the manufacturer and suppliers' models.

utility function

$$\xi = \sum_{\forall t} \sum_{\forall i} \sum_{\forall j} [o_{jt} y_j + h_i^m Y_{it}^m + h_j^m I_{jt} + b_{it} w_i + p_i (D_{it} - q_{it})] \quad (1)$$

production constraints

$$Mb_{it} >= m_{it} \qquad \forall t, \forall i \quad (2)$$

$$\sum_{\forall i} (l_i m_{it} + b_{it} v_i) \leq C_t^m \qquad \forall t \quad (3)$$

supply/delivery constraints

$$A_{jt} = s_j \times O_{jt}^m \qquad \forall t, \forall j \quad (4)$$

$$\sum_t O_{jt}^m \leq k_j \qquad \forall j \quad (5)$$

$$k_j o_{jt} >= O_{jt}^m \qquad \forall t, \forall j \quad (6)$$

$$q_{it} \leq D_{i(t+r_i)} \qquad \forall t, \forall i \quad (7)$$

$$I_{j0}^m = I_j'^m \qquad \forall j \quad (8)$$

$$Y_{i0}^m = Y_i'^m \qquad \forall i \quad (9)$$

$$I_{jt}^m = I_{jt-1} + A_{jt}^m - \sum_{\forall i} B_{ij} m_{it} \qquad \forall t, \forall j \quad (10)$$

$$Y_{it}^m = Y_{it-1} + m_{it} - q_{it} \qquad \forall t, \forall i \quad (11)$$

Fig. 3. Manufacturer model – utility function and constraints.

orders/deliveries. When negotiating these schedules, the agents have to consider production constraints. We assume the manufacturer has a single production facility for producing all products, and there is a setup cost, w_i, and time v_i associated with starting production of a product. Each product takes a specific length of time to build l_i. In any period, the manufacturer will decide how much of each product it will build, m_{it}. Given that each product is independent, the sequence of these productions is irrelevant and so can be done one after the other. In this case, there will at be at most one setup cost per period per product, indicated by the {0,1} variable b_{it} which is forced to be correctly set according to (2). A capacity constraint (3) states that the total production in any period cannot exceed the total factory capacity, C_t^m, for that period. The closing component inventory in any period is equal to the previous days closing inventory plus the number of components arriving less the number of components used in that period – what components go into each product is defined by the bill of materials, B_{ij} (10). The closing product inventory is calculated in a similar manner (11). Note that the latter two constraints also have the effect of ensuring that q_{it} is less than or equal to the products available on any particular day. Holding costs are charged for each item in storage and we assume that these costs incorporate charges for both excess and obsolete items. The number of components that arrive is dependent on the assignment to O_{jt}^m (4), which is also constrained by (5).

Problem parameters
I'^s_j : the opening inventory level for component j,
R'^s_j : the opening inventory level for raw material j,
l_j : the length of time required to build component j,
r_j : the time taken to deliver an order for component j to the manufacturer,
C^s_t : the supplier's factory capacity (total time) available at period t,
M : the maximum number of components that can be built in any period,
k_j : the maximum number of batches for component j,
s_j : the batch size for component j,
A^s_{jt} : the quantity of raw material for j arriving at period t,
h^s_j : the holding cost for a single unit of component/raw material j for one period,
w_j : the setup cost for component j,
v_j : the setup time for component j

Public decision variables
O^s_{jt} : the number of orders delivered for component j in period t

Private decision variables
b_{jt} : a {0,1} variable indicating if any of component j will be built in period t,
m_{jt} : the number of component j manufactured at time t, $m_{jt} \in \mathbb{N}$

Auxiliary variables
ξ : the total cost,
R^s_{jt} : the closing inventory level for raw material of component j at period t, $R^s_{jt} \in \mathbb{N}$
I^s_{jt} : the closing inventory level for component j at period t, $I^s_{jt} \in \mathbb{N}$

Fig. 4. Supplier model – input parameters and variables.

The variable o_{jt} is used identify the periods in which order costs are incurred (6). The quantity of products dispatched in any period cannot be greater than the demand from customers on the corresponding delivery date (dispatch date + delivery time) (7). We assume that the manufacturer will deliver its products on time or not at all. However, there is a cost associated with non-delivery (i.e. profits missed out on). The utility function, (1), calculates costs that arise for the manufacturer from (i) order costs (ii) holding costs (iii) missed customer orders; and (iv) production setup.

