CIRRELT
Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport
Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Concurrent Discrepancy-based Search for Distributed Optimization

Jonathan Gaudreault
Jean-Marc Frayret
Gilles Pesant

January 2007

CIRRELT-2007-09

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

CONCURRENT DISCREPANCY-BASED SEARCH FOR DISTRIBUTED OPTIMIZATION

Jonathan Gaudreault [1], Jean-Marc Frayret[1,2,3], Gilles Pesant[2,3]

[1] Research Consortium in e-Business in the forest products industry (FOR@C)

[2] École Polytechnique de Montréal, Québec, Canada

[3] Centre interuniversitaire de recherche sur les réseaux d'entreprises, la logistique et le transport (CIRRELT)

Abstract. Distributed Constraint Optimization is increasingly used to formalize problem solving by multiple agents. However, there are situations where agents represent an organization made up of heterogeneous agents (e.g. network of companies) in which the context, the structure, and the business rules define the interactions that are possible between them. The solution space for those hierarchical problems is reminiscent of the ones of centralized combinatorial problems. Therefore, we propose a distributed algorithm (MacDS) that performs discrepancy-based search which is known to perform well for centralized problems. The proposed algorithm is complete and aims at producing good solutions in a short amount of time. It allows concurrent computation and is tolerant to message delays. It has been evaluated using real industrial problems with complex subproblems, for which it showed good performance.

Keywords. Distributed constraint optimization; multi-agent; discrepancy-based search; supply chain; lumber industry.

Acknowledgements. This work was funded by the Research Consortium in E-Business in the Forest Products Industry (FOR@C) and supported by the Interuniversity Research centre on Enterprise Networks, logistics and transportation (CIRRELT).

---

* Corresponding author: jean-marc.frayret@cirrelt.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2007

Introduction

Many generic algorithms have been proposed in recent years to solve Distributed Constraint Optimization Problems (DCOP). In (Modi and Veloso 2005), the authors underline that those generic algorithms do not always take advantage of the characteristics of a particular class of problems, e.g. specific constraint structures or domain specific heuristics. Moreover, there are situations where the agents solving the problem represent an organization that exists *a priori* and is made up of heterogeneous agents (e.g. network of companies). The structure of the organization and its context may limit the type of relation and interactions between them (Horling and Lesser 2004).

In this paper, we study a class of distributed problems that are hierarchical in nature. As for classic DCOP, there is an objective function that agents try to minimize. However:

- The problem is partitioned into subproblems and there is a sequence in which they must be solved, imposed by the business domain;
- The agents are constrained when solving a subproblem by the decisions of the previous ones;
- The objective can be formulated as a function of the variables of one subproblem (most of the time the last one).

A wide range of problems meet those criteria, notably many planning problems in a hierarchical context. Schneeweiss describes many problems from industrial applications in (Schneeweiss 2003).

We will especially study the case of industrial supply chain networks (Moyaux, Chaib-draa and D'Amours 2006). More precisely, using the following example where agents are factories offering services to the others agents. An external client announces a call for bids for a product and the work of each plant 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 want to put together a common production plan (e.g. what to do, where and when). Each agent has a local vision: it can only think about its own production, inputs and outputs. Business rules state a resolution sequence; a factory cannot calculate its needs for raw material and send them to its supplier without knowing what it is asked to produce, neither can it plan its production before knowing what supply is granted by the supplier. These define the two subproblems of each agent.

The objective function represents the client's interest, e.g. minimize lateness. Because agents do not know alternative possibilities for the agents they supply, a bound on the objective function cannot be computed until it is time to solve the last subproblem.

The consortium is in competition with others; if the external client rejects the first proposal, it is urgent to propose alternative solutions before it accepts a proposition from another consortium. Our goal is therefore to produce good solutions quickly.

In the following sections, we will formally describe the problem and show that few DCOP methods apply in this context. We will then propose MacDS, a concurrent method allowing the agents to systematically explore the solution space, but aiming at producing good solutions first by using a backtracking strategy based on the computation of discrepancies. The algorithm will then be evaluated for both random data and real industrial problems.

