Supply Chain Coordination Using an Adaptive Distributed Search Strategy

Jonathan Gaudreault[1*], Gilles Pesant[2], Jean-Marc Frayret[1,2], Sophie D'Amours[1]

[1] FORAC Research Consortium, Université Laval, Québec, Canada

[2] École Polytechnique de Montréal, Montréal, Canada

[*] Corresponding author (*jonathan.gaudreault@forac.ulaval.ca*)

ABSTRACT

A tree search strategy is said to be *adaptive* when it dynamically identifies which areas of the tree are likely to contain good solutions, using information gathered during the search process. This work shows how an adaptive approach can be used to enhance the efficiency of the coordination process of an industrial supply chain. The result is a new adaptive method (called *Adaptive Discrepancy Search*), intended for search in non-binary trees, and that is exploitable in a distributed optimization context. For the industrial case studied (a supply chain in the forest products industry), this allowed reducing nearly half the time needed to obtain the best solution in comparison with a standard non-adaptive method. The method has also been evaluated for use with synthesized problems in order to validate the results obtained and to illustrate different properties of the algorithm.

KEYWORDS

Supply chain coordination, Adaptive search, Multi-agent, Distributed search, Discrepancy.

I. INTRODUCTION

Many combinatorial optimization problems can be formulated as tree search problems. This is the case for an important coordination problem found in industrial supply chains. Within this context, each manufacturing unit (or agent) wishes to plan/schedule its own manufacturing operations. However the agents must coordinate in order to produce a global operations plan permitting them to best meet the needs of the external customers. Because the solution space can be represented as a tree, the coordination process between the agents can be seen as a distributed search in that tree [1]. However, the tree is very large and the *search strategy* employed by the agents' collective will have a considerable impact on their capacity to find good solutions rapidly (i.e., solutions best satisfying the demands of external customers).

In this article, we pose the hypothesis that certain zones of the tree are more apt to contain good solutions, and that it is possible for agents to locate them collectively and dynamically (i.e. during search), with the aim of focusing on them in priority. Search strategies presenting these characteristics are called *adaptive* [2] or *reactive* [3]. They exploit a form of *learning* where certain characteristics of the part of the tree already visited serve to feed a statistical model that guides the remainder of the search. Most adaptive approaches proposed in the literature are intended for search in **binary trees** for solving **constraint satisfaction problems**. They are destined to be used in a **centralized** environment.

In this work, we show how these notions can be adapted to coordination in supply chains. The result is an adaptive method (called *Adaptive Discrepancy Search*, or ADS) designed for combinatorial **optimization problems** represented by a **non-binary tree**, and capable of being exploited in a **distributed** context. The proposed approach has allowed a considerable reduction in the time necessary for coordination in an industrial supply chain, in comparison with standard non-adaptive methods.

The remainder of this paper is organized as follows. Sections II, II and IV present some background and review the literature regarding supply chain coordination and adaptive search methods. Then, the proposed adaptive method (ADS) is introduced in Section V. Experimental results are provided in Section VI, for both a real industrial supply chain problem and synthesized problems. Finally, Section VI concludes the paper.

II. The Supply Chain Coordination Problem (SCCP)

To fulfill their mission, today's companies must collaborate with a number of partners. These business units possess varied and complementary expertise. They form together a value creation network constituted of loosely coupled but interdependent business units called the s*upply chain* (SC) [4]. Figure 1 presents a fictitious example of a supply chain (adapted from [5]).

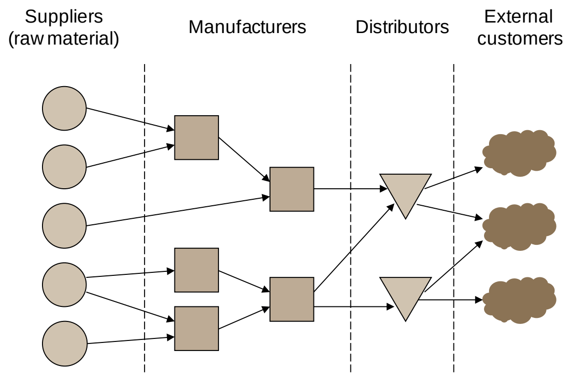


Figure 1. Example of a supply chain. Adapted from [5].
Arcs represent product flows between business units.

Supply Chain Management (SCM) deals with the coordination of manufacturing and logistics activities between these autonomous (or semi-autonomous) decision making units. Partners in the supply chain seek competitive advantages, like the ability to meet the changing and diverse needs of customers within short lead times [6]. SCM encompasses a wide range of problems, including design of production and distribution networks [7, 8], management of inventory [9], design of contracts between buyers and producers [10] and development of advanced supply chain planning systems [11].

These problems are attracting the attention of researchers from many different backgrounds, from economics to operations research and industrial engineering. More recently, they have interested people from the artificial intelligence community, notably because of the fact that the concept of multi-agent system is a natural metaphor to describe a supply chain [5, 12]. Since 2003, the *Trading Agent Competition Supply Chain Management Game* (SCMG) [13] has been held, which puts researchers in contact with certain aspects of supply chain management. While the SCMG competition involves **antagonist** agents competing for customer orders and supply [14], the focus of this paper is on coordination in **collaborative** supply chains. This constitutes another important economic issue, as most executives agree that future competition will more and more occur between supply chains, instead of between isolated companies [15].

This research focuses on the specific problem of synchronizing manufacturing operations in a collaborative supply chain over a short-term horizon. This can be referred to as *collaborative production planning* [16] or the *supply chain coordination problem* [17]. Figure 2 provides an illustrative example. In this example, supply chain partners are manufacturing agents that must coordinate their manufacturing operations in order to deliver what was ordered by an external customer with the least possible delay (they could also want to optimize other criteria; we provide this one only as an example). However, different alternatives are possible regarding the parts to use, the manufacturing processes to follow, the scheduling of operations and the choice of transportation.

From a global perspective, this can be seen as a classic planning and scheduling problem (decide *what to do*, *where*, and *when*). However, this situation is better described as a *coordination problem*: each agent wants to plan and schedule its own operations (that is, produce a *local plan* by itself), but there are dependencies between these local plans (i.e. product flows between agents they must agree on). In an industrial setting, each agent uses specialized software to plan its operations (the planning problems of each factory generally differ from one to another). These tools are called *Advanced Planning and Scheduling Systems* (APS) [18] and they can take a very different form from one factory to another. However, each agent is generally able to propose several alternative local plans in order to achieve the common objective. The different valid combinations of these alternative plans define the *coordination space* that is accessible to the partners. In industrial practice, the collective does not have an infinite amount of time to search this coordination space and make a proposition to the external customer (indeed, producing the local plans takes time and the consortium can be in competition with other supply chains). The consortium will then quickly seek for a solution, and if time remains will try to find a better quality solution. This is called an *anytime algorithm* [19, 20].

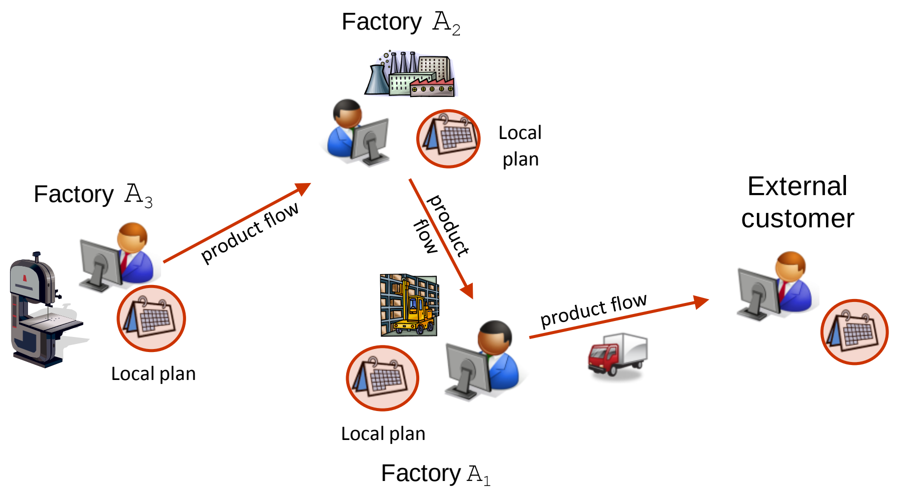


Figure 2. Local plan and product flow for a simple supply chain.

The literature on Supply Chain Management (SCM) is plentiful, but more limited regarding the coordination of the manufacturing operations on a day-to-day basis [17]. There is a gap to be filled between problem solving at the agent level and coordination strategy at the supply chain level [21]. Indeed, at one end of the spectrum, many researchers propose specialized algorithms for the agents to make their local decisions (e.g. scheduling) with no consideration for the other aspects of the global problem. These algorithms have generally made their way into industrial applications, being integrated into software packages, but support for integration of local models is rather limited [22].

At the other end of the spectrum, many researchers study the key characteristics of good business relationships. For example, they study which incentive policies (such as penalty costs) can be used to direct the behavior of the cooperating units (see [10]). Other researchers study the impact of improved information sharing on the performance of the agents. For example, Schneeweiss studied in [23] the performance of the supply chain as a function of the accuracy of the information accessible to agents, and Moyaux *et al* [12] studied the reduction of inventory fluctuations achievable when sharing demand information between partners.

Concerning the coordination mechanism itself, recent literature in SCM proposes other forms of advanced coordination frameworks involving negotiation and information exchange schemes aiming at synchronizing planning decisions through the supply chain (e.g. [17, 24]). Generally, these approaches are applied in contexts where partners have different objectives/interests but wish to agree on a plan. They are usually defined for two-partner relationships or the method involves a mediator.

Finally, in SCM literature, we see an increasing use of multi-agent technologies in order to create advanced systems for enterprises collaboration [25-27]. This community particularly emphasizes the design of interaction protocols for agent coordination (see [28, 29] for a review). However, most of these protocols can be described as *coordination heuristics*. Other authors propose radically different approaches where agents are totally benevolent; agents plan and synchronize at the same time by applying together a distributed algorithm they agreed on in advance (e.g. [30]). This is a different context than the SCPP situation described previously, where each agent uses a specialized algorithm to make its local decision but they need to coordinate.

Another promising path is to allow agents to use *distributed search algorithms* as a coordination mechanism [1, 31, 32], but in a context where the general assignment of responsibilities to agents as well as the structure of the tree (the *coordination space*) are induced by the business environment. Searching this tree allows obtaining good solutions in short computation time, and lead to an optimal solution if enough time is available. The following section reviews how this concept can be used in practice.

III. DISTRIBUTED SEARCH AS COORDINATION MECHANISM

In an industrial context, a certain form of *hierarchy* usually exists in the network. For example, in a customer-supplier relationship, the role of the customer is similar to the one of an order-giver in a military hierarchy. Its decisions determine the leeway available to the followers.

