Discrepancy-based Optimization for Distributed Supply Chain Operations Planning

Jonathan Gaudreault[1,2], Jean-Marc Frayret[1,2], Gilles Pesant[1]

[1] École Polytechnique de Montréal, Montréal, Canada
[2] FORAC Research Consortium, Université Laval, Québec, Canada
{Jonathan.Gaudreault, Jean-Marc.Frayret, Gilles.Pesant}@polymtl.ca

Abstract. This paper studies the case of a supply chain made up of autonomous facilities. They need to coordinate their manufacturing operations in order to optimize customer satisfaction. Although the coordination space can be described as a tree, simple coordination mechanisms used by industry allow them to visit only the first leaf. We show how they can implement a distributed search in order to evaluate alternative solutions. While chronological backtracking can be easily implemented in a distributed framework (e.g. *Synchronous Branch and Bound*); it is not the same for other backtracking strategies such as *Limited Discrepancy Search* (LDS). We therefore propose MacDS, a novel mechanism that allows agents to implement a search strategy based on discrepancies (e.g. LDS or others), while allowing concurrent computation. Use of this mechanism improved the quality of solutions and computation time for both real industrial problems and generated problems.

Keywords: Supply Chain Operations Planning, Multi-agent, Distributed Optimization, Discrepancy

1 Introduction

Supply Chain Management (SCM) deals with the coordination of manufacturing and logistic activities between autonomous business units [1]. The field is very large, including concepts such as the design of distribution network, the evaluation of pricing and penalty policies regarding their ability to encourage collaboration, profit sharing mechanisms, etc. This paper focuses on the coordination of manufacturing operations on a day-to-day basis. This problem is sometimes referred to as Supply Chain Operations Planning (SCOP) [2]. We will propose a new coordination mechanism to address this situation.

To introduce this dynamic, Fig. 1 illustrates a supply chain made up of independent factories offering products to other factories. In this example, an external client announces a call for bids and the cooperation of each factory in the supply chain is needed to produce and deliver the final good. Different alternatives are possible regarding the parts to use, the manufacturing processes to follow, the scheduling of operations and the choice of transportation.

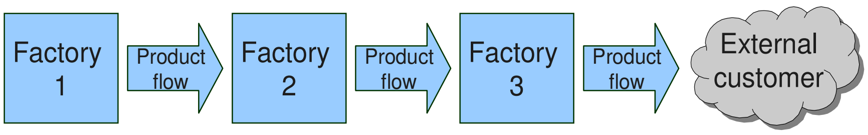

Fig. 1. Example of a simple supply chain.

The partners in the supply chain wish to develop a common production plan (e.g. what to do, where and when). The common objective function represents the client's interest, e.g. minimize lateness. This supply chain is in competition with other supply chains. Consequently, if the external client rejects the first proposal, it is vital to propose alternative solutions before the client accepts a proposition from another consortium. Our goal is then to produce good solutions quickly, but aiming for the optimal solution.

The partners may also be competing against each other for other projects. Therefore, privacy is an important issue and each factory plans its own activities. Moreover, each factory has limited knowledge about the production processes available to the other factories. In other words, each factory has a local vision: it can only think about its own production, inputs and outputs.

This situation cannot be addressed as a centralized optimization problem. The problem is distributed by nature and its resolution must also be distributed (as each factory wants to plan its own operations).

Moreover, some characteristics of the businesses relationships limit our options regarding the choice of a coordination mechanism, such as:

- The information that agents are willing to exchange may be limited.
- Roles and responsibilities in the business network may be established *a priori*.
- The planning abilities of the agents could also be defined *a priori*.

In other words, the chosen coordination mechanism must be compatible with other elements of the business relationship. Section 2 provides more details on this concept. It then presents the main approaches proposed in supply chain management literature and used by industry.

In Section 3, we explain how the most common coordination mechanisms can be generalized while preserving the same business relationships between the agents. According to this view, the coordination space can be represented as a tree. Distributed search in this tree can be done using a basic algorithm such as *Synchronous Branch and Bound* (SyncBB) [3].

Section 4 presents the main contribution of this paper: MacDS, a method which allows agents to systematically search the solution space, but aiming at producing good solutions first by applying a backtracking strategy based on discrepancies (e.g. *Limited Discrepancy Search* [4], or others), while allowing the agents to work concurrently.

In Section 5, SyncBB and MacDS are evaluated using real industrial problems and generated data. Both algorithms produce major improvements in solution quality when compared to current practice, with a strong advantage shown by the MacDS method.

Finally, Section 6 provides a comparison between the present work and the more generic framework of Distributed Constraint Optimization Problems (DCOP).

2 Supply Chain Operations Coordination

In industrial practice, relationships between collaborating business partners are regulated by a set of rules. These rules define what we term the *logistic framework*. This *logistic framework* is often the object of a long term agreement between the partners. It can be formal or informal, negotiated or *ad hoc*. The coordination mechanism is only one of the elements of this framework; however it must be compatible with the other elements.