Problem Definition

A Hierarchical Distributed Constraint Optimization Problem (HDCOP) is defined over a set of variables $X=\{X_1,\ldots,X_m\}$. Each variable X_i can take a value from a set D_i. The set of variables X is partitioned into disjoint subsets (subproblems). Each variable is part of a subproblem $S(X_i)\in S=[S_1,\ldots,S_n]$. Each one is owned by an agent (an agent may own many subproblems). Formally, $A(X_i)=A(S(X_i))\in A=\{A_1,\ldots,A_p\}$. Subproblems must be solved in the order defined by S. Each subproblem S_j is constrained by previous decisions, as stated by a predicate $C_j(\bigcup(S_1,\ldots,S_j))$. Agents wish to minimize a function $F(S_q)$ where $S_q\in S$.

We suppose that agents know algorithms to solve their subproblems and they produce the solutions in an order defined by this algorithm. In that context, we can represent the global solution space as a tree, where each level $j=1,\ldots,n$ corresponds to subproblem S_j. Each node on that level represents an instance of that subproblem (defined by previous decisions). Each arc is an alternative and feasible solution to that subproblem. Arcs are ordered according to the local algorithm used by the agent.

DCOP

The previous problem could be reformulated as a classic DCOP such as in (Modi, Shen, Tambe, and Yokoo 2005). To keep the notion of subproblems, we need to consider that each subproblem in the original HDCOP formulation is now a variable in the DCOP. The domain of each new variable is the Cartesian product of the original variables. The original variables cease to exist. Unfortunately, we then lack the representation of the mandatory solving sequence. We also miss the fact that each time a variable must be valued, a complex subproblem must be solved and that we must use the local solver of the agent. In contrast, most DCOP algorithms specify the rule the agent must use for value ordering for that variable/subproblem. For those reasons, we will prefer the HDCOP definition.

Algorithms for Classical DCOP

The simplest algorithm is Synchronous Backtracking (SyncBT) (Yokoo, Ishida, Durfee and Kuwabara 1992). It mimics chronological backtracking in the tree representing the solution space (for our HDCOP, the one presented with our definition). Agents solve the subproblems in sequence. The messages transmitted by agents represent current partial assignments (CPA). In the case of a dead end, or when a global solution is found, a chronological backtrack

occurs. Of course, the method is complete. Synchronous Branch-and-Bound (SyncBB) improves SyncBT by calculating a bound on the objective in each node. The value is used to prune the tree and guide the value ordering (Hirayama and Yokoo 1997). For a hierarchical problem where the objective is represented by a function of the variables of the last subproblem and the agents lack a good representation of the other subproblems, the bound would be equal to zero at each node. SyncBB is then equivalent to SyncBT. In Asynchronous Forward-Bounding (AFB) (Gershman, Meisels and Zivan 2006), when an agent assigns a value it transfers the CPA to every following agent. Agents then compute bounds concurrently.

Some methods allow the agent to assign a value to variables asynchronously, as soon as it supposes this change could improve the global solution. Distributed local search does this but it is incomplete (Sun, Zhang, and Nee 2001; Hirayama and Yokoo 1997). Asynchronous Distributed Optimization (ADOPT) was the first method both asynchronous and complete (Modi, Shen, Tambe, and Yokoo 2005) but the algorithm needs agents to be able to calculate good bounds and change their values accordingly. It therefore violates our assumption about agents using specialized solvers.

Another approach is distributed dynamic programming. The best known algorithm is DPOP (Petcu and Faltings 2005) for which different improvements have been proposed over time. It makes the assumption that agent $A(S_j)$ can solve S_j before S_{j-1} is solved. $A(S_j)$ is asked for the best solution for S_j for any potential solution of S_{j-1}. This supposes that $A(S_j)$ has a good representation of the domain for S_{j-1} and can solve S_j to optimality in reasonable time.

Multi-Agent Concurrent Discrepancy Search

The solution space for the introduced hierarchical distributed optimization problems is reminiscent of the ones for centralized combinatorial problems. The search tree (see HDCOP definition) is equivalent to one for a centralized problem given a variable ordering heuristic and a value ordering heuristic.

In such a centralized context, chronological backtracking is most of the time outperformed by search methods based on the computation of discrepancies, both for satisfaction and optimization problems (Harvey and Ginsberg 1995; Le Pape and Baptiste 1999). These strategies have the characteristic of not relying on bound calculation.

This section introduces a new distributed algorithm (MacDS) that performs discrepancy-based search. It is complete (exploring the same search space as SyncBT) but aims at producing good solutions in a short amount of time. It allows concurrent computation, uses asynchronous communication, an asynchronous timing model (Lynch 1996) and is tolerant to random message transmission delays. It also takes advantage of situations where subproblem solving times vary from one agent to another.

