Discrepancy-based Method for Distributed Supply Chain Operations Planning

Jonathan Gaudreault[1,2], Jean-Marc-Frayret[1,2] and 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. 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 factories can implement 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 strategies such as Limited Discrepancy Search (LDS). We therefore propose MacDS, a novel mechanism allowing the agents to implement a search strategy based on discrepancies (LDS or others) while allowing concurrent computation. Use of this mechanism improved quality of solutions and computation time for real industrial problems.

Introduction

This paper studies the case of industrial supply chains where agents represent factories offering services to the other factories (Figure 1). An external client announces a call for bids and the cooperation of each factory 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. The partners 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. However, the factories may be competing against each other for other projects. Therefore privacy is an important issue; each factory wants to plan its own activities, doesn't know alternative production processes of the others, etc.

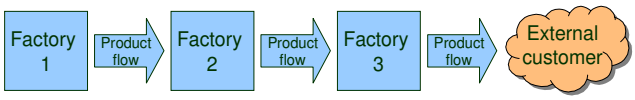

Figure 1: Example of a simple supply chain

Supply Chain Operations Coordination

The traditional supply chain management literature focuses on coordination practices found in real life supply chains, with a few exceptions (responsibility tokens, Porteus 2000). Indeed, this literature studies inventory policies in a basic coordination context, including transactional information exchange, request for quote/tender or JIT kanban cards.

Recent literature in supply chain management proposes more advanced coordination frameworks involving various forms of negotiation and information exchange scheme (e.g. Dudek and Stadtler 2005). Most of the time, negotiation methods are defined for contexts in which agents have different goals but wish to reach an agreement, are defined for binary partnership only or involves a mediator agent.

Finally, the multi-agent community dedicated to building advanced information systems for network enterprises proposes a literature that particularly emphasizes the design of interaction protocols for agent coordination. A review is presented in (Frayret et al. 2005). These protocols can be described as coordination heuristics.

The most common class of coordination heuristics is the 'hierarchical' approach (de Kok and Fransoo 2003), both in the literature and industrial practice. Figure 2 shows an example of a hierarchical method. The agent first makes a temporary plan to compute its needs in raw materials. 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.

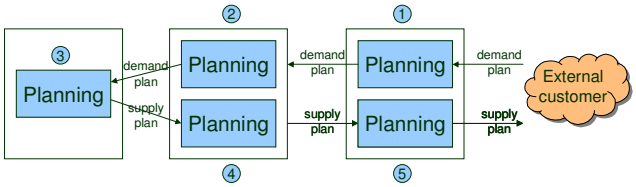

Figure 2: Two-phase planning

Coordination Space as a Tree

In this section we propose to generalize the hierarchical approach described in the previous section. The algorithms used to make each local decision usually allow producing alternative solutions (Kilger and Reuter 2005). By considering each of these propositions we can describe the coordination space as a tree. The tree has one level per type of subproblem (each level will correspond to one of the boxes in Figure 2). 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.

The simplest method for the agents to collectively explore this tree is to perform what Hirayama calls *Synchronous Branch and Bound* (SyncBB) (Hirayama and Yokoo 1997). 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 methods based on discrepancies like Limited Discrepancy Search (LDS) (Harvey and Ginsberg 1995). We propose a method allowing the agents to implement a search strategy based on discrepancies (like LDS or others) and allowing concurrent computation.

Multi-agent Concurrent Discrepancy Search

Limited Discrepancy Search (LDS) was the first method based on discrepancies (Harvey and Ginsberg 1995). It was proposed for centralized problems. The main idea is that the leaves of the tree (solutions) do not all have the same expected quality; that it decreases with the number of times one branch 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. LDS aims to first visit the leaves with the fewest discrepancies. Another effect of LDS is that the solutions visited in a given period of time will be from more different parts of the tree than those produced using chronological backtracking.

Proposed Algorithm (MacDS)

We will first describe the algorithm informally. Each agent manages a list of nodes (corresponding to the alternative solutions from the previous agent). With each solution to its subproblem sent to the next agent, the agent attaches the 'path' that would go from the root of the global tree to this specific node. At all times, each agent works on the node/subproblem with the highest priority from its list. The priority of a node is a function of its path and the number of alternative solutions already sent for that subproblem (i.e. the path of next child to be sent). This approach is similar to implementing a backtracking strategy in centralized search using a node selector (Beck and Perron 2000), except that it is applied locally by each agent. By changing the selector function, it is possible to implement different known strategies: LDS or others (even chronological backtracking).

Pseudocode. The following objects are manipulated by the algorithm:

- A message `msg` is a couple `<d,p>` where `d` represents the solutions for the previous subproblems and `p` is a vector of integers representing the path. The element `p[j]` defines, for a level `j`, which arc should be followed when going from the root to the corresponding node in the global tree.
- 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 solutions (`noMoreSol`).