Schneeweiss proposed a general framework in order to model these kinds of *problem hierarchies* in [33]. According to this framework, the decision taken at a given level (top) is interpreted by the following level (bottom) as a parameter of its subproblem (i.e. the subproblem *bottom* is defined as a function of the top decision). Of course, this can be generalized to situations with more than two subproblems. The subproblems can be solved sequentially, from top to bottom, which is termed *upstream planning* [24, 34].

The decomposition of the overall global problem into a sequence of subproblems is determined by the business context. To illustrate, let us see how this concept of hierarchy applies to our supply chain example from Figure 2. One of the agents (A_1) is in contact with the external customer who expresses his need/demand. Knowing the demand from the external customer, this agent makes a temporary plan to compute its needs in raw material and sends this information to its supplier (A_2) as a *demand plan* (a list of products, quantities and expected delivery dates). In turn, the supplier tries to satisfy this demand and responds with a *supply plan* that does not necessarily meet all demands (e.g. 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 (A_1) has to revise its production plan in order to account for supply constraints. When this protocol is extended to the whole supply chain, the succession of planning activities forms a loop with two phases: one upstream, where demand is tentatively propagated, and the other downstream, where final supply is propagated (see Figure 3). This is called *Two-phase planning* in [28].

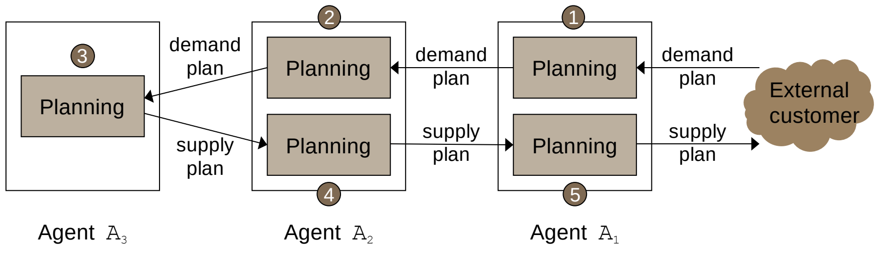


Figure 3. Two-phase planning protocol as coordination mechanism for the previous supply chain.

This illustrates well how business is done in practice; partners have well defined local planning/scheduling domains, and synchronize by exchanging business artifacts (e.g. demand plan, supply plan) following well-established protocols. Each specific situation can be modeled using a workflow diagram similar to the one in Figure 3 that states a sequence (or hierarchy) of subproblems.

A. Modeling the SCCP as a Hierarchical Distributed Constraint Optimization Problem (HDCOP)

Given the hierarchy (or sequence) of subproblems, and knowing that each agent is able to produce alternative solutions (local plans) for its local subproblems, the whole situation can be modeled as a *Hierarchical Distributed Constraint Optimization Problem* (HDCOP) [31], and the coordination space can be represented as a non-binary tree of fixed depth.

Definition 1: *Hierarchical Distributed Constraint Optimization Problem* (HDCOP) [31]. A global problem is defined by a vector of subproblems: $X=[X_1,...,X_M]$. Each subproblem X_i comes under the responsibility of an agent $\mathcal{A}(X_i)\in A=\{A_1,...,A_N\}$. For each subproblem X_i the agent $\mathcal{A}(X_i)$ has access to a solver $\mathcal{S}^i$ producing a vector S^i of alternative solutions: $\mathcal{S}^i(X_i)\to S^i$ where $S^i=\left[S^i_1,...,S^i_{|S^i|}\right]$. These local solutions (in our case, local plans) are not known *a priori*; they are revealed one after another by the solver (in our case the APS tools used by the industry). Eventually one gets selected. We will denote it by S^i_*. Agents look for the vector $\left[S^1_*,...,S^M_*\right]$ minimizing an objective function $\mathcal{F}\left(\left[S^1_*,...,S^M_*\right]\right)$. Each subproblem X_i is defined by the chosen solutions for the previous subproblems: $X_i=\mathcal{G}^i\left(\left[S^j_*\middle|1\le j<i\right]\right)$.

For example, for the *two-phase planning* protocol previously described in Figure 3, the coordination space can be modeled as the following tree (Figure 4). Each node on a specific level represents an instance of the corresponding subproblem (defined by decisions for previous subproblems). Each arc is an alternative and feasible solution to the subproblem. The number and order of these arcs depend on the local algorithm used by the agent. To each leaf of the tree corresponds a solution to the global problem.

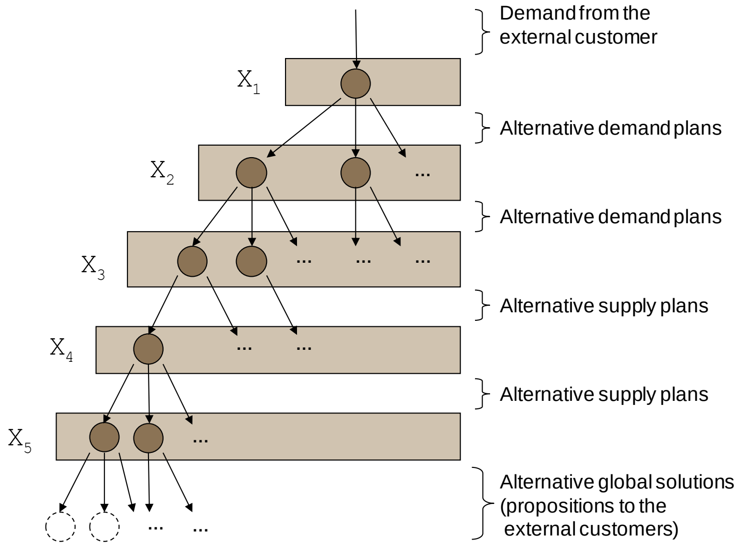


Figure 4. Coordination space for Two-phase planning within HDCOP.

Thus, this tree models the *coordination space* available to agents for a given business context/protocol and for a given set of local algorithms used by the agents. Of course, using different local algorithms or business protocols would lead to a different tree/coordination space.

This reformulation of the coordination problem as a tree calls for the use of a distributed search algorithm as a coordination mechanism. However, agents must perform search in a tree that is not fully defined *a priori*. That is, (1) the alternative solutions (arcs) of the local subproblems are not known before being produced by the local solver, and (2) the specific instances of each subproblem (node) are not known either, because they depend on the solution obtained for previous subproblems. The tree will therefore be 'revealed' progressively during the search process. The next section presents basic algorithms that can be used to search this HDCOP space.

B. Basic algorithms

The simplest but complete method for the agents to collectively explore the coordination space is for them to perform a distributed *chronological backtracking*. This can be done by applying what Yokoo calls *Synchronous Branch and Bound* (SyncBB) [35]. 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 an agent has considered all of its local solutions, this agent sends a message back to the previous agent, asking for an alternative proposition. This backtracking message contains the value of the best solution found so far. It is used during search to prune the tree.

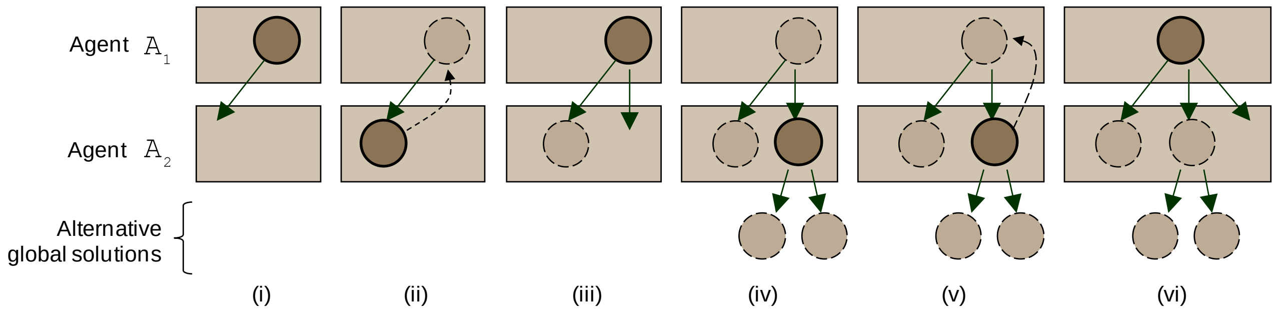


Figure 5. Example of an execution trace for SyncBB.

Because it applies chronological backtracking, SyncBB presents the following drawback. Once the first global solution is found, it tries to enumerate every other alternative solution for the last subproblem without considering other options for the previous subproblems. By doing so, it persists in exploring only minor variations of the first global solution (for industrial cases, a subproblem may have thousands of alternative solutions) in a given period of time.

In contrast, for centralized search, other backtracking strategies allow rapidly visiting different areas of the tree at the beginning of the search. This is the case for methods based on the computation of *discrepancies*, like, for example, *Limited Discrepancy Search* (LDS) [36]. LDS implements the following backtracking strategy. For a binary tree, each node is characterized according to the number of times one should branch to the right when going from the root of the tree to that node (each of them is called a discrepancy, see Figure 6). When backtracking conditions occur (we encounter a global solution and aim for another) we 'backjump' to the node already visited for which the next unvisited child has the fewest discrepancies (for non-binary trees, it was proposed in [37] to count the discrepancies as follows: the *i*-th arc followed at a given level counts as $i-1$ discrepancies). One consequence of doing that is that solutions visited in a given period of time will be from more different parts of the tree than those produced using chronological backtracking.

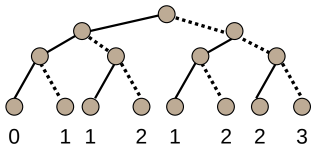


Figure 6. Binary tree (two arcs per node) with the associated number of discrepancies for each leaf. It corresponds to the number of times one branches to the right when going from the root to that leaf (dotted arcs on the figure).

An adaptation of LDS for distributed search was proposed in [31]. The protocol, called SyncLDS, is the following. Agents perform the search in a scheme similar to SyncBB, except that when a global solution is found, the agent detecting that condition sends a message to other agents asking each of them to identify the subproblem/node under its jurisdiction for which the next child has the smallest number of discrepancies. This can be seen as a kind of bid; the agent that detects the need for backtracking gives control to the agent with the smaller bid. The next global solution is reached from that point by solving the remaining sequence of subproblems.

SyncLDS was applied to the Supply Chain Coordination Problem and evaluated for an industrial problem in [32]. It allowed considerable improvement of solution quality and computation time in comparison with SyncBB which proved to be inefficient. Even for huge computation times, SyncBB keeps exploring one region of the tree. In contrast, SyncLDS quickly explores different areas of the tree.

Therefore, the key to good results is through rapidly exploring different areas of the tree. It is thus legitimate to question whether dynamically detecting (during the search) which areas of the tree are most interesting (in order to focus on them first) will lead to a gain in performance. This is what we propose for our distributed problem in Section V. But first, the following section indicates how this idea of *adaptive search* has already been exploited in a centralized context.

IV. ADAPTIVE SEARCH AND OTHER LEARNING TECHNIQUES FOR CENTRALIZED COMBINATORIAL SEARCH

This section reviews some of the learning methods that are used in centralized global combinatorial search. Some will serve as root for the distributed scheme we propose in Section V.