The supplier model is described in Figures 4 and 5. The supplier has a decision variable O^s_{jt}, which is constrained to be equal to the corresponding manufacturers variable (O^m_{jt}). The suppliers internal problem is similar to the manufacturers, except we are assuming that a known, fixed amount of raw materials for producing components will arrive in each time period A^s_{jt} (supply constraints). The supplier has a capacity constraint (14) on what it can produce in any given period. The inventory at the end of any period has to be non-negative, when the number of components dispatched to the manufacturer is subtracted – the number of components dispatched in each period is calculated to be the number of batches that have to be sent (taking into account the delivery time) multiplied by the batch size (19). The raw material inventory is also constrained to be

utility function

$$\xi = \sum_{\forall t} \sum_{\forall j} [h_i^s R_{jt}^s + h_j^s I_{jt}^s + b_{jt} w_j] \tag{12}$$

production constraints

$$Mb_{jt} >= m_{jt} \qquad \forall t, \forall j \tag{13}$$

$$\sum_{\forall j} (l_j m_{jt} + b_{jt} v_j) \leq C_t^s \qquad \forall t \tag{14}$$

supply/delivery constraints

$$\sum_{t} O_{jt}^s \leq k_j \qquad \forall j \tag{15}$$

$$R_{i0}^s = R_i'^s \qquad \forall i \tag{16}$$

$$I_{j0}^s = I_j'^s \qquad \forall j \tag{17}$$

$$R_{jt}^s = R_{jt-1}^s + A_{jt}^s - m_{jt} \qquad \forall t, \forall j \tag{18}$$

$$I_{jt}^s = I_{jt-1}^s + m_{jt} - (s_j \times O_{j(t+r_j)}^s) \forall t, \forall j \tag{19}$$

Fig. 5. Supplier model – utility function and constraints.

non-negative (18). The setup costs are calculated as in the manufacturer model using the variable b_{it} (13). The supplier's utility function, (1), is the sum of the holding costs and production setup costs. The overall objective of the agents is to coordinate their schedules such that the total costs in the supply chain network are minimised.

We make a number of assumptions in this scenario, but it is possible to extend it to consider more details if required. Production scheduling could be extended to allow multiple production facilities, dependencies, etc., if desired. We assume unlimited storage space, but the scenario can easily be extended to include storage limits. A more sophisticated cost model with customers, incorporating various late-delivery charges could also be included if necessary. Finally, it is also possible to include further tiers in the supply chain, i.e. sub-suppliers delivering goods to the suppliers. Agents representing the sub-suppliers can be introduced and the supply constraints of the suppliers can be replaced with delivery schedules that should be agreed with these sub-suppliers.

3.3 Benchmark Parameters

To instantiate the models we provide parameter settings, chosen to be representative of real-world scenarios, in Table 1. The ranges for each parameter allow a variety of different SCC instances to be considered, e.g. high/low product demand, underconstrained/overconstrained factory capacities, different ratios of holding cost to penalty cost etc. The benchmark specification is also currently being finalised for inclusion in CSPLib [10]. This will include OPL models for each agent type and a number of problem instances and their solutions.

Table 1. Parameter ranges used for generating problem instances.

Symbol	Description	Parameter
H	planning horizon	4–12
s_j	batch size for component j	50–100
D_{it}	amount demanded for product i in period t	10–120
C_t^m,C_t^s	factory capacity (total time) available at period t	50–100
$I_j'^m$,$Y_i'^m$, $R_j'^s$,$I_j'^s$	opening inventory levels for raw material/components/products	0–100
l_i,l_j	length of time required to build component/product	1–2
r_i,r_j	time taken to deliver a component/product order	0–2
h_j	holding cost for a single unit of component j for one period	1–25
h_i	holding cost for a single unit of product i for one period	$\sum_{\forall j} B_{ij} h_j^m$ +/- 5
w_j,w_i	setup cost for the production of component j/product i	$h_j, h_i \times 100$
v_i,v_j	setup time for component/product manufacturing	5–10
o_j	cost of single order for component j	20–50 * s_j
p_i	cost of not meeting an order for product i	250–500
A_{jt}^s	quantity of raw material for j arriving at period t	50–100