The *logistic framework* can specify the following elements (Fig. 2): (i) the subject of the business relationship [exchanged products, minimum quantities, responsibilities and ownership of inventories, penalties, profit sharing, etc.]; (ii) the information businesses agree to exchange [stocks levels, sales forecast, order bills, etc.]; (iii) the coordination mechanism itself, stating how the relationship is operationalized [rules and protocols for information sharing, rules specifying which decisions must be made, by whom and when].

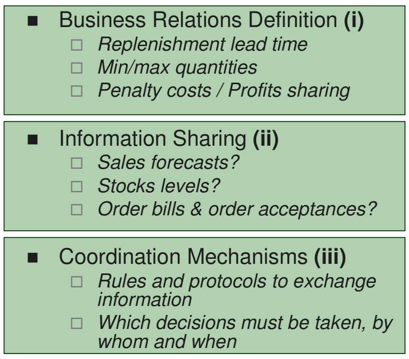
Business Relations Definition **(i)**
Replenishment lead time
Min/max quantities
Penalty costs / Profits sharing
Information Sharing **(ii)**
Sales forecasts?
Stocks levels?
Order bills & order acceptances?
Coordination Mechanisms **(iii)**
Rules and protocols to exchange information
Which decisions must be taken, by whom and when

Fig. 2. Elements of a *logistic framework*.

The negotiation of the agreement and its acceptance by the business partners depends on strategic considerations, which are not the subject of this work. However, a company will only accept an agreement if it is rational to do so. Also, the company will have to manage its day-to-day operations in conformance with the agreement. This means it will have to use local algorithms or local decision models to make the decisions under its jurisdiction (for example, plan its own production). The agent must therefore have access to the necessary algorithms. All these considerations must be taken into account when it comes to setting up a collaboration process.

2.1 Common Approaches

The literature in Supply Chain Management (SCM) is plentiful, but more limited regarding the coordination of the manufacturing operations on a day-to-day basis [5]. In this section, we will briefly review the literature concerning the main components of the *logistic framework* defined in Fig. 2.

At one end of the spectrum, many researchers propose specialized algorithms for the agents to make their local decisions (ex: scheduling) with no consideration for the other aspects of the problem. These algorithms have generally made their way into industrial applications, being integrated into software packages (*Enterprise Resource Planning* software, *Advanced Planning and Scheduling* systems, etc).

At the other end of the spectrum, many researchers study the key characteristics of good business relationships (i). For example, they study which incentive policies (such as penalty costs) can be used to direct the behavior of the cooperating units (see [6]).

Other researchers study the impact of improved information sharing (ii) on the performance of the agents. For example, Schneeweiss studies in [1] the performance of the supply chain as a function of the accuracy of the information accessible to agents. Moreover, there are numerous studies concerning the sharing of inventory levels to calculate forecasts and security stocks, but these works subscribe more to the field of inventory management rather than production management.

Concerning the coordination mechanism itself (iii), the traditional SCM literature is rather limited to practices found in real life supply chains, with a few exceptions (e.g. responsibility tokens [7]). Recent literature in SCM proposes more advanced coordination frameworks involving various forms of negotiation and information exchange schemes aimed at synchronizing planning decisions through the supply chain (e.g. [5,8]). Most of the time, these approaches apply in the contexts where agents have different objectives/interest but wish to agree on a plan, are defined for binary partnership only, or the method involves a mediator agent. Finally, we see the increasing use of multi-agent technologies in order to create advanced systems for enterprises collaboration [9]. This community particularly emphasizes the design of interaction protocols for agent coordination (see [10] for a review). But, most of these protocols can be described as coordination heuristics.

The most common class of coordination heuristics (both in literature and industrial practice) is the 'hierarchical' approach [2]. In this approach there is a sequence (naturally defined or specified in the long term agreement) specifying an order in which the partners must plan their operations. Schneeweiss describes many problems using real industrial applications in [11]. The most common hierarchical approach is *upstream planning* [8,12]. Agents plan their operations one after the other, beginning with the agent that is closest to the customer (Fig. 3). This presupposes that each agent is able to satisfy the demand of the previous one. Therefore, the agreement must specify maximum quantities that can be ordered and other conditions. Of course, those assumptions cannot be met in all contexts.

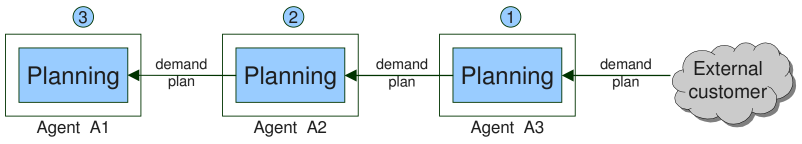

Fig. 3. *Upstream planning*.