First, let us recall the major difference between our context and the more general one of combinatorial optimization (in addition to the fact that our problem is distributed). In classical combinatorial problems, it is the solving algorithm that constructs the tree (by choosing the order in which variables are instantiated and the order in which values are tried). Learning can then be used in order to establish a strategy for variable ordering and value ordering. In the context of hierarchical decision making, the sequence of sub-problems (equivalent to the sequence of variables) is determined in advance and the sequence of local solutions (equivalent to value ordering) depends on the local solver S^i of each decision maker (see problem definition in Section III.A). In this situation, our room for manoeuvre is limited to the learning of a backtracking strategy. We will however have a look at methods that use learning for a different purpose, as it seems relevant to establish a parallel with those approaches.

Firstly, we must distinguish between two major currents: (1) the *training* (or *offline*) approach, and (2) the *adaptive* approach. The first approach (1) consists in training a system for the resolution of a particular family of problems. In its basic form, the system achieves what a practitioner would do manually, that is, to configure a solver for a particular context [38]. For example, using a set of training problems, the system could determine which variable ordering heuristics and value ordering heuristics are best for a given family of problems. The ACE system [39] uses a similar but more advanced approach. After the training phase, it attributes weights to different heuristics according to their pertinence. Once in production, the system will have the different heuristics to vote for the next variable to instantiate, taking their weight into account. The performance obtained can be better than that of individual heuristics. Other approaches can be used. As an example, the system proposed in [40] studies a set of trees in order to identify the cuts that can be carried out on all these trees while still being assured that good solutions can be found. After training, these cuts are applied to the new problem instances submitted to the system.

The *adaptive* (or reactive) approach (2), concerns the development of systems that dynamically react and adjust during the resolution of a particular instance of a problem [3]. This approach is often put to good use for *constraint satisfaction problems* (CSP). Many authors apply what is called *learning from failure*. When a constraint is violated during the descent of the tree, the conditions of that failure are analyzed with the view of making the most of this knowledge throughout the remainder of the search. For example the techniques of *nogood recording* and *clause learning* seek to avoid redoing combinations of variable/value affectations that are mutually inconsistent. Others try to learn during the search which variables are the most difficult to instantiate, in order to change dynamically the order of variables (e.g. *YIELDS* [41]). In *Impact Based Search* (IBS), the impact of variables is measured by observing how their instantiation reduces the size of the search space [42]. In [43] and [44], each time a constraint causes a failure, the priority of variables implicated in this constraint is increased. In [45], another approach for variable ordering is proposed, but in the context of *Weighted CSP*.

As regards backtracking strategy, approaches where the system learns to evaluate the quality of nodes are of particular interest to us. Ruml has made an interesting proposal regarding this. While a basic LDS strategy gives the same importance to any discrepancy, BLFS [2] dynamically attributes different weights to discrepancies according to their depth. For a binary tree, BLFS will define two parameters for each level of the tree. One corresponds to the "cost" of branching to the left, and the other to the right. The value of a leaf is reckoned to be equal to the sum of the costs along the path from the root to this leaf. By knowing the value of a certain number of leaves, BLFS uses a linear regression in order to establish the value of the parameters. The model was not really used in order to define a backtracking strategy. Instead, the search algorithm proceeds by a series of successive descents in the tree. At each run, it tries to reach a leaf using a path that minimizes the costs. The branching choices are made stochastically in order to avoid always taking the same path. Ruml has achieved very good results with this algorithm (see [46]).

The limits of this approach are the following. The branching factor must be the same for each node on the same level, and one can say nothing about the cost of a supplementary discrepancy at a given node as long as at least as much has been done at another node on the same level. Moreover, the impact of performing an *i*-th discrepancy at a given node is supposed to be the same for all other nodes on the same level, and preliminary experiments showed that this hypothesis was not met for our industrial problem.

The next sections explain how the idea can however be adapted to support distributed optimization in a non-binary tree in order to be applied for supply chain coordination.

V. ADS: AN ADAPTIVE SEARCH STRATEGY FOR DISTRIBUTED OPTIMIZATION

This section proposes an adaptive search strategy (*Adaptive Discrepancy Search*, or ADS) that can be used by agents to solve a hierarchical distributed optimization problem (HDCOP). During the search process, agents collectively and dynamically identify the most promising areas of the tree in order to explore them first. It allows agents to systematically search the solution space (thus looking for the optimal solution) but aims at producing good solutions in a short amount of time.

The proposed approach can be described as an *adaptive backtracking strategy* (see Section IV). It exploits the facts that the tree is not binary and that we are facing an optimization problem. Since each subproblem (node) has many solutions (arcs), we have to backtrack many times to each node during the search process. Each time we will do so, we will measure how beneficial it has been to produce an *i*-th solution for the corresponding subproblem (that is, how beneficial it has been to allow an additional discrepancy on that node). We then seek to extrapolate for which node it would be more profitable to produce other local solutions (that is, allow additional discrepancies), and how many should be generated before it becomes better to pass on to another node.

The basic concepts are introduced informally in Section A using a naive example. We will then introduce formally a method that allows extrapolating (predicting) the contribution associated to performing an *i*-th discrepancy at a given node (B). Finally, we will define a backtracking strategy that uses this information (C), and a protocol that allows agents to implement it in a distributed scheme (D).

A. Main idea

To illustrate the basic concepts, we will consider a fictitious minimization problem represented by a non-binary tree.

Figure 7i shows the part of the tree that is known after reaching the first global solution. Let us suppose this solution's quality (as measured by the objective function) is equal to 1.0. At that moment, three nodes are candidates for backtracking (**a**, **b** and **c**). Backtracking to any of them would lead to a new global solution having a number of discrepancies equal to 1. If we were to apply an LDS policy, we would not have a preference for any of them. Let us suppose we backtrack to each of them in turn. We then get three new solutions of quality 1.2, 0.6 and 0.9 (subfigure ii).

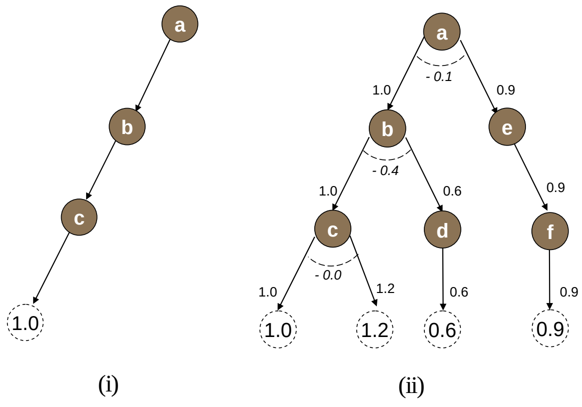


Figure 7. ADS - Illustrating the main idea.

We now have 6 nodes (**a** to **f**) that are candidates for backtracking (nodes **a, b** and **c** are still candidates since the tree is not binary). Again, these nodes are of equal interest from the point of view of an LDS policy (each would lead to a leaf having 2 discrepancies). However, we can see that for node **c**, the second arc led us to a leaf for which the solution quality is worse than for the first arc (1.2 > 1.0). For node **b**, the second arc produces an improvement of 0.4 (1.0 – 0.6). For node **a**, the improvement is 0.1 (1.0 – 0.9).

In short, although nodes **a**, **b** and **c** are equally appealing from an LDS point of view, when considering recent history it seems more interesting to produce a third solution from node **b**. As for nodes **d**, **e** and **f**, we have no information available but they could lead to promising areas. We thus have to make a choice between exploiting available information (and choosing node **b**) or exploring new areas. The following sections develop these ideas.

B. Predicting the effect of performing an i-th discrepancy at a given node

Let us suppose that a search is in progress. An *i*-th global solution has just been found and we need to backtrack in order to explore alternative solutions. Let $nodeList$ be the list of nodes $node$ that are candidates for backtracking (i.e. nodes for which there are unexplored local solutions for their local subproblems). For each $node \in nodeList$, we will denote by n the number of local solutions already explored (i.e. number of arcs exiting from this nodes already explored). We will define the following terms in the context of a specific $node$:

Definition 2 : $\text{Child}(i)$ corresponds to the node headed by the *i*-th arc of $node$. It is defined for $i = 0...n-1$. $\text{Child}(i)$ may be a leaf or an internal node. As an example, on Figure 7ii, for node **b** we have $n = 2$, $\text{Child}(0)$ corresponds to node **c** and $\text{Child}(1)$ to node **d**.

Definition 3 : $\text{NextLeaf}(i)$ corresponds to the first leaf we would encounter performing a depth-first search in the subtree having $\text{Child}(i)$ as its root. If $\text{Child}(i)$ is a leaf, then $\text{NextLeaf}(i)$ returns this leaf. Otherwise we have: $\text{Leaf}(i) := \text{Child}(i).\text{NextLeaf}(0)$.

Definition 4 : $\text{ArcValue}(i)$ corresponds to the quality of the first global solution that can be reached by following arc *i*. That is $\text{ArcValue}(i) := \text{NextLeaf}(i).score$. As an example, each arc on Figure 7ii is labeled by its value. One might use other definitions for $\text{ArcValue}(\ )$ (e.g. best solution in the subtree). However, the definition we proposed has the following advantage. Each time we backtrack to a node, we can measure the quality of the followed arc as soon we reach the next leaf (that is before the next backtrack) and this value remains unchanged for the rest of the search, which limits data updates.

Definition 5 : $\mathit{bestToDate}[\ ]$ is a vector of size *n* such as $\mathit{bestToDate}[i] = \min_{j=0}^{i} \mathrm{ArcValue}(j)$.

Figure 8 illustrates the relationship between $\mathrm{ArcValue}(\ )$ and $\mathit{bestToDate}[\ ]$. It shows a node for which $n = 4$ (subfigure i). Each arc $i = 0...3$ is labeled by its value $\mathrm{ArcValue}(i)$. Subfigure ii illustrates the apparent difficulty of extrapolating a value for $\mathrm{ArcValue}(4)$. In contrast, $\mathit{bestToDate}[\ ]$ is monotonic decreasing[1] (iii). It seems easier to extrapolate a value for $\mathit{bestToDate}[4]$ than for $\mathrm{ArcValue}(4)$.

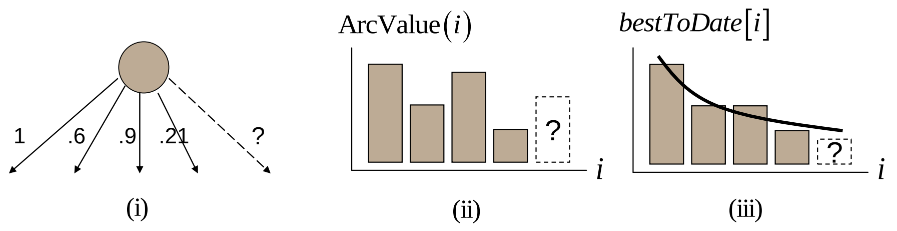


Figure 8. Relation between $\mathrm{ArcValue}(\ )$ and $\mathit{bestToDate}[\ ]$.