Discrepancy-based Search

Limited Discrepancy Search (LDS) was the first method based on discrepancies (Harvey and Ginsberg 1995). The main idea is that the leaves of the tree (solutions) do not have the same expected quality; that it decreases with the number of times one should 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 more different from one another than those produced using chronological backtracking. It is this interesting characteristic we seek in our distributed algorithm. In the original description by Harvey, LDS was a search procedure, but the idea can be used to specify a node selector: when backtracking conditions occurs, the search engine must select the node for which the next unvisited child has the fewest discrepancies (Beck and Perron 2000). This is why LDS can be described as a backtracking policy.

LDS has been applied with success to optimization problems (Le Pape and Baptiste 1999). For n-ary trees, they proposed to count the discrepancies as follows: the **i-th** arc followed at a given level counts as **i-1** discrepancies. Over time, other discrepancy-based methods have been proposed (Walsh 1997; Beck and Perron 2000). Discrepancy-based search is integrated into commercial solvers as, for example, ILOG Solver.

Concurrent Search Algorithms

Classic algorithms for DCOP and DisCSP (distributed satisfaction problems) exploit two different approaches to achieve concurrency (Zivan and Meisels 2006). The first is the asynchronous approach used by ADOPT, as described previously. The second is called Concurrent Search Algorithm (CSA) by (Zivan and Meisels 2006). They exploited it in the ConcDB algorithm (for DisCSP). With this approach, each global solution is constructed sequentially by the agents, but agents collectively work on many solutions. A concurrent algorithm must specify how and when a new "path" must be explored. Our algorithm makes use of this form of concurrency.

Proposed Algorithm (MacDS)

We will first describe the algorithm informally using a simple example. Agents `A={B,C,D}` should solve the problem made-up of subproblems `S=[B,C,D]`. Each global solution is constructed sequentially by the agents. They solve the subproblems as ordered in the vector. The first agent (`B`) uses its local solver. Once it has a first solution, agent `B` sends it to agent `C` in a message named "`B0`" (first solution for subproblem `B`). Agent `C` then repeats the same and transmits the message "`B0-C0`" (first solution of `C` according to the first solution of `B`) to the next agent. But, as soon as the first agent has sent its solution "`B0`" to the second agent, it started looking for an alternative decision ("`B1`") using its local solver. As soon

as "B1" is ready, it is sent asynchronously to agent C, whether or not it has already found a solution for "B0". When receiving this message, agent C must ask itself whether it is better to continue working on "B0-C0" or begin to work on "B1-C0". This decision will be based on discrepancies (the names of messages contain information to compute them).

Essentially, each agent manages a list of tasks, supplied asynchronously by the previous one. Tasks are prioritized locally according to the discrepancy profile of the next message the task would generate. The policy used to compare them defines a backtracking strategy that will be enforced collectively by all agents. The policy is implemented by a function comparing two profiles and selecting the one with greater priority. It is possible to implement it based on different known strategies: LDS, DDS, SBS, DFS, etc. (in the latter case, we would then perform a concurrent synchronous backtracking).

In the extreme case where a single agent owns every subproblem, then MacDS visits the nodes of the tree in the same order as a centralized search algorithm (applying the same backtracking strategy) would.

In a distributed context where each agent manages its own task list, each solution to the global problem will be obtained in no more time than would be necessary for the centralized algorithm (ignoring communication delays). Each agent is working as soon as there is a task in its list, but preempts its current work when a task with greater priority is added to the list.

The algorithm is complete since it explores the same solution space as SyncBT; it only changes the sequence in which nodes will be visited. In the case of communication breakdown (or in presence of random messages transmission delays), the agent always works on the available task with the greatest priority, rather than remaining idle. Therefore, it is not mandatory for messages to arrive in the same order as they were sent.

Pseudocode. The following objects are manipulated by the algorithm:

- A message `msg` is a couple `<d,p>` where `d` represents the decisions for the previous subproblems and `p` is a vector of integers representing the discrepancy profile. The element `p[j]` defines, for a level `j`, which arc should be followed when going from the root to the "current node" in the solution tree.
- A list of tasks (`tasks`) contains the running and waiting tasks of the agent. A task is defined by `d` and `p`, by the number of local solutions produced to date for the task (`i`) and by a boolean indicating if the subproblem solver thinks there is no more solution (`noMoreSol`).