One variant is to apply two phases of planning: one upstream and the other downstream [10] (Fig. 4). This way, we have a hierarchy of subproblems, rather than a hierarchy of agents. The agent first makes a temporary plan to compute its needs in raw materials and sends this information to its supplier. The supplier will try to satisfy this demand and responds with a supply plan, but it is not mandatory for that plan to satisfy all demand. For example, some deliveries may be planned to be late or some products can be replaced by substitutes. When informed of the supply granted by its supplier, the initial agent has to revise its production plan. When applied to the whole supply chain, the task flow forms a loop.

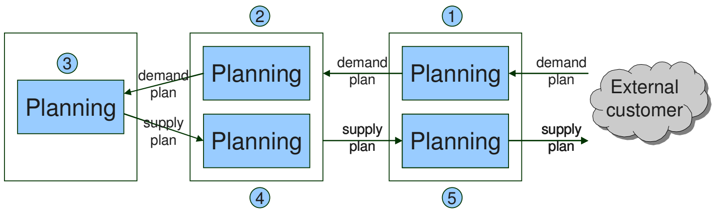

Fig. 4. *Two-phase planning* example

There are some reasons why those two coordination heuristics are so widely used by industry. First of all, the exchanged data are standard (order bills, supply plans). They correspond to standard ways of doing business so it is rational for businesses to agree on the associated *logistic framework*. Privacy and autonomy are also respected (each agent plans its own activities). Finally, these methods are well supported by current decision support technologies: each agent still continues to use specialized planning tools to plan its activities.

3 Coordination space as a Tree

In this section we show how to generalize the hierarchical approach described in the previous section. This is possible as the algorithms used to make each local decision usually allow producing alternative solutions [13]. Each of these is considered as an alternative proposition to the next agent. By considering each of these propositions we can describe the coordination space as a tree (Fig. 5). The tree has one level per type of subproblems. For example, if we want to generalize the *Two-phase planning* approach, each level will correspond to one of the boxes in Fig. 4. Each node on a specific level represents an instance of that subproblem type (defined by decisions for previous subproblems). Each arc is an alternative and feasible solution to the subproblem. The number and the order of the arcs depend on the local solver used by the local algorithm used by the agent.

Formally, the global problem is defined by a vector of subproblems: $X=[X_1,\ldots,X_M]$. Each subproblem X_i is under the responsibility of an agent $\mathcal{A}(X_i)\in A=\{A_1,\ldots,A_N\}$. For each subproblem X_i the agent $\mathcal{A}(X_i)$ has a solver $\mathcal{S}^i$ producing a vector S^i of alternative solutions: $\mathcal{S}^i(X_i)\rightarrow S^i$ where $S^i=\left[S^i_1,\ldots,S^i_{|S^i|}\right]$. These solutions are not known *a priori*; they are revealed one after another by the solver. Eventually one gets selected. We will denote it by S^i_*. The agents are looking for the vector $\left[S^1_*,\ldots,S^M_*\right]$ minimizing an objective function

$\mathcal{F}\left(\left[\mathtt{S}_*^1,\ldots,\mathtt{S}_*^M\right]\right)$. Each subproblem $\mathtt{X}_i$ is defined over the chosen solutions for the previous subproblems: $\mathtt{X}_i = \mathcal{G}^i\left(\left[\mathtt{S}_*^j \middle| 1 \le j < i\right]\right)$.

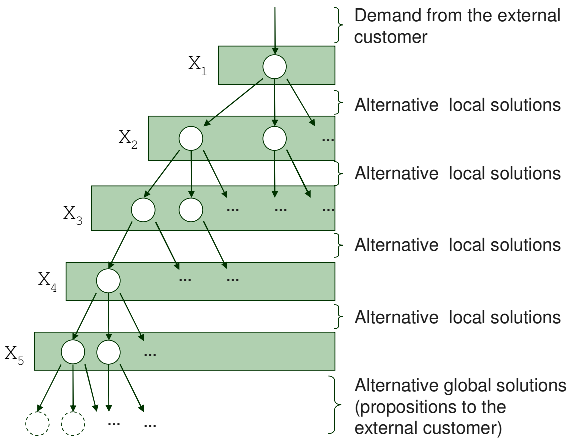
Demand from the external customer
X1
Alternative local solutions
X2
Alternative local solutions
X3
Alternative local solutions
X4
Alternative local solutions
X5
Alternative global solutions (propositions to the external customer)

Fig. 5. Tree for generalized *Two-phase planning*.

The simplest method for the agents to collectively explore this tree is to perform what Hirayama and Yokoo terms *Synchronous Branch and Bound* (SyncBB) [3]. In SyncBB, agents solve their subproblems in sequence: the first one solves its subproblem and sends its decision to the second agent, and so on. In the case of a dead-end, or when a leaf is reached, chronological backtracking occurs: the agent that is in control sends back a message to the previous agent that then has to come up with an alternative proposition. This message contains the value of the best solution found so far. It will be used during the rest of the search to prune the tree. Of course, the method is complete.