Definition 6: $\mathrm{F}(\ )$. We will suppose a continuous function $\mathrm{F}(\mathbb{R}^+) \to \mathbb{R}^+$ that approximates $\mathit{bestToDate}[\ ]$. This function will be generated by an algorithm named $\mathrm{A}(\ )$ that analyzes the already known value for $\mathit{bestToDate}[\ ]$. Formally, $\mathrm{A}(\ \mathit{bestToDate}[0...n-1]\ ) \to \left(\mathrm{F}(\mathbb{R}^+) \to \mathbb{R}^+\right)$.

The following shows how we can implement $\mathrm{A}(\ )$. Our principal concern is not to generate a function which explains well the values in the known range (*interpolation*), but rather to *extrapolate* a new one outside the known range. Generally, extrapolation represents an additional challenge in comparison with interpolation [47]. However, considering our definition for $\mathit{bestToDate}[\ ]$, algorithm $\mathrm{A}(\ )$ can limit itself considering only monotonic decreasing functions. For example, if one puts forward the hypothesis that $\mathit{bestToDate}[\ ]$ decreases from its initial value $(\mathrm{ArcValue}(0))$ by a constant rate β until it reaches an inferior limit α, algorithm $\mathrm{A}(\ )$ must then produce a function with the following form: $\mathrm{F}(i) := (\mathrm{ArcValue}(0) - \alpha)\, e^{-\beta i} + \alpha$. Figure 9 illustrates this situation. The role of algorithm $\mathrm{A}(\ )$ is then to estimate the value of parameters α and β for a given node. This could be done, for example, by achieving a least squares curve fitting using the *Gauss-Newton Algorithm* or the *Levenberg-Marquardt Algorithm* [47, 48].

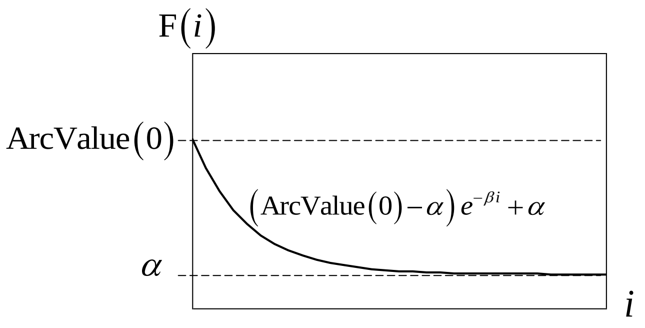


Figure 9. A simple model for function $\mathrm{F}(i)$.

[1] $\mathit{bestToDate}[i] \le \mathit{bestToDate}[i-1] \quad \forall i > 0$

Definition 7 : $improvement[\ ]$. Let us consider a node and two of its consecutive arcs, $i-1$ and i. They respectively lead to leaves having quality $arcValue[i-1]$ and $arcValue[i]$. The value of $improvement[i]$ corresponds to the improvement effected at the level of $bestToDate[\ ]$:

$$improvement[i] = bestToDate[i-1] - bestToDate[i]$$

Definition 8 : $\mathrm{Improvement}(\ )$. In a similar way, we will define the function $\mathrm{Improvement}(i)$ as being the expected improvement associated to the generation of an *i*-th local solution for the subproblem associated to current node, according to our approximation $\mathrm{F}(\ )$ for $bestToDate[\ ]$:

$$\mathrm{Improvement}(i) := F(i-1) - F(i)$$

For a given node *node* for which *n* local solutions have been generated, we are particularly interested in the value $\mathrm{Improvement}(n)$. This value is the extrapolation we seek, indicating whether an additional local solution (arc) is believed to lead to a global solution $(\mathrm{NextLeaf}(n))$ better than the one associated to the previous arcs. For simplification, we will write $\mathrm{Improvement}(\ )$ in place of $\mathrm{Improvement}(n)$ in the rest of the text.

C. Backtracking strategy

The strategy we propose is the following. As in SyncBB and SyncLDS, the first global solution is obtained by solving the subproblems sequentially. For the first few backtracks, we select a node using an LDS policy. When we have backtracked at least once for each level of the tree, we can start using the method of the previous section in order to choose the node to backtrack to. The nodes will be compared according to their $\mathrm{Improvement}(\ )$ value. We use $\mathrm{Improvement}(\ )$ rather than $\mathrm{F}(\ )$ for the following reason: using $\mathrm{F}(\ )$, a node having generated a good solution in the past would therefore be preferred even if it started generating bad solutions.

By definition, $\mathrm{Improvement}(\ )$ can only be computed for nodes for which at least two arcs have been explored. Consequently, only the nodes meeting this condition are considered. We also suppose there is a threshold value ε beyond which the improvement is significant in the application domain. Besides this value, we suppose it is preferable to explore other nodes. In practice, we can use ε=0. When not enough nodes qualify according to the previous conditions (less than 2), we apply an LDS policy to select a node. Doing so increases the number of nodes which will qualify in the future, and represents an opportunity to discover new promising nodes. No node (even if it is expected improvement is below ε) is discarded forever. The exploration of its remaining arcs is postponed until no other node seems more interesting. The method is complete (exploring the same search space as SyncBB or SyncLDS) but aims at producing good solutions in a short amount of time.

Each time a new global solution is found, data must be updated. This dynamically modifies the priority given to the nodes, and by doing so, brings about the adaptive character of the strategy. Next section presents the pseudocode performing this update. We than presents the pseudocode of a *node selector*[2] applying the proposed strategy.

1) Updating the Data

Let us suppose that a global solution (leaf) has just been obtained. One must then update the vector $bestToDate[\ ]$ and recompute the function $\mathrm{F}(\ )$ for certain ancestors of this leaf. Figure 10 presents the pseudocode achieving this update. The function `UpdateData` receives as arguments the quality of the leaf (`score`) and its unique identifier `p[]`. It is a vector of integers representing the path that would lead to the leaf in the corresponding global tree. The element `p[j]` defines, for a level `j`, which arc should be followed when going from the root to that leaf. The function `Card` returns the length of the vector. Finally, we recall that `nodeList` contains the nodes available for backtracking. Each node is defined by a tuple `<p[], bestToDate[], F()>`.

[2] In the present context, a *node selector* is a function that identifies the node to which we should backtrack to. It is called every time we need to backtrack.

The main loop navigates along the path `p[]`, from the leaf to the root. For the leaf's parent node, we always update $bestToDate[\ ]$ and $\mathrm{F}(\ )$. As for the upper nodes along the path, the update is not always necessary. Consider a node *node* and its *i*-th child *child* such as they are both on the path leading to *leaf*. If *leaf* is not the first leaf in the subtree rooted by *child*, then $leaf \neq node.\mathrm{NextLeaf}(i)$ and this leaf's score is of no use in the computation of *node*'s $\mathrm{ArcValue}(i)$ and $bestToDate[i]$ (according to Definitions 4 and 5). In this case, the nodes on the path that are above *node* do not need to be updated.

```
Procedure UpdateData(p[], score)
 do
 {
  i := p[Card(p)-1];
  remove last element from p; // p is now the path of the parent
  node := select node in nodeList : (node.p = p); // node is that parent
  if (i = 0) node.bestToDate[0] := score;
  else node.bestToDate[i] := Min(node.bestToDate[i-1], score);
  node.F := A(node.bestToDate);
 }
 while ((Card(p) > 0) and (i = 0))
```

Figure 10. Updating $bestToDate[\ ]$ and $\mathrm{F}(\ )$ when a global solution is found.

2) *Node Selector Implementing the Strategy*

This section describes a node selector implementing the backtracking strategy. Figure 11 presents the pseudocode (see `SelectNode`). Since the model is unusable as long as a minimum of information has not yet been accumulated, the first backtracks are produced by virtue of an LDS policy. This is done by giving priority to the nodes for which the next global solution created will have a total number of discrepancies inferior to or equal to 1. We also prefer the LDS policy if too few nodes meet the ADS selection criteria (enforced by `FilterADS`). Nodes for which `n` is equal to zero also have priority as this corresponds to normal descent of the tree (when no backtracking is required). The function `CompareLDS` allows applying the LDS policy. It compares two nodes and returns the one of highest priority. The arguments of this function are: the path in the global tree of the next local solution each node would generate (thus, the concatenation of `p[]` and `n` as a new vector). In the case of equality, the equivalent of a chronological backtrack is applied to separate between the nodes (`CompareBT`).

```
Function SelectNode(nodeList)
 nodeListADS:= FilterADS(nodeList)
 if (Card(nodeListADS) < 2)
 or (∃ node in nodeList) : ((SumOfDisc(node)+node.n ≤ 1) or (node.n = 0))
   candidate := select node in nodeList according to function CompareLDS()
 else
   candidate := select node in nodeListADS : node.Improvement() is maximal
 return candidate

Function FilterADS(nodeList)
 return all node in nodeList : (node.n >= 2) and (node.Improvement() > ε)

Function SumOfDisc(node)
 return Σ(j=0..Card(node.p[])-1) node.p[j]

Function CompareLDS(p1, p2)
 t1 := Σ(j=0..Card(p1)-1) p1[j]
 t2 := Σ(j=0..Card(p2)-1) p2[j]
 if (t1 < t2) return p1
 else if (t2 < t1) return p2
 else return CompareBT(p1, p2)

Function CompareBT(p1, p2)
 depth := Min(Card(p1), Card(p2))
 j := 0
 while (p1[j] = p2[j] and j < depth) j := j+1
 if (j < depth)
    if (p1[j] ≤ p2[j]) return p1 else return p2
 else
    if (Card(p1) ≥ Card(p2)) return p1 else return p2
```

Figure 11. Node selector implementing the ADS strategy.

```
WhenReceive MsgProposition(<d,p[]>) do
 nodeList.add(<d, p[], n=0>)
 Work(node)

Procedure Work(node)
 proposition := NextSolution(node);
 if (proposition ≠ ∅)
   node.n := node.n+1
   if (Successor(node) ≠ ∅)
     send MsgProposition(<proposition, node.p[]+[node.n-1]>) to Successor(node)
   else
     UpdateBestToDate(node, node.n-1, proposition.score);
     node.F := A(node.bestToDate[])
     if (node.n = 1) send MsgQuality(score, node.p[]+node.n-1) to Predecessor(node)
     CooperativeBacktracking()
 else
   nodeList.remove(node)
   CooperativeBacktracking()

Procedure UpdateBestToDate(node, i, score)
 if (i = 0) node.bestToDate[0] := score
 else node.bestToDate[i] := Min(node.bestToDate[i-1], score)

WhenReceive MsgQuality(score, p[]) do
 node := select node in nodeList : p[] begins with node.p[]
 i := p[Card(node.p)-1]
 UpdateBestToDate(node, i, score)
 if (i = 0) and (Predecessor(node) ≠ ∅)
    send MsgQuality(score, p[]) to Predecessor(node)

Procedure CooperativeBacktracking()
 send MsgAskNbADSQualifiedNodes to Everybody // including itself
 nbQualifiedNodesADS := Σ answer from Everybody
 send MsgAskBestLocalNode() to EveryBody
 answers := all answer from EveryBody : (answer ≠ ∅)
 if (nbQualifiedNodesADS < 2)
 or (∃ node in answers) : ((SumOfDisc(node)+node.n ≤ 1) or (node.n = 0))
   candidate := select node in answers[] according to function CompareLDS()
 else
   candidate := select node in answers[] : node.Improvement() is maximal
 send MsgBacktrackADS(node) to Agent(answer)

WhenReceive MsgAskNbADSQualifiedNodes() do return Card(FilterADS(nodeList))

WhenReceive MsgAskBestLocalNode() do return SelectNode(nodeList)

WhenReceive MsgBacktrack(node) do Work(node)
```

Figure 12. Pseudocode for SyncADS. Each agent executes this pseudocode at the same time. They communicate using the following mechanism. An agent sends a message to another using the following syntax: “send MsgName(content) to AgentName”. On the receiving side, the pseudocode defined by the label “WhenReceive MsgName(content)” is automatically executed on reception of the message.