Each agent runs many threads: one for each node plus a control thread. A single thread per agent is active at any time. The control thread (Figure 3) is activated when the agent receives a message (`WhenReceiveMsg`) and when the agent has just produced 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`). In the pseudocode, we suppose that the list of nodes is sorted by decreasing priority (according to the chosen policy, e.g. LDS).

```
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 ← ∅;
```

Figure 3: Control thread of the agent

The pseudocode for the node threads is shown in Figure 4. When a node is created, its thread is idle. It must be activated by the control thread. When a thread produces a new subproblem solution, it signals this fact to the control thread (`SignalNewSolution`) and goes idle (`sleep`). The control thread then sends the message to the agent that owns the next subproblem (`Successor`).

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

Figure 4: Thread associated to a node

Evaluation

We compared MacDS (applying LDS policy) to SyncBB using real industrial data with complex subproblems. The case is a supply chain coordination problem in the forest products industry (Frayret et al. 2005). The network has three facilities (Sawing, Drying and Finishing). They apply two-phase planning (Figure 2) in order to minimize orders' lateness. The data were extracted from the company databases at different moments in 2005.

For both SyncBB and MacDS, the first global solution and computation time are always the same (same as standard two-phase planning). Consequently, we compared the algorithms according to the reduction of the objective function they achieved with additional computation time (in seconds). Figure 5 illustrates the results for the four industrial cases studied. The industrial impact of both algorithms is huge. MacDS outperforms SyncBB in a significant manner, except for one case (ii) here SyncBB provides the best solution by 0.5% given a computation time greater than 1000 sec.

Conclusion

From a supply chain management point of view, we showed how hierarchical approaches can be generalized and the coordination space 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 improvement in solution quality. As for distributed optimization, we showed how discrepancy-based methods can be applied in a distributed and concurrent context. It improved computation time and quality for both real problems and generated trees (results not shown here).

References

Beck, J.C., Perron, L. 2000. Discrepancy-Bounded Depth First Search. *Proc. of CPAIOR*, 7-17. Paderborn, Germany.

de Kok, A.G., Fransoo, J.C. 2003. Planning Supply Chain Operations. In: *Supply Chain Management*. de Kok, A.G. and Graves, S.C. eds. Amsterdam: Elsevier.

Dudek, G., Stadtler, H. 2005. Negotiation-based collaborative planning between supply chains partners. *European Journal of Operational Research*. 163(3): 668-87.

Frayret, J.M.et al. 2005. *Agent-based Supply Chain Planning in the forest products industry.* Québec: CENTOR, Université Laval. DT-2005-JMF-1.

Harvey, W.D., Ginsberg, M.L. 1995. Limited discrepancy search. *Proc. of IJCAI*, 607-613. Montreal, Can: Morgan Kaufmann.

Hirayama, K., Yokoo, M. 1997. Distributed partial constraint satisfaction problem. *Int. Conf. on Principles and Practice of Constraint Programming, LNCS #1330*, 222-236. Linz : Springer.

Kilger, C., Reuter, B. 2005. Collaborative Planning. In: *Supply Chain Management and Advanced Planning*. Stadtler, H. and Kilger, C. eds. New York: Springer.

Porteus, E.L. 2000. Responsibility tokens in supply chain management. *Manufacturing and Service Operations Management*. 2(2): 203-19.

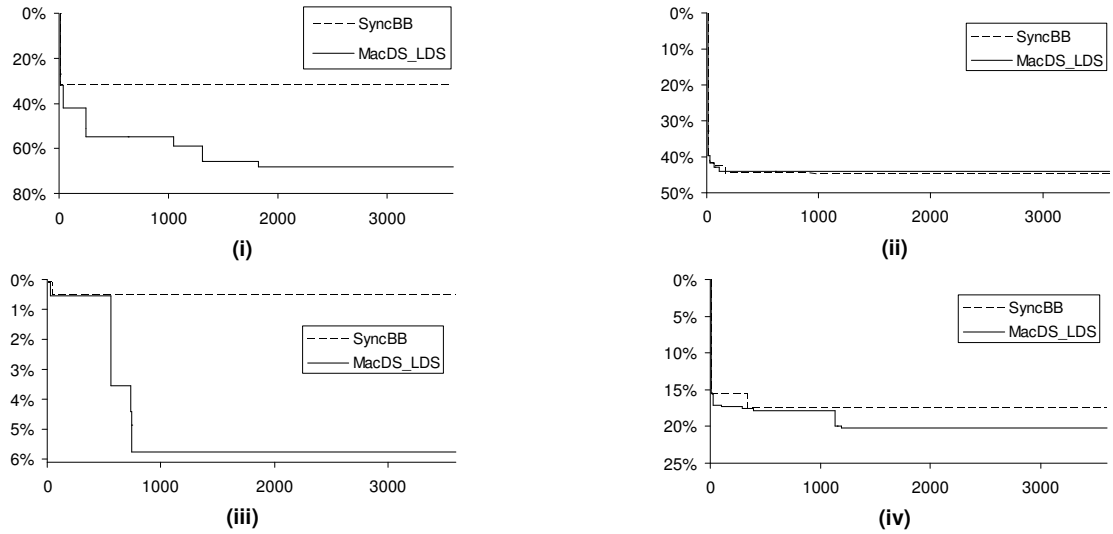

Figure 5: Reduction of the objective function, according to computation time (in seconds) for cases (i), (ii), (iii) and (iv)