SyncBB has two main drawbacks. First, only one agent at a time is working. Second, it applies chronological backtracking. In a centralized context chronological backtracking is often outperformed by other backtracking strategies, for example, those based on the computation of discrepancies [4,14] (see section 4.1). In the next section, we propose a method that can be used to perform a distributed search while applying backtracking based on discrepancies and allowing the agents to work concurrently.

4 Multi-agent Concurrent Discrepancy Search (MacDS)

This section introduces a new method (MacDS) that performs distributed search with backtracking based on discrepancies. It is complete, but aims at producing good solutions in a short amount of time. It allows concurrent computation, uses asynchronous communication, an asynchronous timing model [15] and is tolerant to random message transmission delays. It also takes advantage of situations where subproblem solving times vary from one agent to another.

4.1 Discrepancy-based Methods

Limited Discrepancy Search (LDS) was the first method based on discrepancies. It was developed for centralized problems and was introduced by Harvey and Ginsberg in [4]. The basic premise is as follows. The solutions in the search space (leaves of the tree) do not all have the same expected quality. That is, solution quality decreases with the number of times one branches to the right when going from the root to that leaf (i.e. the number of discrepancies). The rationale is that a move 'to the right' is a move against the value ordering heuristic used at each node. Fig. 6 shows a simple tree and the number of discrepancies associated to each leaf.

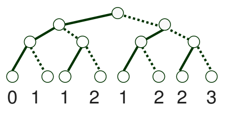

Fig. 6. Binary tree (two arcs per node) with the associated number of discrepancies for each leaf. It corresponds to the number of time one branches to the right when going from the root to that leaf (dotted arcs on the figure).

LDS aims to first visit first the leaves with the fewest discrepancies (that is, solutions with greatest expected quality). Another effect of LDS is that the solutions visited in a given period of time will be more varied from one another than those produced using chronological backtracking. It is this interesting characteristic we seek to exploit in our distributed algorithm.

LDS was introduced as a search procedure, but the concept can be used to specify a node selector to then use with a generic search engine. When backtracking conditions occurs, the search engine must select the node for which the next unvisited child has the fewest discrepancies [16]. This is why LDS can also be seen as a backtracking policy.

LDS has been applied with success to optimization problems in [14]. For n-ary trees, the authors proposed to count the discrepancies as follows: the `i`-th arc followed at a given level counts as `i-1` discrepancies. Over time, other discrepancy-based methods have been proposed [16,17]. Discrepancy search has been integrated into commercial solvers such as, for example, ILOG Solver.

4.2 Proposed Algorithm

We will first describe the algorithm informally using a simple example (Fig. 7). As for SyncBB, the first leaf is reached by solving the sequence of subproblems (subfigure i). The first main difference with the aforementioned algorithms is the following: as soon an agent has sent a proposal to the next agent, it starts working on an alternative proposal and sends it to the next agent. In subfigure i, we already have our first global solution and each agent is working on an alternative local decision. Eventually, one of the agents will come up with a new proposal. Let us suppose its agent C (subfigure ii): we then have our second global solution. Agent C is then working on a third proposal. Then, suppose agent B is the next to come up with a new proposal (subfigure iii): agent C must then decide whether it is better to produce a third solution based on the node it is currently working on (to the left), or produce a first solution based on the new node (to the right). This decision must be based on discrepancies. The following explains how.

With each message sent, the agent attaches the path that would go from the root to the corresponding node in the global tree (in the figure, those paths are identified onto the arcs). So, although the agent has a local vision, it can 'imagine' the path in the global tree leading to each of its nodes. An agent can then make its decisions about the best node to work on based on discrepancies. In our example (subfigure iii), agent C must choose between producing a new solution with a total of 2 discrepancies (0+0+2), or a solution with 1 discrepancy (0+1+0). Because we want to apply an LDS policy (smaller number of discrepancies first), the agent will choose the latter (subfigure iv). In subfigure v, agent B has received a new message from agent A and has switched to that node.

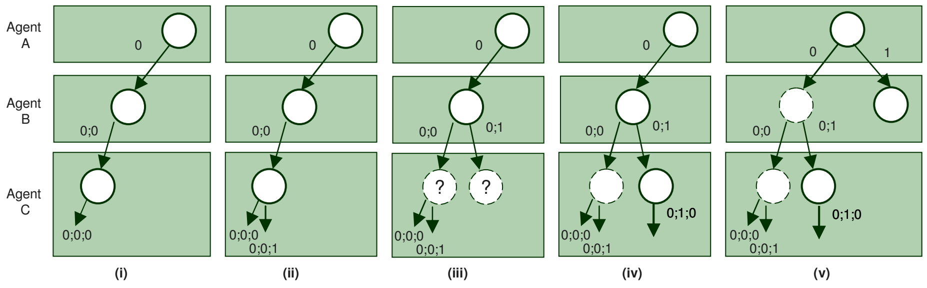

Fig. 7. Part of a MacDS execution trace.