D. Distributed implementation

This section introduces a protocol (SyncADS) allowing agents to perform distributed search while applying the previous adaptive backtracking strategy (ADS). Strictly speaking, the global tree exists nowhere, but the global solutions will be visited in the same order as if one was carrying out centralized search in the equivalent tree. As in SyncBB and SyncLDS, only one agent at a time is active. The transition from one agent to the other takes place by the exchange of messages that could be seen as the transmission of a privilege (or token). The term *synchronous* refers to the fact that an agent cannot select/change the solution for his local problem asynchronously (that is at any moment).

Each agent manages a list of nodes/subproblems under its authority (`nodeList`) and executes the pseudocode in Figure 12. For a given agent, the main procedure (`MsgProposition`) is activated when the agent receives a proposition from the previous one (or from the external customer). This proposition is denoted by a couple `<d,p[]>`. The element `d` represents the decisions for the previous subproblems and `p[]` is a vector of integers representing the path leading to the corresponding node in the global tree. Upon receiving this message, the agent creates a node corresponding to the new subproblem to solve and then adds it into `nodeList`. It then begins solving this instance of the subproblem (`Work`), finds a first solution and sends it to the next agent as a proposition (`send MsgProposition`). If there is no following agent, then we have on hand a solution for the global problem. The agent then updates its data (`UpdateBestToDate` and `F()`), informs its predecessors about the new solution quality (the message `MsgGlobalSolQuality` is propagated upward) and starts the cooperative backtracking mechanism (`CooperativeBacktracking`).

In function `CooperativeBacktracking`, each agent is asked to identify which node under its authority would be locally chosen (each agent selects it using the `SelectNode` function in Figure 11. Agents are also asked to count how many nodes qualify according to the ADS filtering criteria. Knowing that information, the calling agent can identify the node/subproblem with highest priority (using code similar to Figure 11 `SelectNode`). It then gives control to the agent responsible for that subproblem (`MsgBacktrack`). Please note that agents do not really send nodes as we do in pseudocode. They only need to communicate the vector `p[]` and the value `n`.

VI. Evaluation

We will first apply the proposed approach to a real supply chain coordination problem from the forest products industry (Section A). This case has the advantage of being well documented, relies on real data, and has already been used in published works. We will evaluate the gains associated with the use of the adaptive strategy, and the quality of the predictive function $F(\ )$ we defined. Finally, we will evaluate the approach using a wider range of synthesized problems, in order to generalize the obtained results (Section B).

A. Evaluation with industrial data: Coordination in a lumber production supply chain

A decade ago, the Canadian lumber industry was confronted with the need to reengineer the way they manage and plan their supply chain operations. Together with researchers, they created the FORAC Research Consortium with the mission to develop *Advanced planning and scheduling systems* (APS). This section first describes a typical lumber supply chain, and the planning problems and tools developed for each manufacturing unit, as they were described in [28, 49]. We will then evaluate the performance of the proposed adaptive coordination mechanism. Basic distributed search was already evaluated for this case study in [32]. Here, we will evaluate the improvement that can be obtained by using the adaptive search strategy.

1) *Lumber production supply chain*

Figure 13 presents the different production units involved in softwood lumber production: (1) the sawing facility, where logs are cut into various sizes of rough pieces of lumber; (2) the drying facility, which reduces the lumber moisture content; and (3) the finishing facility, where lumber is planed (surfaced), trimmed and sorted.

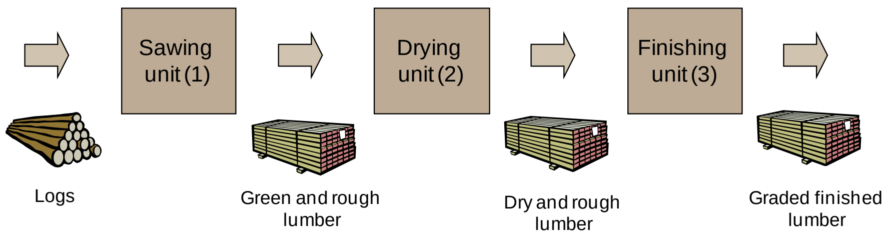


Figure 13. Softwood lumber production supply chain.

Sawmilling produces multiple types of lumber at the same time (co-production) from a single type of log at the input (divergence). Many setup configurations are possible for the plant during each shift. Each configuration limits the log types and cutting patterns that can be used. Therefore, the planning decisions the production manager must make are the following: (1) decide how the plant will be set up for each production shift, and (2) decide which quantities of each log class to consume at each production shift.

Once logs are sawn, pieces of lumber are assembled into bundles of the same *dimension* (2”x3”, 2”x4”, etc.) and *length* (8-foot, 12-foot, 16-foot, etc.), and generally of the same *species* (spruce, fir, etc.) in order to be dried.

Lumber drying is a transformation operation which aims at decreasing the lumber moisture content in order to meet customer requirements. These requirements are usually specified by industry standards, although some customers may require specific levels of moisture content. Softwood lumber drying is a rather complex process to carry out. It takes days and it is done in batches within large kiln dryers. Bundles of lumbers of different length can be dried in the same batch (e.g. 8-foot and 16-foot), but lumbers must be of the same dimension and species (although there are some exceptions).

For a given batch of green lumber, there are different possible alternative operations that can be used for air-drying and kiln-drying. The planning decisions for this production unit are the following: (1) what drying activities to perform, (2) what loading pattern to use, and (3) when to perform them.

At the finishing facility, lumber is first planed (or surfaced). Lumber is then sorted according to its grade (i.e. quality) with respect to the residual moisture content and physical defects. Lumber may be trimmed in order to produce a shorter lumber of a higher grade and value. This process is usually optimized by hardware to yield products with the highest value, with no consideration for the actual customer demand. It is common to obtain more than 20 different types of products from a single product and the exact products mix to obtain from a batch depends on the drying process used. There is also a setup cost each time the facility processes a different dimension (e.g. from 2”x3” to 2”x6”). To sum up, the decisions that must be taken in order to plan the finishing operations are the following: (1) which campaign to realize (i.e. which lumber dimensions), (2) when and for how long and (3) for each campaign, which quantities of each length to process. Figure 14 shows a simple example of a production plan, including the campaigns (2”x3”, 2”x6” and 2”x4”) and the time spent on each length.

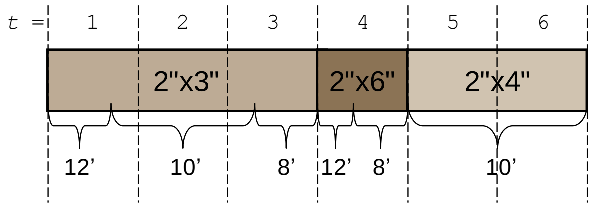


Figure 14. Production plan for a finishing line for six consecutive production shifts.

2) *Local decision-making*

Each of these local planning problems were described formally in [49]. Each production unit is able to plan its own production using a different specialized tool/algorithm (see [32, 49, 50]). Sawing operations are planned using a Mixed Integer Linear Programming model. Drying operations and finishing operations are planned using Constraint Programming models. Each unit can utilize its algorithm to produce a local production plan aiming to deliver the products ordered by its customer with the least possible delay (e.g. the sawing unit seeks to deliver on time what the drying unit has ordered, etc).

3) *Coordination*

This situation has been modeled as a *Hierarchical Distributed Constraint Optimization Problem* (HDCOP) in [32]. By default, the units apply the *Two-phase planning* protocol in order to synchronize their operations (see Figure 15 and Section III). Considering that each unit is able to produce alternative local plans, the coordination spaces can be modeled as a tree (HDCOP) and the coordination process is assimilated into a search in this tree.

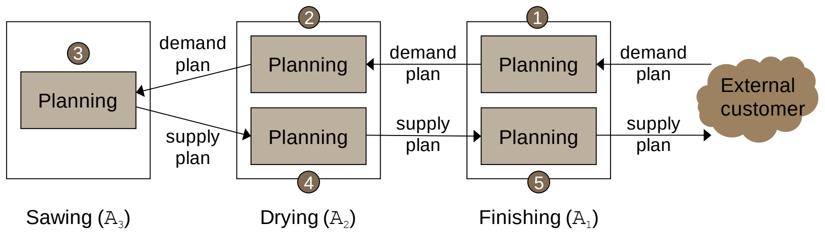


Figure 15. Subproblems hierarchy for lumber production.

Synchronous Branch and Bound (SyncBB) and *Synchronous Limited Discrepancy Search* (SyncLDS) were evaluated and compared using industrial data in the mentioned article. Even for a large computation time, SyncBB persisted in exploring only minor variations of the first solutions. In contrast, SyncLDS allowed better sampling of the solution space as LDS rapidly explores different areas of the tree. As a consequence, the quality of the solutions found by SyncBB stops improving after a short computation time, while it continues to improve with SyncLDS - until it finally reaches a plateau after around an hour of calculation time. In the following subsection, we will use the same search trees in order to evaluate SyncADS.

4) *Experiments*

SyncADS will be evaluated for the aforementioned supply chain coordination problem. We will measure the gain associated to the use of the adaptive strategy (SyncADS), in comparison with what was reported as the best non adaptive technique (SyncLDS) in [32].

We will also evaluate different predictive function $\mathrm{F}(\ )$ used for the extrapolation of $bestToDate[\ ]$. First (1), the function based on the learning parameters α and β presented in Section V. We recall it makes the assumption that $bestToDate[\ ]$ decreases according to a constant rate β until it reaches a plateau α. We therefore have: $\mathrm{F}(i) := (\mathrm{ArcValue}(0) - \alpha)\, e^{-\beta i} + \alpha$.

Second (2), we will test a simplified version where it is supposed that $bestToDate[\ ]$ decreases until it reaches zero: $\mathrm{F}(i) := \mathrm{ArcValue}(0)\, e^{-\beta i}$.