4 Modelling and Solving the Problem as a DisCOP

4.1 Basic algorithm

We will use the ADOPT algorithm to coordinate the decision making of the agents. To integrate the local solving process with the distributed search we use a compilation approach [11]. In a preprocessing phase before the distributed search begins we find the optimal local solution for each valid combination of assignments to the public variables. Each compiled value contains a unique assignment to the public variables (public assignment) and the optimal local cost for that assignment. It is not necessary to store the assignments to the private variables as these are not required by the distributed search. Once the distributed algorithm is finished, we then rediscover the optimal assignment to the private variables for the chosen public assignment. There are a number of motivations for using compilation: (i) it can be done offline and independently by each agent – given the nature of this application, both time and memory are available to the agents for this; (ii) it acts as a caching mechanism, which avoids repeated local search – i.e. once an optimal local solution has been found for a particular public assignment then this search will not need to be performed again (except for the final chosen public assignment); and (iii) all local cost information is calculated prior to the distributed search beginning – this reduces the computation and time required for the distributed search to execute.

Consider the supply chain in Fig. 6 (a) with a single manufactuer M producing 2 products, and three suppliers $S1$, $S2$ and $S3$ producing 4 components. The agents are prioritised such that agent M is the root of the priority tree and

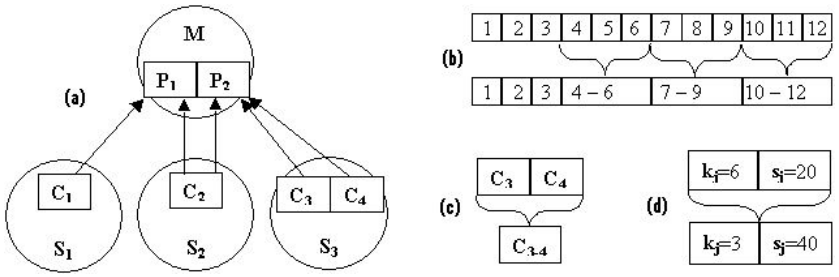


Fig. 6. (a) Example SCC problem, 1 manufacturer, 3 suppliers. Product P_1 consists of components C_1, C_2. Product P_2 consists of components C_2, C_3, C_4. (b) The number of periods to consider in the planning horizon can be reduced by aggregating periods together. (c) Components from the same supplier, used in the same product, can be aggregated to reduce the number of components that need to be considered. (d) The number of batches to schedule can be reduced by combining two or more batches, e.g. the batch size s_j is doubled to consider pairs of batches together.

agents $S1 - S3$ its children. To begin solving the problem, M finds the local assignment that gives it the lowest potential cost. Since M has not yet received cost information from its children, the lower bound for each subtree is 0 and the upper bound is ∞ for all assignments, therefore it chooses the assignment that has the lowest local cost and this cost becomes the lower bound for M. M then sends its public assignment to its children in the priority tree (the suppliers). Agents $S1 - S3$ calculate the cost that they would incur if they agree to this schedule and inform M of this cost. M now has the full cost for the current assignment, which is an upper bound on the optimal cost. Considering this new cost information, M then performs a new search for its best assignment. Since the current assignment now has known costs for the suppliers, it is likely that a different assignment now has the lowest potential cost and so is chosen. This new assignment is then sent to the suppliers and the search continues in this manner until M's lower bound is equal to its upper bound. At this stage an optimal solution has been found and the algorithm can terminate.

This standard search process, while simple and adequate, is far from efficient. The reason for this is the complexity caused by the public decision variables of the agents. The number of public variables of the agent is $ct = 48$, where c is the number of components and t is the number of periods. The domain size of the public variables is k – the maximum number of batches to schedule for each component, i.e. k batches of each component will ensure that all customer demand can be met. This means that the number of possible combinations of assignments to the public variables is huge – k^{ct}. If we assume that $t = 12$ (e.g. 12 weeks/1 quarter) and $k = 6$, then k^{ct} >2e+37). However, not all of these assignments need to be considered, since the total number of batches for each component over the planning horizon does not need to be greater than k. This reduces the number of public assignments to approximately 1e+17. In the worst case, the manufacturer would have to suggest each of its 1e+17 possible public assignments to its suppliers in order to find the optimal solution.