To summarize, each agent manages a list of nodes (alternative solutions from previous agent). With each solution sent, agents attach the 'path' that would go from the root of the global tree to that specific node. At all times, each agent works on the node/subproblem from its own list with the highest priority. The priority of a node is a function of its path and the number of alternative solutions already sent for that subproblem (that is, the number of discrepancies of the next local solutions).

This approach is similar to implementing a backtracking strategy in centralized search using a node selector, except that it is applied locally by each agent. By changing the selector function, it is possible to implement different known strategies, such as LDS, *Depth-bounded Discrepancy Search* (DDS) [17], or even chronological backtracking (BT).

In the extreme case where a single agent owns every subproblem, then MacDS visits the nodes of the tree in the same order as a centralized search algorithm (applying the same backtracking strategy) would. In a distributed context where each agent manages its own task list, each solution to the global problem will be obtained in no more time than would be necessary for the centralized algorithm (ignoring communication delays). Each agent begins work as soon as there is a task in its list, but preempts its current work when a task with greater priority is added to the list.

As in SyncBB, the value of the best solution found to date is sent to the previous agents (although it is not explicit in the following pseudocode). The algorithm is complete since it explores the same solution space as SyncBB. In the case of communication breakdown (or in presence of random message transmission delays), the agent always works on the available node with the highest priority rather than remaining idle. In this way, it is not mandatory for messages to arrive in the same order as they were sent.

4.2.1 Pseudocode

The following objects are manipulated by the algorithm:

- A message `msg` is a pair `<d,p>` where `d` represents the decisions for the previous subproblems and `p` is a vector of integers representing the path to the corresponding node in the global tree. The element `p[j]` defines, for a level `j`, which arc should be followed when going from the root to that node.
- A list of nodes (`nodes`) contains the nodes under the responsibility of the agent for which there is unexplored alternative solutions. A node is defined by `d` and `p`, by the number of local solutions produced to date (`i`) and by a boolean indicating if the agent thinks there are no more alternative solution (`noMoreSol`).

Each agent runs many threads: one for each node plus a control thread. A single thread per agent is active at any moment. The control thread (Fig. 8) is activated when the agent receives a message (`WhenReceiveMsg`) and when the agent produces a new solution for a node (`WhenNewSolution`). The agent then updates the node list and transfers control to the thread of the node with higher priority (`ActivateANode`).

```
WhenReceiveMsg(msg)
  if (running ≠ ∅) running.Sleep();
  nodes.insert(<msg.d, msg.p, 0, false>));
  ActivateANode();

WhenNewSolution(node)
  node.Sleep();
  SendMessage(Successor(node), <node.d + node.sol.d, node.p + node.i>);
  node.i++;
  if (node.noMoreSol)
       nodes.remove(node);
       running ← ∅;
  ActivateANode();

ActivateANode()
  if (nodes.count() > 0)
       running ← nodes[1];
       running.WakeUp();
  else
       running ← ∅;
```

Fig. 8. Control thread of the agent.

The pseudocode for the node threads is shown in Fig. 9. When a node is created, its thread is idle. It must be activated by the control thread. When the node thread produces a new subproblem solution, it signals this to the control thread (`SignalNewSolution`) and goes idle (`sleep`). The control thread then sends the message to the agent that owns the next subproblem.

In the illustrated pseudocode, we suppose that nodes are ordered by decreasing priority. Fig. 10 shows examples of comparator functions that can be used to keep the list sorted. They identify, over a pair of nodes, which one should be given the highest priority according to a given backtracking strategy. The first example (`CompareBT`) defines a chronological backtracking policy. The other one (`CompareLDS`) defines an LDS policy. The arguments of the functions are the 'path' in the global tree of the next local solution each node would generate (thus, the concatenation of `node.p` and `node.i`).

```
Run(node)
  node.noMoreSol ← false;
  node.sol ← NextSolution(node);
  while (node.sol ≠ ∅)
        SignalNewSolution(node);
        Sleep();
        node.sol ← NextSolution(node);
  node.noMoreSol ← true;
  SignalNewSolution(node);
```

Fig. 9. Thread for a node.

```
CompareBT(p1, p2)
  depth ← Min(Card(p1), Card(p2));
  j ← 1;
  while (p1[j] = p2[j] && j ≤ depth) j++;
  if (j ≤ depth)
        if (p1[j] ≤ p2[j]) return p1; else return p2;
  else
  if (Card(p1) ≥ Card(p2)) return p1; else return p2;

CompareLDS(p1, p2)
  t1 ← Σ(j=1..Card(p1)) p1[j];
  t2 ← Σ(j=1..Card(p2)) p2[j];
  if (t1 < t2) return p1;
  else
        if (t2 < t1) return p2; else return CompareBT(p1,p2);
```

Fig. 10. Two different backtracking policies.

5 Evaluation

MacDS was evaluated with generated data and with real industrial problems. We compared MacDS applying an LDS policy (MacDS_LDS) with *Synchronous Branch and Bound* (SyncBB) which apply chronological backtracking (BT). To measure which part of the gain was due to concurrency and which to the backtracking policy, we also tested a version of MacDS applying chronological backtracking (MacDS_BT). We also implemented SyncLDS (synchronous as SyncBB, but applying LDS).