Function 1 is expected to give better results than Function 2. However, Function 2 has the advantage that there is only one parameter to fit, and this can be done using simple linear regression. For evaluation purposes we used the *Levenberg-Marquardt Algorithm* (LMA) to fit/update all parameters[3]. This standard non-linear least-squares minimization method [47] has the following advantage over the *Gauss-Newton Algorithm*. Although for both algorithms we need to provide initialization values for the parameters, LMA is less dependent on the quality of the initialization values. In our experiments, we used the following initial values: $\beta = 0.5$ and $\alpha = 0$.

For comparison, we will also test (3) a second-degree polynomial function : $F(i) := \gamma i^2 + \lambda i^2 + \varphi$. Although we should expect good interpolation capacity from this function (i.e. it should well fit the known points of the curve), it is expected to show a rather poor extrapolation capacity, as it is not a monotonic decreasing function.

5) *Results*

Table 1 compares the time needed to get the best solution. It shows the reduction of computation time (%) achieved when using SyncADS instead of SyncLDS for the four industrial cases studied. Function 1 allowed an average reduction of **47.3%**, in comparison with **43.2%** for Function 2. As for the third function, it is with no surprise that we obtained poor results (which, in our opinion, is explained by the weak extrapolation capacity of this function, see next section).

Table 1. Computation time needed to get best solution – Reduction (%) made possible by SyncADS (vs SyncLDS).

Predictive function	*Case #1*	*Case #2*	*Case #3*	*Case #4*	***Average***
1. $F(i) := (\mathrm{ArcValue}(0) - \alpha)\, e^{-\beta i} + \alpha$	47.6 %	30.9 %	55.1 %	55.7 %	47.3 %
2. $F(i) := \mathrm{ArcValue}(0)\, e^{-\beta i}$	42.9 %	25.7 %	50.1 %	54.2 %	43.2 %
3. $F(i) := \gamma i^2 + \lambda i^2 + \varphi$	-17.5 %	-3.9 %	-7.8 %	-4.2 %	-8.4 %

Table 2 presents the average reduction of computation time needed to get solutions of intermediate quality. This performance measure was proposed to verify that SyncADS allows obtaining solutions of intermediate quality (not just the best one) for a computation time equal or less than SyncLDS. The metric is calculated in the following way. Let us consider the execution of SyncLDS. The quality of the “best solution to the global problem found up to now” evolved over time. For any level of quality reached by SyncLDS, we evaluate the time necessary for SyncADS to reach a solution that is equal or better.

For both the predictive functions 1 and 2, the reduction is smaller than for the previous indicator (time needed to get the best solution). Indeed, the more one is ready to accept poor solutions, the more the advantage of SyncADS over SyncLDS diminishes. The explanation is as follows. For both algorithms it takes less time to find poor solutions than good solutions; SyncADS then has less time to learn and to distinguish itself over SyncLDS. We will further study this relationship between the performance of ADS and

[3] We use the following implementation: *Levenberg-Marquardt.NET*, by Kris Kniaz. See http://kniaz.net

computation time in Section B. As for the third predictive function, we again observe a poor performance, but there is no clear correlation between results for this indicator and the previous one.

Table 2. Average reduction (%) of the computation time needed to get solutions equal or better than SyncLDS.

Predictive function	*Case #1*	*Case #2*	*Case #3*	*Case #4*	***Average***
1. $F(i) := (\mathrm{ArcValue}(0) - \alpha)\, e^{-\beta i} + \alpha$	31.2 %	2.5 %	44.9 %	37.1 %	28.9 %
2. $F(i) := \mathrm{ArcValue}(0)\, e^{-\beta i}$	26.6 %	11.5 %	41.2 %	36.6 %	29.0 %
3. $F(i) := \gamma i^2 + \lambda i^2 + \varphi$	25.9 %	-6.3 %	23.4 %	-5.4 %	9.4 %

6) *Quality of the predictive functions*

We have seen in the previous section that SyncADS allowed better performance than SyncLDS for the industrial problems. In our opinion, two elements explain this result. First (1), the proposed predictive functions $\mathrm{F}(\ )$ model well *bestToDate*$[\ ]$ and allow good extrapolation of further values. Second (2), the curve/profile for *bestToDate*$[\ ]$ is relatively different from one node to another (i.e. there are some subproblem instances for which it is more worthwhile to generate alternative local solutions). Indeed, if this curve had been identical for all the nodes, it would have been futile to choose the backtracking candidate on the basis of this; an LDS strategy would have given equivalent results.

In order to verify hypothesis 1, we measured the gap between the forecasts (i.e. the values returned each time $\mathrm{F}(i)$ is called) and the actual values of *bestToDate*$[i]$ known *a posteriori*. We measured an average error of **1.6%** for Function 1, **3.6%** for Function 2 and **9.6%** for Function 3. We note that the function with the best extrapolation is also the one that gave the best results in previous section.

To verify hypothesis 2, we proceeded as follows. For each node we characterized its vector *bestToDate*$[\ ]$ using a single value β. We simply took the values in *bestToDate*$[\ ]$ (they are all known after the search) and best fit them.

Figure 16 shows the distribution for β in the industrial data. The trend line shows the distribution is close to being exponential $\left(e^{-12.718x}\right)$, defined for the interval $[0,\ldots,0.5]$. Let us recall that nodes for which β is close to zero correspond to subproblems instances for which it has been unprofitable to generate alternative solutions (these are therefore very numerous in our industrial case). Nodes with a bigger β have more potential (and the strategy seeks to dwell on these in priority).

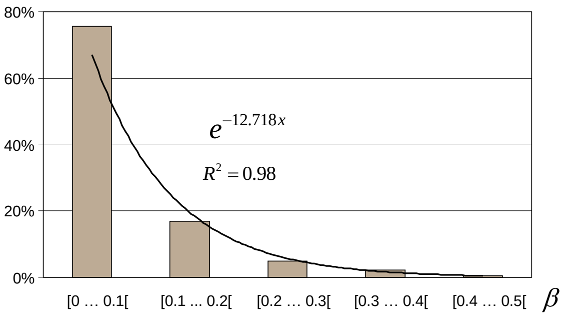


Figure 16. Distribution of the values β for the nodes of the studied trees.

In a similar way, for each node we observed the last value in the vector *bestToDate*$[\ ]$ in order to estimate α. For our node population, values for α are distributed more or less uniformly between $\mathrm{ArcValue}(0)$ and $0.5 \times \mathrm{ArcValue}(0)$.

B. Evaluation with synthesized data: Generalization

In the trees corresponding to our industrial coordination problems, there are some node/subproblems for which it is more worthwhile to produce alternative solutions (these are the nodes for which we measured large β). The performance of our approach is strongly related to the distribution of the values for β. Here, we will generate new datasets that will be more or less favorable to our algorithm, in order to study some of its characteristics. We will also study the impact of the distribution for δ, and the impact of the number of subproblems (that is, the depth of the tree).

We will suppose minimization problems such as the first leaf of the tree corresponds to a solution with quality equal to 1. For each node, we randomly choose a value for parameter β using the probability distribution $\mathrm{PR}(\beta = x) := e^{-\gamma x}$. For our industrial cases we measured $\gamma = -12.718$ but we will try other values. In a similar way, the values for α will be randomly chosen using a uniform distribution defined between 0 and $\delta \times \mathrm{arcValue}[0]$. Knowing values β and α for each node, we can calculate the quality of other leaves using Function 1. The trees are generated dynamically during search so the total number of nodes will depend on the computation time allowed to the search.

To introduce our evaluation framework, we will first study the following case. We have trees corresponding to a hierarchy of 4 subproblem types, generated using $\gamma = 10$ (considering $0 \le \beta \le 0.5$) and $\delta = 0$. This case was chosen because it allowed producing solutions with a score very close to 0 within reasonable computation time. Figure 17i shows the quality of the best solution found so far, according to computation time (measured as the number of visited nodes) for SyncLDS and SyncADS. We can see that for large computation time, both allow very good solutions. Subfigure (ii) presents the reduction in computation time allowed by SyncADS (in comparison with SyncLDS) for equal solution quality. As an example, obtaining a solution with a score equal to 0.4 (i.e. a reduction of the objective function of **60%**) takes approximately half the time using SyncADS. We can also point out that that the relative advantage of SyncADS decreases for scores very close to 0. For both methods the score tends towards 0 for large computation time (subfigure i) and the relative advantage tends to diminish.

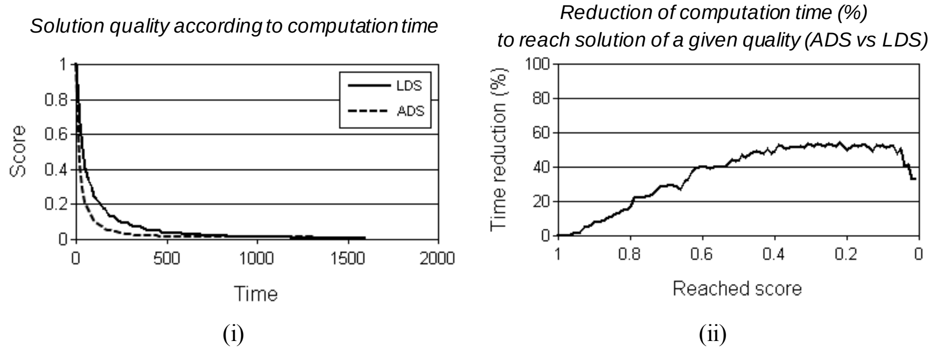
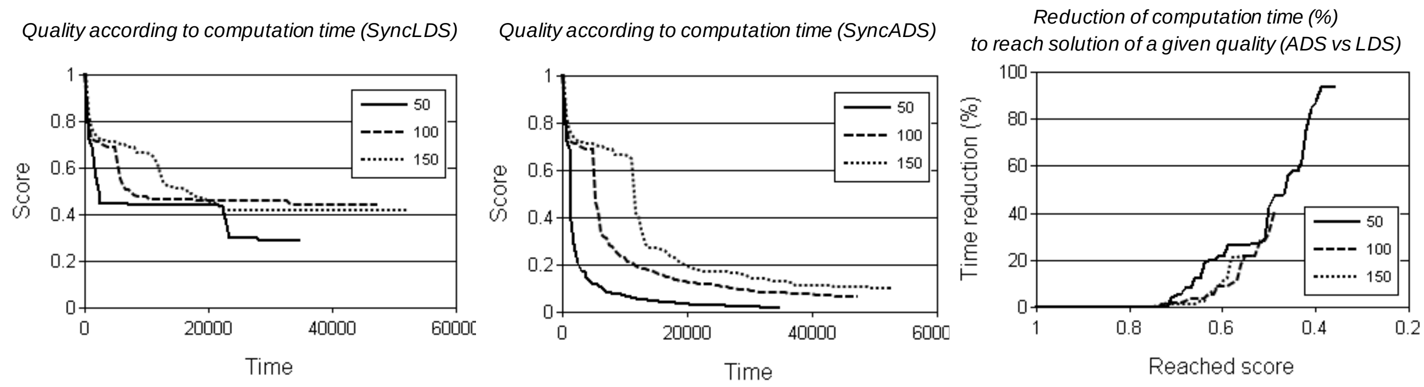


Figure 17. Comparison of SyncLDS and SyncADS for trees of depth=4, with [$\gamma = 10$; $0 \le \beta \le .5$; $\delta = 0$].