This a problem common to DisCOPs with complex local problems where the agents have many public variables. Search based algorithms like ADOPT are faced with exponentially large search spaces. An alternative inference based algorithm, DPOP, is faced with exponentially large memory requirements. Hybrid search/inference versions of DPOP [5, 12] reduce the memory problems, but are still impacted by the increasing search space. In the following sections we propose two methods (*aggregation* and *pre-propagation*) that can be used to reduce the the domain of public assignments, and an additional technique (*dynamic value ordering*) that can be used to efficiently process the domain of public assignments. The first two are general, algorithm-independent techniques, while the third is specific to ADOPT + compilation. All methods may be of use in other problem domains.

4.2 Aggregation

As the planning horizon, number of components and number of batches per component are increased, the domain of public assignments increases. If we aggregate these variables, we can reduce the number of possible public assignments.

Planning Horizon Aggregation: The 12 period horizon from our example is too large to deal with effectively. To reduce this we aggregate periods of the planning horizon. In Fig 6 (b) we consider the initial 3 periods as they are, but we combine the remaining periods in blocks of 3. This allows us to still produce a detailed delivery schedule for the near future, while being less accurate in the longer horizon. As the problem is solved periodically, the part of the planning horizon that is considered in detail will shift with each execution. In our proposed approach we do not modify the agents local problem, i.e. each agent still deals with 12 periods. However, the periods are combined for the purpose of the distributed search, thus reducing the number of public variables. To ensure correctness, deliveries are assumed to be made on the first of the aggregated periods, e.g. if the aggregated delivery variable for the periods 4–6 is set to 2, then suppliers must ensure that 2 batches can be delivered in period 4 while the manufacturer assumes that 2 batches will be delivered in period 4. The remainder of the agents' models still function considering all 12 periods.

Component aggregation: There is also scope for aggregating components. Two components from the same supplier for the same product could be treated as one, since the manufacturer will always need these components in equal amounts. In our example in Fig 6 (c), C_3 and C_4 can be aggregated.

Batch aggregation: To aggregate batches, two or more can be combined to reduce the number of batches that have to be considered. In Fig 6 (d), we group batches into pairs, thus doubling the batch size.

In our example, we reduce the number of public assignments down to 7.2e+11 through aggregation of the planning horizon, down to 7.8e+8 when the component aggregation is included, and down to 592,704 when we aggregate the

batches. However, it is important to note that the resulting solutions will only be optimal for the aggregated problem rather than the original (aggregated solutions provide an upper bound on the optimal solution cost for the original problem), and so aggregation should be used sparingly.

4.3 Pre-propagation

We can further reduce the number of delivery schedules to be considered, through introduction of a propagation phase prior to search. For the suppliers several schedules may be infeasible. E.g. unless it has a large opening inventory it may not be able to deliver all batches in the first period. The supplier can calculate an upper bound (based on the opening inventory level, daily factory capacity and component manufacturing time) for the number of batches it can deliver for each period t of the planning horizon (Eqn. 20). The upper bound on batches for each period can then be propagated up the supply chain network to the manufacturer, who can add an additional constraint to its model (Eqn. 21).

$$\kappa_{jt} = \Big(I^s_{j0} + \sum_{t' \in 1..t} \frac{C^s_{t'} - v_j}{l_j}\Big)/s_j \tag{20}$$

$$\sum_{t' \in 1..t} O_{jt'} \leq \kappa_{jt} \qquad \forall t, \forall j \tag{21}$$

We can also propagate information down the supply chain, taking into consideration what batches are required by the manufacturer. The supply chain will never benefit from the manufacturer receiving more batches for a component than it needs to produce its expected demand. If we consider each period of the horizon in turn and then just look at future demand from that period on, we can determine the maximum number of batches that the manufacturer will want to receive in that horizon. (Eqn. 22). The manufacturer can add a constraint to its model (Eqn. 23), thus pruning public assignments that are dominated and can never be part of an optimal solution. The upper bounds can also be propagated to the suppliers, who can add a similar constraint to their model.

$$\lambda_{jt} = \Big(\sum_{\forall i, t' \in t..H} D_{it'} B_{ij}\Big)/s_j \tag{22}$$

$$\sum_{t' \in t..H} O_{jt'} \leq \lambda_{jt} \qquad \forall t, \forall j \tag{23}$$

In our example, let us assume that each supplier has no opening inventory and can manufacture half of a batch size ($\frac{s_j}{2}$) in each period. The maximum number of batches that the supplier can deliver in each period is then $\{0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6\}$. If the product demand is such that the manufacturer also requires half of a batch size in each period to meet its demand, then the maximum number of batches that the manufacturer requires over the planning horizon is $\{6, 6, 5, 5, 4, 4, 3, 3, 2, 2, 1, 1\}$. Taking these two constraints into account, the domain of public assignments for the manufacturer is now 46,656.