Each agent runs many threads: one for each task plus a control thread. A single thread per agent is active at any moment. The control thread (Figure 1) is activated when the agent receives a message (`WhenReceiveMsg`) and when a task has just produced a new subproblem solution (`WhenNewSolution`). It then updates the task list and transfers control to the thread of the task with the greatest priority (`ActivateTask`).

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

WhenNewSolution(task)
  task.Sleep();
  SendMessage(successor(task), <task.d +
              task.sol.d, task.p + task.i>)
  task.i++;
  if (task.noMoreSol)
       tasks.Remove(task);
       running ← ∅;
  ActivateATask();

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

Figure 1: Control thread of the agent

The pseudocode for the task threads is shown in Figure 2. When a task is created, its thread is idle. It must be activated by the control thread. When the task produces a new subproblem solution, it signals it to the control thread (`SignalNewSolution`) and goes idle (`sleep`). It is the control thread that sends the message to the agent that owns the next subproblem.

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

Figure 2: Thread implementing a task

In the shown pseudocode, we suppose that tasks are ordered by decreasing priority. Figure 3 shows examples of comparator functions that can be used to maintain the list sorted. They identify, over a pair of discrepancy profiles, which one have greater priority, according to a backtracking strategy. The first one (`CompareBT`) defines a chronological backtracking policy. The other one (`CompareLDS`) defines an LDS policy. When one calls those functions, the discrepancy profile representing a task is the concatenation of `task.p` and `task.i`.

Evaluation

MacDS was evaluated with generated data and with a real industrial problem.

Evaluation with Generated Data. We suppose n-ary trees randomly generated, such that the probability that a leaf is the best solution is proportional with $\delta^{(p)}$, where `p` is the number of discrepancies of the leaf and δ is a parameter that varies from 0.1 to 1.0 (by steps of 0.1). When δ is maximal, all leaves have the same probability. The

performance measure used is the expected time needed to find the best solution on these trees. Others parameters we experimented with were: number of subproblems (Card(S)), message transmission delay (τ), subproblem solving time (α), and the number of solutions for each subproblem (n). This experiment was conducted in a simulation environment similar to (Modi, Shen, Tambe, and Yokoo 2005).

```
CompareBT(p1, p2)
  depth ← Min(Card(p1), Card(p2));
  j ← 1;
  while (p1[j] = p2[j] && j ≤ depth) j++;
  if (j ≤ depth)
       if (p1[j] ≤ p2[j]) return p1;
       else return p2;
  else
       if (Card(p1) ≥ Card(p2).depth()) return
p1;
       else return p2;
 CompareLDS(p1, p2)
  t1 ← Σ(j=1..Card(p1)) p1[j];
  t2 ← Σ(j=1..Card(p1)) p2[j];
  if (t1 < t2) return p1;
  else
       if (t2 < t1) return p2;
       else return CompareBT(p1,p2);
```

Figure 3: Comparators implementing two backtracking policies

We compared MacDS using a LDS policy (MacDS_LDS) with SyncBT. To measure which part of the gain was due to concurrency and which to the backtracking policy, we also tested a version of MacDS using chronological backtracking (MacDS_BT) and also implemented SyncLDS (synchronous as SyncBT, but performing LDS).

MacDS_LDS is always the best of the four algorithms (caught up by MacDS_BT when δ=1.0). For both backtracking policies (BT and LDS), the MacDS version always beats the synchronous one. The following figures shows results when δ=0.7, which is representative of the average case.

Figure 4(a) shows that message delay (τ) has a linear impact for all four algorithms, but a much smaller one for both versions of MacDS. For them, expected time needed to find the best solution is equal to (Card(S)-1)τ. Figure 4(b) shows that subproblem solving time (α) also has a linear impact, again smaller for the MacDS versions. Figure 4(c) shows that the impact of the number of subproblems (Card(s)) is exponential.

Industrial Evaluation. We then evaluated the algorithms using real industrial data with complex subproblems. The case is a supply chain coordination problem in the forest products industry. The data were extracted from the company databases at different moments in 2005.

The network has three plants $A=\{A_1,A_2,A_3\}$ and four subproblems S=[V,W,X,Y]. The equivalent centralized problem has millions of solutions. In terms of product flow, the agents form a chain. But, the decision flow is more complex (see Figure 5). Agent A_2 is in contact with the external customer. An customer order is a set of tuples <product,dueDate,quantity>. A_2 first computed a demand for A_1 (subproblem V). A_1 then builds its production plan and allocates supply to A_2 (subproblem W). A_2 builds its production plan and allocates supply to A_3 (subproblem X). Then, A3 plans its production (subproblem Y). The objective is to minimize the sum of order lateness, which is a function of the variables in Y.