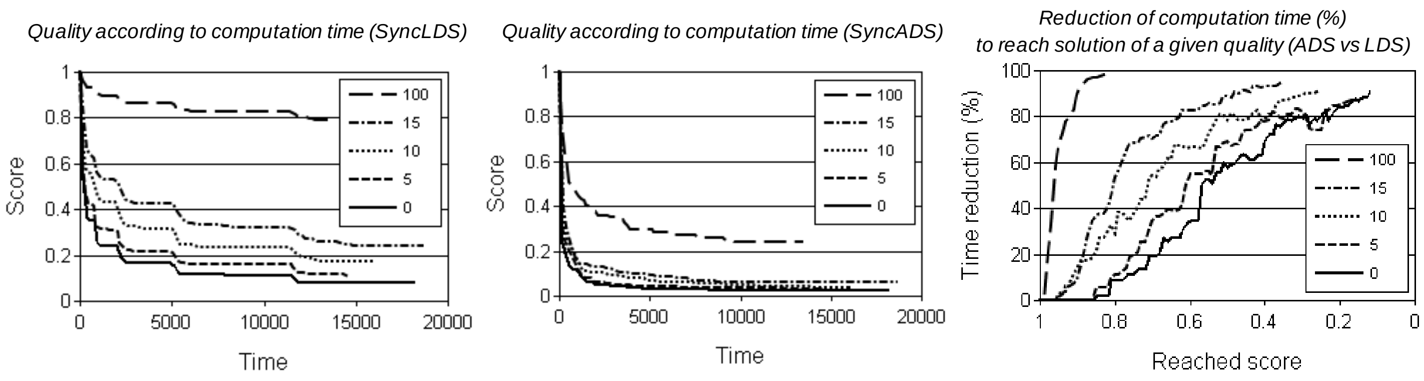
The next experiment (Figure 18) shows how the number of subproblems (that is, the depth of the tree) affects the performance. Subfigures (i) and (ii) show solution quality according to computation time for SyncLDS and SyncADS (for depth=50, 100, 150). Subfigure (iii) shows the reduction in computation time permitted by ADS. Two details receive our attention. With ADS, the quality of the solutions improves more gradually and continuously according to computation time. However, at the beginning of the search the results for ADS are identical to those for LDS. This corresponds to the initialization phase of our algorithm, where backtracking is performed like LDS. We can see (subfigure ii) that we quickly reach a plateau (especially noticeable for depth=150). The end of this plateau corresponds to the end of the initialization phase.



(i) (ii) (iii)

Figure 18. Impact of the number of subproblems (depth=50, 100, 150), with [$\gamma = 10$; $0 \le \beta \le .5$; $\delta = 0$].

The results shown in Figure 19 demonstrate the impact of parameter γ. Once again, we have assumed the values of β to be between 0 and 0.5 according to an exponential distribution $e^{-\gamma x}$. With $\gamma = 0$, the values β are chosen from a uniform distribution. The higher γ is, the fewer nodes there are with high β. Stated otherwise, the greater γ is, the fewer nodes there are for which it is profitable to produce a great number of discrepancies. It should thus be more difficult to find good solutions (hypothesis 1) and the search for and detection of 'profitable' nodes should be worthwhile (hypothesis 2). The subfigures (i) and (ii) confirm hypothesis 1; for a same strategy, the quality curves are less and less good as γ grows. Subfigure (iii) confirms hypothesis 2; the greater γ is, the more the ADS strategy has a significant advantage over the LDS strategy. These results empirically illustrate the following intuitive idea: the rarer the 'promising' nodes are, the more searching for and remaining with them is worthwhile. Even for $\gamma = 0$, we can observe an advantage of ADS over LDS. This is because there is still variability in the tree (and thus some nodes are more interesting than others) even if the β values are taken from a uniform distribution[4].



(i) (ii) (iii)

Figure 19. Impact of parameter γ (0, 1, 10, 15, 100), with [$0 \le \beta \le .5$; depth=10; $\delta = 0$].

Finally, the last experiment illustrates the impact of parameter δ. Let us recall that the α values of the nodes (the value towards which $bestToDate[i]$ tends for the high i) are chosen between $arcValue[0]$ and $\delta \times arcValue[0]$ according to a uniform distribution. When we have $\delta = 0$, then for every node the value $\text{Next Leaf}(i)$ tends toward 0 for a high i. The higher δ is, the more variability there is in the tree and the more the learning becomes profitable. This is illustrated by the results in Figure 20.

[4] If we use the same value β for any node in the tree, then ADS reports the same results as LDS (not shown on the chart).

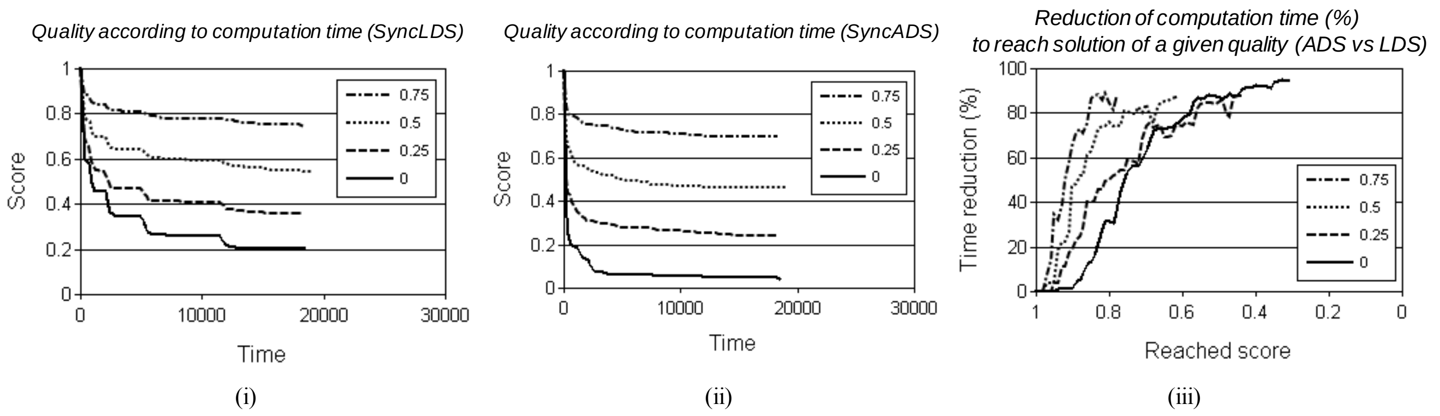


Figure 20. Impact of parameter δ (0, 0.25, 0.50, 0.75), with [$\gamma = 10$; $0 \le \beta \le .5$; depth=10].

VII. Conclusion

The contributions of this work are situated at different levels. Firstly, we have demonstrated the potential of adaptive search methods in an industrial context, more precisely for supply chain coordination. We applied the proposed method (ADS) to a real industrial coordination problem in the Canadian forest industry for which it reduced by almost half the computation time needed to get the best solution. As well, we have evaluated ADS for synthesized problems. It allowed evaluating the performance of the algorithm with a wider range of problems, according to how difficult it is to find nodes leading to good global solutions. ADS qualified as an excellent algorithm for distributed optimization, with the limitation that the problem needs to be formalized as a Hierarchical Distributed Constraint Optimization Problem (HDCOP).

By doing so, we have shown how the basic principles of adaptive approaches can be extended to support distributed optimization in non-binary trees (rather than centralized constraint satisfaction using binary trees, as is usually the case in the literature). Working on a real optimization problem, we have brought to light the fact that a certain structure exists in the problem, which can be identified and dynamically exploited during search with the help of a learning model analyzing the non-binary tree.

A question remains, however. Could the proposed adaptive method prove interesting in a classical/centralized optimization context? While in a centralized environment it is common usage for solvers to binarize the search space, one wonders whether directly exploiting the non-binary tree would allow finding and exploiting a structure in the problem, with the help of the learning model. This question remains open for future work.

REFERENCES

[1] V. R. Lesser, "An overview of DAI: viewing distributed AI as distributed search", *Journal of Japanese Society for Artificial Intelligence*, vol. 5, pp. 392-400, 1990.

[2] W. Ruml, "Adaptive Tree Search", Ph.D. Thesis, Harvard University, 2002.

[3] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*. New York: Springer, 2008.

[4] J. M. Swaminathan, S. F. Smith, and N. M. Sadeh, "Modeling supply chain dynamics: a multiagent approach", *Decision Sciences*, vol. 29, pp. 607-632, 1998.

[5] T. Moyaux, B. Chaib-draa, and S. D'Amours, "Supply chain management and multiagent systems: an overview", in *Multiagent-Based Supply Chain Management*, B. Chaib-draa and J. P. Müller, Eds. New York: Springer, 2006.

[6] D. Castle and F. R. Jacobs, *Operations management body of knowledge framework*. Chicago: APICS The Association for Operations Management, 2008.

[7] C. Chandra and J. Grabis, "Scope of supply chain configuration problem", in *Supply chain configuration: concepts, solutions and applications*, C. Chandra and J. Grabis, Eds. Berlin: Springer, 2007, pp. 17-45.

[8] S. Kumar and J. Bisson, "Utilizing analytic hierarchy process for improved decision making within supply chains", *Human Systems Management*, vol. 27, pp. 49-62, 2008.

[9] A. C. Hax and D. Candea, *Production and Inventory Management*. Engelwood Cliffs: Prentice-Hall, 1984.

[10] G. P. Cachon, "Supply chain coordination with contracts", in *Supply chain management: design, coordination and operation*, A. G. de Kok and S. G. Graves, Eds. Amsterdam: Elsevier, 2003.

[11] H. Stadtler, "Supply chain management and advanced planning: basics, overview and challenges", *European Journal of Operational Research*, vol. 163, pp. 575-588, 2005.

[12] T. Moyaux, B. Chaib-draa, and S. D'Amours, "Information sharing as a coordination mechanism for reducing the bullwhip effect in a supply chain", *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 37, pp. 396-409, 2007.

[13] J. Collins, W. Ketter, and N. Sadeh, "Pushing the limits of rational agents: the trading agent competition for supply chain management", *AI Magazine*, vol. 31, pp. 63-80, 2010.

[14] J. Collins, R. Arunachalam, N. Sadeh, J. Eriksson, N. Finne, and S. Janson, "The supply chain management game for the 2007 trading agent competition", Carnegie-Mellon University CMU-ISRI-07-100, 2006.

[15] J. B. Rice and R. M. Hoppe, "Supply chain vs. supply chain: the hype and the reality", *Supply Chain Management Review*, pp. 46-53, 2001.

[16] C. Kilger and B. Reuter, "Collaborative planning", in *Supply Chain Management and Advanced Planning*, H. Stadtler and C. Kilger, Eds., Third ed. New York: Springer, 2005, pp. 259-278.