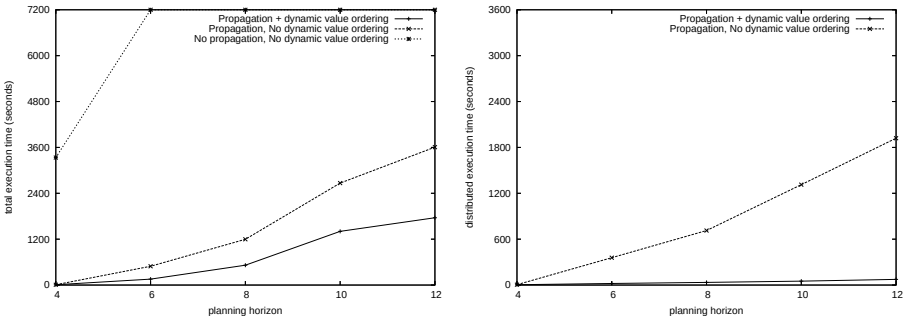
Finally, any agent that is building a product containing more than one component can further prune its search space. E.g. the manufacturer has two components being used in each product. No solution should be considered that will leave the inventories imbalanced by greater than a batch size. This is because if there is greater than a single batch left over at the end of the horizon, there is guaranteed to be a better solution with one less batch (this is assuming a linear cost model, i.e. no economies of scale). In this case, the constraint is specific to the agent concerned and related to the BOM for a product. For our example, we can use Eqn. 24, which when implemented leaves the manufacturer with 11,406 public assignments to consider.

$$\begin{aligned}&\left((\textstyle\sum_{\forall t} O^m_{C1,t} s_{C1}) - s_{C1} + (\sum_{\forall t} O^m_{C3,t} s_{C3}) - s_{C3}\right) < \left(\textstyle\sum_{\forall t} O^m_{C2,t} s_{C2}\right) < \\ &\left((\textstyle\sum_{\forall t} O^m_{C1,t} s_{C1}) + (\sum_{\forall t} O^m_{C3,t} s_{C3}) + s_{C2}\right)\end{aligned} \tag{24}$$

Note that all public assignments (and their corresponding local solutions) that we have pruned are dominated, i.e. they could never be part of an optimal solution. This pre-propagation phase could be seen as a special form of i-consistency. Consistency mechanisms for use during search in distributed constraint satisfaction have previously been proposed to consider pairs of variables (arc-/local-consistency) [13, 14]. Our mechanism is a stronger consistency that considers sets of variables of size $\leq i$, where i = planning horizon. We add sum constraints (21,23) over these sets, and propagate between agents the bounds to be used in the constraints. In this particular case, the propagation follows the supply chain topology, with the bounds calculated in Eqn. 20, propagated up the supply chain from leaf suppliers to the manufacturer, with each agent waiting for all propagation messages from agents below it in the supply chain, before calculating its own bounds and propagating them further up. The bounds calculated in 22 are propagated in a similar manner in the opposite direction. A general consistency mechanism for DPOP [15] could also be adapted for ADOPT. However, we achieve greater pruning than this by removing dominated as well as infeasible assignments. Also, by propagating information on pruned assignments to neighbouring agents prior to search we further reduce the search space.