5.1 Industrial Evaluation

We first evaluated the algorithms using real industrial data with complex subproblems. The case is a supply chain coordination problem in the forest products industry [10]. The supply chain has three facilities (Sawing, Drying and Finishing). Each local problem solver was developed by FORAC, a consortium of companies and researchers. The sawing agent plans its operations using a Mixed Integer Linear Programming model. The drying agent uses Constraint Programming. The finishing agent uses a forward-scheduling heuristic, but uses *Depth-bounded Discrepancy Search* (DDS) [17] to produce alternative solutions. *Two-phase planning* (see section 2.1) is used as basic/reference coordination mechanism. The data were extracted from the company databases at different moments in 2005.

The experiments were done in a distributed environment where each agent runs on a different computer. It allows measuring the real impact of concurrent computation in a situation where subproblems are complex and take a different amount of time to be solved (the difference in computation time between subproblems can vary five fold). The production of solutions for local subproblems varies from a few seconds to several minutes. In this context, few messages are sent, so communication complexity is not an issue [15].

For a given case, the first global solution of each compared algorithm is always the same (it corresponds to the first leaf of the tree). It is also this solution that would be obtained by the companies using the current business process (in this case, *Two-phase planning*). Consequently, we compared the algorithms according to the reduction of the objective function they achieved with additional computation time (in seconds).

Fig. 11 illustrates the results of the four industrial cases studied (i, ii, iii, iv). For a very short computation time, SyncBB and MacDS_LDS give comparable results. This is because they produce the same solutions until the last agent receives a second task in its list. Then, MacDS_LDS starts to outperform SyncBB in a significant manner. While SyncBB persists in exploring only minor variations of the first solutions, MacDS_LDS explores different areas of the search tree. Case (ii) is an exception: SyncBB provides the best solution by 0.5% given a computation time greater than 1000 sec.

We can see the impact of the backtracking policy solely by comparing SyncBB and SyncLDS. The latter outperforms SyncBB, except for very short computation times.

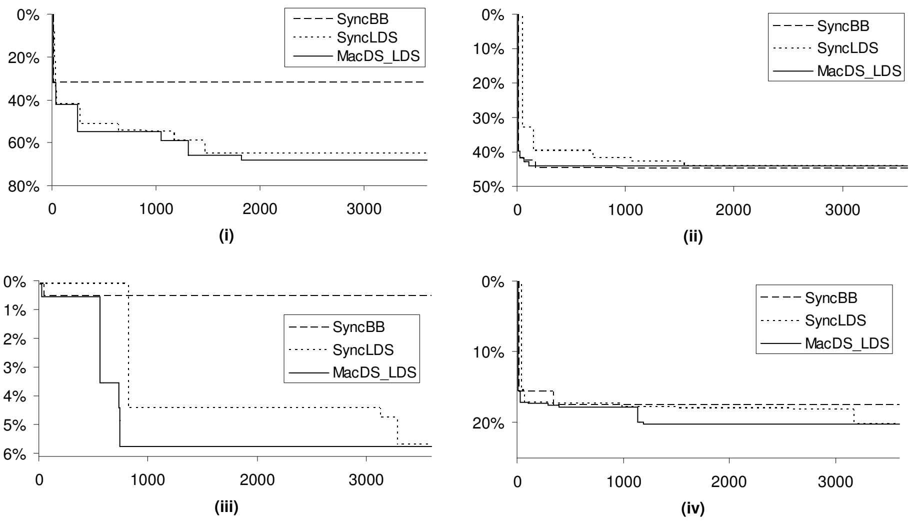

Fig. 11. Reduction of the objective function, according to computation time (in seconds) for cases (i), (ii), (iii) and (iv).

As for the impact of concurrency, we compared MacDS_LDS with SyncLDS. For any solution quality reached by SyncLDS, MacDS_LDS produces an equal or better solution in an equal or shorter computation time. The average reduction of computation time for each case is as follows: **18.5%**, **88.7%**, **54.5%** and **64.0%**. Two reasons explain this. Concurrency makes each solution being produced in an equal or shorter amount of time, but it also gives agents the opportunity to explore more alternative solutions in a given amount of time.

MacDS_BT gave results indistinct from those of SyncBB. For this reason, these results are not shown in the figures. The explanation is the following: for industrial cases, any subproblem has many alternative solutions. When performing chronological backtracking, the system takes a long time to explore alternative solutions of the last subproblem only. In MacDS_BT, previous agents produce alternative solutions but they are never exploited by the last agent in the given computation time.