[17] A. Fink, "Supply chain coordination by means of automated negotiations between autonomous agents", in *Multiagent-base supply chain management*, B. Chaib-draa and J. P. Müller, Eds. New York: Springer, 2006.

[18] J. F. Shapiro, *Modeling the supply chain*, Second ed. Belmont: Thomson-Brooks/Cole, 2007.

[19] P. Beaumont, "Multi-platform coordination and resource management in command and control", M.Sc. Thesis, Université Laval, Quebec City, 2004.

[20] T. Dean and M. Boddy, "An analysis of time-dependant planning", Seventh National Conference on Artificial Intelligence (AAAI), Saint Paul, U.S.A., 1988, pp. 49-54.

[21] W. Shen, L. Wang, and Q. Hao, "Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey", *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 36, pp. 563-77, 2006.

[22] C. Chandra and J. Grabis, "Information technology support for integrated supply chain modeling", *Human Systems Management*, vol. 27, pp. 3-13, 2008.

[23] C. Schneeweiss and K. Zimmer, "Hierarchical coordination mechanisms within the supply chain", *European Journal of Operational Research*, vol. 153, pp. 687-703, 2004.

[24] G. Dudek and H. Stadtler, "Negotiation-based collaborative planning between supply chains partners", *European Journal of Operational Research*, vol. 163, pp. 668-687, 2005.

[25] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie, "Applications of agent-based systems in intelligent manufacturing: an updated review", *Advanced Engineering Informatics*, vol. 20, pp. 415-431, 2006.

[26] L. Monostori, J. Vancza, and S. R. T. Kumara, "Agent-Based Systems for Manufacturing", *CIRP Annals - Manufacturing Technology*, vol. 55, pp. 697-720, 2006.

[27] M. S. Fox, M. Barbuceanu, and R. Teigen, "Agent-oriented supply-chain management", *International Journal of Flexible Manufacturing Systems*, vol. 12, pp. 165-88, 2000.

[28] J.-M. Frayret, S. D'Amours, A. Rousseau, S. Harvey, and J. Gaudreault, "Agent-based supply chain planning in the forest products Industry", *International Journal of Flexible Manufacturing Systems*, vol. 19, pp. 358-391, 2007.

[29] J.-M. Frayret, "A multidisciplinary review of collaborative supply chain planning", IEEE International Conference on Systems, Man and Cybernetics, San Antonio, Texas, 2009, pp. 4414-4421.

[30] L. Sheremetov and L. Rocha-Mier, "Supply chain network optimization based on collective intelligence and agent technologies", *Human Systems Management*, vol. 27, pp. 31-47, 2008.

[31] J. Gaudreault, J.-M. Frayret, and G. Pesant, "Discrepancy-based Method for Hierarchical Distributed Optimization", Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Patras, Greece, 2007, pp. 75-81.

[32] J. Gaudreault, J. M. Frayret, and G. Pesant, "Distributed Search for Supply Chain Coordination", *Computers in Industry*, vol. 60, pp. 441-451, 2009.

[33] C. Schneeweiss, *Distributed Decision Making*. New York: Springer, 2003.

[34] R. Bhatnagar, P. Chandra, and S. K. Goyal, "Models for multi-plant coordination", *European Journal of Operational Research*, vol. 67, pp. 141-160, 1993.

[35] K. Hirayama and M. Yokoo, "Distributed partial constraint satisfaction problem", Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP), LNCS #1330, Linz, Austria, 1997, pp. 222-236.

[36] W. D. Harvey and M. L. Ginsberg, "Limited discrepancy search", Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995, pp. 607-613.

[37] C. Le Pape and P. Baptiste, "Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem", *Journal of Heuristics*, vol. 5, pp. 305-325, 1999.

[38] F. Hutter, D. Babic, H. H. Hoos, and A. J. Hu, "Boosting verification by automatic tuning of decision procedures", Formal Methods in Computer Aided Design Conference (FMCAD), Austin, Texas, 2007, pp. 27-34.

[39] S. L. Epstein, E. C. Freuder, and R. J. Wallace, "Learning to support constraint programmers", *Computational Intelligence*, vol. 21, pp. 336-371, 2005.

[40] E. Breimer, M. Goldberg, D. Hollinger, and D. Lim, "Discovering optimization algorithms through automated learning", *DIMACS series in discrete mathematics and theoretical computer science*, vol. 69, pp. 7-27, 2005.

[41] W. Karoui, M. J. Huguet, P. Lopez, and W. Naanaa, "YIELDS: a yet improved limited discrepancy search for CSPs", Proceedings of the 4th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR), LNCS #4510, Brussels, Belgium, 2007, pp. 99-111.

[42] P. Refalo, "Impact-based search strategies for constraint programming", International Conference on Principles and Practice of Constraint Programming (CP), LNCS #3258, Toronto, Canada, 2004, pp. 557-571.

[43] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints", Proceedings of the 16th European Conference on Artificial Intelligence (ECAI), Valencia, Spain, 2004, pp. 146-150.

[44] D. Grimes and R. J. Wallace, "Learning from failure in constraint satisfaction search", Proceedings of the AAAI Workshop, Boston, Massachusetts, 2006, pp. 7-14.

[45] N. Levasseur, P. Boizumault, and S. Loudni, "A value ordering heuristic for weighted CSP", Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Patras, Greece, 2007, pp. 259-62.

[46] W. Ruml, "Heuristic Search in Bounded-depth Trees: Best-Leaf-First Search", Working Notes of the AAAI-02 Workshop on Probabilistic Approaches in Search, Edmonton, Canada, 2002.

[47] W. H. Press, *Numerical recipes the art of scientific computing*. Cambridge: Cambridge University Press, 2007.

[48] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, pp. 431-441, 1963.

[49] J. Gaudreault, P. Forget, A. Rousseau, J.-M. Frayret, and S. D’Amours, "Distributed operations planning in the lumber supply chain: Models and coordination", *International Journal of Industrial Engineering: Theory, Applications and Practice*, vol. 17, pp. 168-189, 2010.

[50] J. Gaudreault, J. M. Frayret, A. Rousseau, and S. D'Amours, "Combined planning and scheduling in a divergent production system with co production: a case study in the lumber industry", *Computers and Operation Research*, vol. 38, pp. 1238-1250, 2011.

Jonathan Gaudreault is currently research professor at *Université Laval* and codirector of FORAC, a research consortium dedicated to the development of decision support systems for the forest products industry. He holds a Ph.D. from the *École Polytechnique de Montréal*. His thesis concerned coordination in supply chains using distributed planning. Jonathan previously worked as a R&D manager in the software industry. He was awarded as "One to watch" by the *Fédération de l'Informatique du Québec* in 1995 and 1999 (OCTAS award). His main research interests include distributed artificial intelligence, operations management, distributed optimization and simulation.

Gilles Pesant is currently Professor of Computer and Software Engineering at the *École Polytechnique de Montréal*, Canada, after postgraduate studies at *McGill*, *Université de Montréal*, and the *Centre for Research on Transportation*. For the last twenty years, his research interests have focused on Constraint Programming and its application to workforce scheduling, transportation logistics and telecommunications. Until recently he had worked mostly on inference algorithms for constraints but his focus has turned to search, specifically generic search heuristics built from the number of solutions of individual constraints. He has authored more than fifty refereed papers in the area and has given several invited tutorials on Constraint Programming. He is regularly involved in the scientific programming of international conferences in the area. He serves as an Associate Editor for *Constraint Programming Letters*, *Journal of Heuristics*, *INFOR*, and is the Editor-in-Chief of *Constraints*.

Jean-Marc Frayret is currently Associate Professor at the *École Polytechnique de Montréal*, Canada. He holds a PhD in Mechanical Engineering from *Université Laval*, Québec City, Canada. He is a regular member of the CIRRELT, a research centre dedicated to the study of network organizations and logistics. His research interests include agent-based and distributed manufacturing systems, supply chain management and interfirm collaboration. Dr. Frayret has published several articles in these fields in various journals and international conferences. Between 2002 and 2007, Dr. Frayret was also Associate Director of Research at the FORAC Research Consortium.

Sophie D'Amours is professor in industrial engineering at Université Laval and member of the CIRRELT and FORAC Research Consortium. She holds a Canada Research Chair in Planning Sustainable Forest Networks and an Industrial NSERC Research Chair in the domain of value chain optimization in the forest sector. Her research expertise is in the area of collaborative planning of value chains. She has applied most of her research to the forest sector and published her work in industrial engineering and operations research related journals. Finally, she is the Scientific Director of the Strategic NSERC Network VCO.

Captions

Table 1. Computation time needed to get best solution – Reduction (%) made possible by SyncADS (vs SyncLDS).

Table 2. Average reduction (%) of the computation time needed to get solutions equal or better than SyncLDS.

Figure 1. Example of a supply chain. Adapted from [5]. Arcs represent product flows between business units.

Figure 2. Local plan and product flow for a simple supply chain.

Figure 3. Two-phase planning protocol as coordination mechanism for the previous supply chain.

Figure 4. Coordination space for Two-phase planning within HDCOP.

Figure 5. Example of an execution trace for SyncBB.

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

Figure 7. ADS - Illustrating the main idea.

Figure 8. Relation between $\text{ArcValue}(\ )$ and $bestToDate[\ ]$.

Figure 9. A simple model for function $\mathrm{F}(i)$.

Figure 10. Updating $bestToDate[\ ]$ and $\mathrm{F}(\ )$ when a global solution is found.

Figure 11. Node selector implementing the ADS strategy.

Figure 12. Pseudocode for SyncADS. Each agent executes this pseudocode at the same time. They communicate using the following mechanism. An agent sends a message to another using the following syntax: "send MsgName(content) to AgentName". On the receiving side, the pseudocode defined by the label "WhenReceive MsgName(content)" is automatically executed on reception of the message.

Figure 13. Softwood lumber production supply chain.

Figure 14. Production plan for a finishing line for six consecutive production shifts.

Figure 15. Subproblems hierarchy for lumber production.

Figure 16. Distribution of the values β for the nodes of the studied trees.

Figure 17. Comparison of SyncLDS and SyncADS for trees of depth=4, with [$\gamma = 10$; $0 \le \beta \le .5$; $\delta = 0$].

Figure 18. Impact of the number of subproblems (depth=50, 100, 150), with [$\gamma = 10$; $0 \le \beta \le .5$; $\delta = 0$].

Figure 19. Impact of parameter γ (0, 1, 10, 15, 100), with [$0 \le \beta \le .5$; depth=10; $\delta = 0$].

Figure 20. Impact of parameter δ (0, 0.25, 0.50, 0.75), with [$\gamma = 10$; $0 \le \beta \le .5$; depth=10].

Footnotes

[1] $bestToDate[i] \le bestToDate[i-1] \quad \forall i > 0$

[2] In the present context, a *node selector* is a function that identifies the node to which we should backtrack to. It is called every time we need to backtrack.

[3] We use the following implementation: *Levenberg-Marquardt.NET*, by Kris Kniaz. See http://kniaz.net

[4] If we use the same value β for any node in the tree, then ADS reports the same results as LDS (not shown on the chart).