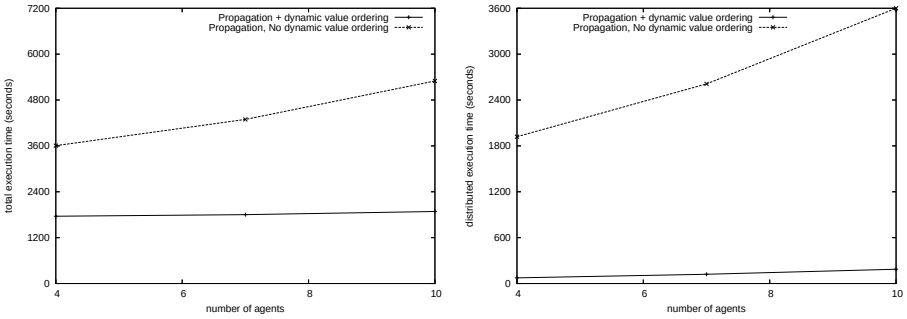
4.4 Dynamic Value Ordering

At this stage, the domain of public assignments has been significantly reduced. However, it is still quite large, and how we process it can have significant performance impacts. The root agent, M, will always make proposals suggesting the assignment that gives it the best lower bound. In the basic algorithm, there is no specific order placed over public assignments, and each assignment will need to be examined in turn to find the assignment that currently has the best lower bound. We can improve on this by keeping track of the currently known lower bound for each public assignment, and maintaining a sorted list of assignments that is updated everytime new cost information becomes available. The agent can then always just choose the assignment at the top of the list, eliminating the need to evaluate all the public assignments.



(a) Total execution time (b) Distributed execution time

Fig. 7. (a) Pre-propagation reduces the number of public assignments that have to be considered by each agent, thus reducing execution time. Dynamic value ordering improves the results further. (b) The benefit of dynamic value ordering can be seen clearer if we just compare the distributed search time, i.e. exclude compilation time.



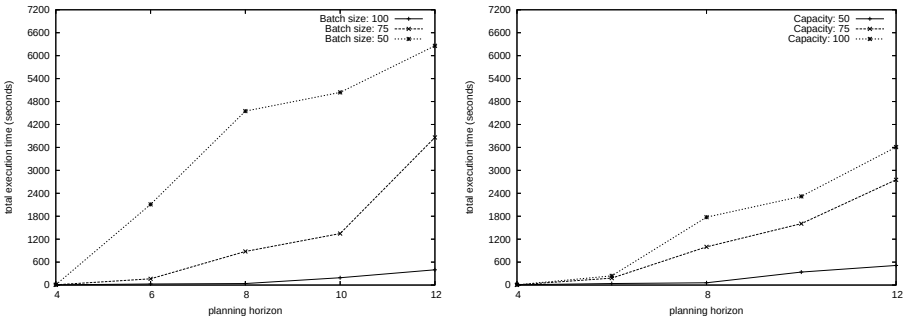
(a) Total execution time (b) Distributed execution time

Fig. 8. Dynamic value ordering allows our approach to scale well as additional agents are added into the supply chain by reducing the required distributed search time.

5 Experimental analysis

To demonstrate that DisCOP can be used to solve the SCC problem, we perform a case-study considering a 'W' supply chain topology. In this scenario the manufacturer produces 2 products. Each product is made from 2 components, one of which is common to both products. Thus, there are 3 components, each produced by a different supplier. We consider planning horizons of $4, 6, 8, 10$ and 12 periods, but aggregate to 6 periods for the latter 3. The remaining parameters are generated randomly from within the ranges specified in Table 1, and each of our results is averaged over 10 problem instances. The test cases are executed in a properly distributed environment, with each agent on its own machine. This environment allows time to be used as a accurate metric for evaluation.

In Fig. 7 we can see that the basic algorithm performs poorly, reaching an imposed cut-off of two hours when more than 4 periods are considered. As the planning horizon increases the pruning of the search space in our pre-propagation



(a) Batch size (b) Capacity

Fig. 9. Parameters that affect the number of possible delivery schedules that have to be considered by the agents affect the runtime. E.g. Batch size and factory capacity.

phase proves essential. Even when pre-propagation is applied, the number of public assignments in the compilation increases as the planning horizon is extended, but we can see that dynamic value ordering allows the distributed search phase of the algorithm to scale gracefully. In Fig. 8, we extend our experiements to consider 7 agents (a single sub-supplier for each supplier) and 10 agents (two sub-suppliers for each supplier). Again, dynamic value ordering is shown to be important to allow the distributed search scale. It should also be noted that for 10 agents and 12 periods, the average time to find the (aggregated) optimal solution using our best method is 1884 seconds. If this system was to be deployed we would envisage it being run overnight, which would allow us to deal with much larger problems (or the existing problem with fewer aggregations).