5.2 Evaluation with Generated Data

It is known that the performance of LDS for a given problem depends on how well the search tree respects the basic premise (i.e. good correlation between the quality of the leaf and the number of discrepancies). Consequently, we evaluated MacDS using a wide range of generated problems; some completely random and others having good correlation. We generated n-ary trees such that the probability that a leaf is the optimal solution is proportional with $\delta^{(p)}$, where p is the number of discrepancies of the leaf and δ is a parameter that varies from 0.01 to 1.00. When δ is maximal, all leaves have the same probability (the tree is completely random). The smaller the δ, the better the correlation between solution quality and discrepancies.

Others parameters we experimented with were: number of subproblems (Card(X)), message transmission delay (τ), subproblem solving time (α), and the number of solutions for each subproblem (n). The performance measure used is the expected time needed to find the best solution. Fig. 12 illustrates the performance of the algorithms according to the value of parameter δ. MacDS_LDS is always the best of the four algorithms tested (equalled by MacDS_BT when δ=1.0). For both backtracking policies (BT and LDS), the MacDS version always beats the synchronous one.

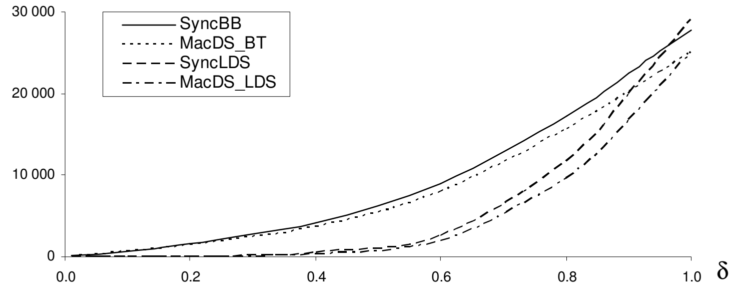

Fig. 12. Expected time to get best solution (in sec.) according to δ [Card(X)=4;n=10;α=1; τ=0].

Fig. 13 illustrates the impact of the other parameters (all results shown for δ=0.7). Subfigure i shows that message delay (τ) has a linear impact for all four algorithms, but a much smaller one for both versions of MacDS. For them, expected time needed to find the best solution is equal to `(Card(X)-1)`τ. Subfigure ii illustrates that subproblem solving time (α) also has a linear impact, again smaller for the MacDS versions. Subfigure iii shows that the impact of the number of subproblems (`Card(X)`) is exponential.

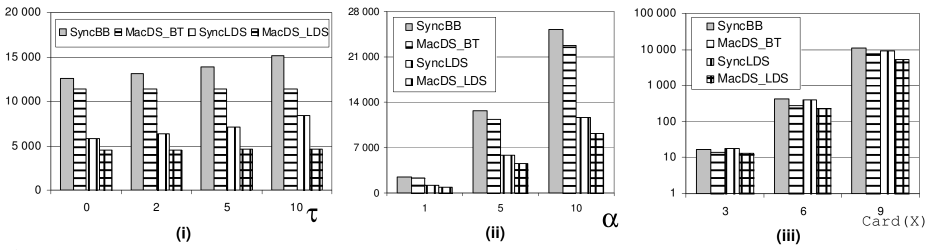

Fig. 13. Expected time to get best solution, according to: (i) message transmission time [Card(X)=4;n=10;α=1;δ=0.7], (ii) time to solve subproblem [Card(X)=4;n=10;τ=0;δ=0.7] and (iii) total number of subproblems [n=3;α=1;τ=0;δ=0.7].

6 Related Work in DCOP

In section 2 we presented a literature review regarding the coordination of manufacturing operations (SCOP) in the Supply Chain Management community. In this section we will compare our current work with the more generic framework of *Distributed Constraint Optimization Problems* (DCOP) [18].

DCOP algorithms (and those for *Distributed Constraint Satisfaction Problems*, DisCSP) can be classified between two families: *single search process algorithms* (SPA) and *multiple search process algorithms* (MPA) [19].

The mechanism used by MacDS makes it an MPA: each global solution is constructed sequentially by the agents, but they work simultaneously on many solutions (it is up to the MPA algorithm to specify how and when a new ‘path’ must be explored). Other MPA algorithms have been proposed, but for DisCSP solving. Our own approach to creating new search processes (by allowing the agents to sent unsolicited messages to the next agents) can be described as naïve: it makes memory complexity exponential. However, this did not pose a problem in our SCOP applications as local subproblems were very long to solve. Therefore our bottleneck is time rather than memory. Yet, it would be easy to allow each agent to implement a bound on memory. Each agent would just refuse incoming messages when this bound is met, forcing the sender agent to ‘slow down’ message transmission.

With algorithms of the SPA family, each agent keeps only one local solution at a time. This family includes synchronous algorithms (like SyncBB) but also others that allow concurrent computation. The latter are said to be ‘asynchronous’: agents solve their subproblems simultaneously, knowing that local solutions are constrained by shared constraints. The most well-known algorithm for DCOP is ADOPT [18].