While for most problem instances an optimal (aggregated) solution could be found, some proved too difficult to solve within the time limit. In particular, the search space is increased by parameter settings that increase the number of batches that the agents have to consider scheduling. E.g. the smaller the batch size, the greater the number of batches and public assignments that have to be considered (Fig. 9 (a)) – although in this case batch aggregation could be used to reduce the problem. Other parameters that effect the number of public assignments to be considered are lead time, raw material availability, ratio of capacity to processing cycles (Fig. 9 (b)) and customer demand. Each of these parameters affects the level of pre-propagation that can be done. Suppliers that are under-constrained (e.g. high capacity) will have more feasible schedules to consider. Manufacturers that have high customer demand will require more batches, which also leads to more schedules that have to be considered. Future work will investigate incomplete search methods for these harder problem instances.

6 Conclusion

We have presented the Supply Chain Coordination (SCC) problem as a benchmark for distributed constraint optimisation. We plan to make this benchmark

available in CSPLib, including a formal specification, agent models and problem instances. Unlike many other DisCOP benchmarks, this problem deals with agents that have large local problems. Each agent also has several variables constrained with other agents, which poses complexity difficulties that have not previously been examined in DisCOP. To reduce the complexity we have proposed an aggregation method that can be applied when modelling the problem. To further prune the search space we also advocate a propagation phase prior to search that eliminates infeasible and dominated solutions in each agent. Finally, we introduce a dynamic value ordering mechanism that allows the agents to efficiently search through a compiled cache of their many possible public assignments. In our initial case-study we have demonstrated that we can solve problem instances of varying size and complexity. We have also identified problem settings that our approach can not find an optimal solution for in a reasonable time. Future work will consider incomplete search mechanisms that are more suited to solving these problem instances and we will also extend our experiments to consider a wider range of supply chain network topologies.

References

1. Béjar, R., Domshlak, C., Fernàndez, C., Gomes, C., Krishnamachari, B., Selman, B., Valls, M.: Sensor networks and distributed CSP: communication, computation and complexity. Artificial Intelligence **161**(1-2) (2005) 117–147
2. Wallace, R., Freuder, E.: Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. Artificial Intelligence. **161**(1-2) (2005) 209–227
3. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. Autonomous Agents and Multi-Agent Systems **3**(2) (2000) 185–207
4. Bessière, C., Maestre, A., Brito, I., Meseguer, P.: Asynchronous backtracking without adding links: a new member in the ABT family. A. I. **161** (2005) 7–24
5. Petcu, A., Faltings, B.: MB-DPOP: A new memory-bounded algorithm for distributed optimization. In: Proc. IJCAI. (2007) 1452–1457
6. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward-bounding for distributed constraints optimization. In: Proc. ECAI. (2006) 103–107
7. Modi, P., Shen, W., Tambe, M., Yokoo, M.: Adopt: Asynchronous distributed constraint optimization with quality guarantees. A. I. **161**(1–2) (2005) 149–180
8. Xu, L., , Beamon, B.M.: Supply chain coordination and cooperation mechanisms: An attribute-based approach. Journal of Supply Chain Mgmt. **42**(1) (2006) 4–12
9. Ş. Selçuk Erengüç, Simpson, N., Vakharia, A.J.: Integrated production/distribution planning in supply chains: An invited review. European Journal of Operational Research **115**(2) (1999) 219–236
10. Gent, I.P., Walsh, T.: CSPLib (2007) http://www.csplib.org.
11. Burke, D., Brown, K.: Efficient handling of complex local problems in distributed constraint optimization. In: Proc. ECAI. (2006) 701–702
12. Petcu, A., Faltings, B.: O-DPOP: An algorithm for open/distributed constraint optimization. In: Proc. AAAI. (2006) 703–708
13. Hamadi, Y.: Optimal distributed arc-consistency. In: Proc. CP. (1999) 219–233
14. Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Consistency maintenance for ABT. In: Proc. CP. (2001) 271–285
15. Kumar, A., Petcu, A., Faltings, B.: H-DPOP: Using hard constraints to prune the search space. In: Proc. Distributed Constraint Reasoning, IJCAI. (2007) 40–55