Until now, we compared MacDS to SyncBB but not to asynchronous SPA. The main reason is some difficulties in mapping the actual business context of SCOP into the DCOP framework. We next introduce some of the main differences between DCOP and our specific SCOP context:

- DCOP focus on providing a single algorithm that will be executed by each agent. In SCOP we want to propose a mechanism aiming to coordinate agents having selected *a priori* a specialized (and private) algorithm/criteria to solve its local subproblem (and propose alternative solutions). This is different from DCOP methods like ADOPT that specify the criteria to be used by the agent to choose its value.
- In DCOP, the different subproblems are defined *a priori*. Agents then have to solve them knowing there are some constraints between them. In comparison, in our model presented in section 3, a subproblem is defined by the decisions for previous subproblems. In SCOP, agents know how to solve a specific instance of the subproblem (e.g. plan its production according to its demand) but not 'the' generic subproblem (e.g. plan its production while having no idea of the manufacturing processes of the others agents and with no insight into what is needed). Those limitations come from the business context, as reflected by the *logistic framework* (see introduction and section 2).
- In DCOP, the domains of the variables are defined *a priori*. It is often supposed that it takes negligible time for the agent to choose the available value minimizing a specific function. In our SCOP context, each solution must be found by the local solver… and it may takes a long time just to find one local solution, as subproblems are complex. For most of the subproblems we experimented with in section 5.1, it was not possible to find the optimal local solution in the given computation time.

Although the use of the DCOP framework seems difficult in our SCOP context, this does not imply that it is impossible. Moreover, other problems in SCM can benefit from the use of the DCOP framework and algorithms.

7 Conclusion

There are many levels and aspects to the contributions made in this article. From the supply chain management point of view, we showed how hierarchical approaches can be generalized to consider the entire coordination space, which can be represented as a tree. Using distributed search allows for the exploration of alternative solutions while maintaining current business relationships, responsibilities and local decision-making algorithms. We also showed that even simple algorithms like SyncBB provide for great improvements in solution quality in comparison with current practice.

As for distributed optimization, we demonstrated how to use discrepancy-based methods in a distributed and concurrent context. It improved computation time and quality for both real problems and generated trees.

References

1. Schneeweiss, C., Zimmer, K.: Hierarchical coordination mechanisms within the supply chain. European Journal of Operational Research. 153, 687-703 (2004)
2. de Kok, A.G., Fransoo, J.C.: Planning Supply Chain Operations: Definitions and Comparison of Planning Concepts. In: de Kok, A.G., Graves, S.C. (eds). Supply Chain Management: Design, Coordination and Operation. Elsevier, Amsterdam (2003)
3. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. International Conference on Principles and Practice of Constraint Programming, LNCS #1330, pp. 222-236 (1997)
4. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. International Joint Conference on Artificial Intelligence, pp. 607-613. Montreal (1995)
5. Fink, A.: Supply Chain Coordination by Means of Automated Negotiations Between Autonomous Agents. In: Chaib-draa, B., Müller, J.P. (eds). Multiagent-Based Supply Chain Management. Springer, New York (2006)
6. Cachon, G.P.: Supply Chain Coordination with Contracts. In: de Kok, A.G., Graves, S.G. (eds). Supply Chain Management: Design, Coordination and Operation. Elsevier, Amsterdam (2003)
7. Porteus, E.L.: Responsibility tokens in supply chain management. Manufacturing and Service Operations Management. 2, 203-219 (2000)
8. Dudek, G., Stadtler, H.: Negotiation-based collaborative planning between supply chains partners. European Journal of Operational Research. 163, 668-687 (2005)
9. Shen, W., Hao, Q., Yoon, H.J., Norrie, D.H.: Applications of agent-based systems in intelligent manufacturing: An updated review. Advanced Engineering Informatics. 20, 415-31 (2006)
10. Frayret, J.M., D'Amours, S., Rousseau, A., Harvey, S., Gaudreault, J., CENTOR, Université Laval, Québec, DT-2005-JMF-1 (2005)
11. Schneeweiss, C.: Distributed Decision Making, Springer, New York (2003)
12. Bhatnagar, R., Chandra, P., Goyal, S.K.: Models for multi-plant coordination. European Journal of Operational Research. 67, 141-60 (1993)
13. Kilger, C., Reuter, B.: Collaborative Planning. In: Stadtler, H., Kilger, C. (eds). Supply Chain Management and Advanced Planning. Springer, New York (2005)
14. Le Pape, C., Baptiste, P.: Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. Journal of Heuristics. 5, 305-325 (1999)
15. Lynch, N.A.: Distributed algorithms, Morgan Kaufmann, San Francisco (1996)
16. Beck, J.C., Perron, L.: Discrepancy-Bounded Depth First Search. Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems, pp. 7-17. Paderborn, Germany (2000)
17. Walsh, T.: Depth-bounded discrepancy search. International Joint Conference on Artificial Intelligence, pp. 1388-1393 (1997)
18. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence. 161, 149-180 (2005)
19. Zivan, R., Meisels, A., Message delay and DisCSP search algorithms. Annals of Mathematics and Artificial Intelligence. 46, pp. 415-39 (2006)