Each local solver was made available by FORAC, a consortium of companies and researchers. Agent A_1 plans its sawing operations using a Mixed Integer Linear Programming model. Agent A_2 uses Constraint Programming to plan and schedule its wood drying operations. Agent A_3 schedules its wood finishing operation using a forward-scheduling heuristic, but uses a DDS backtracking strategy to produce alternative

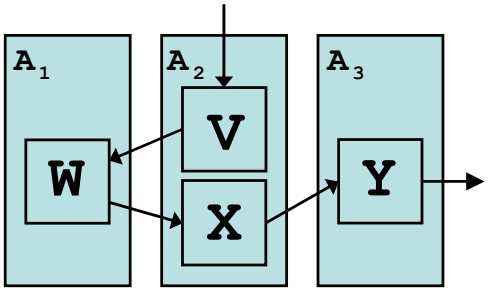

Figure 5: Agents, subproblems and solving sequence

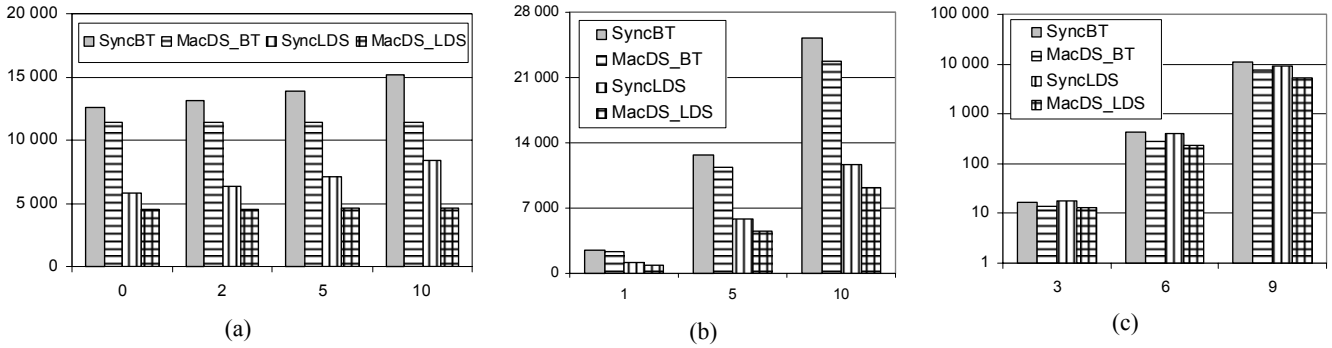

Figure 4: Expected time to get best solution, according to: (a) message transmission time [Card(S)=4;n=10;α=1;δ=0.7]*, (b) time to solve subproblem* [Cards(S)=4;n=10;τ=0;δ=0.7] *and (c) total number of subproblems* [n=3;α=1;τ=0;δ=0.7]

solutions.

These experiments were done in a distributed environment where each agent runs on a different computer. It allows measuring the real impact of concurrency in a situation where subproblems are hard and take a different amount of time to be solved. The difference in computation time between subproblems can vary five fold. Production of alternative solutions varies from a few seconds to several minutes. In this context, communication complexity is not an issue (Lynch 1996).

For a given case, the first global solution of each compared algorithm is always the same. It is also this solution that would be obtained by the companies using standard business process. Consequently, we compared the algorithms according to the percentage of reduction of the objective function they achieved, with respect to the computation time (in seconds). Figure 6 illustrates the results for the four industrial cases studied (a,b,c,d). For a very short computation time, SyncBT and MacDS_LDS give comparable results. This is because they produce the same solutions until the last agent receives a second task in its list. Then, MacDS_LDS starts to outperform SyncBT in a significant manner: while SyncBT persists in exploring only minor variations of the first solutions, MacDS_LDS explores different areas of the search tree. Case (b) is an exception: SyncBT is the winner by 0.5% for pretty much any computation time.

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

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

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

Conclusion

The good performance of classic algorithms for DCOP is based on the ability of each agent to locally compute a good bound on the global objective and make its decisions based on this bound. In a heterogeneous context, where specialized agents are facing different complex subproblems, agents may lack such global vision. Hierarchical problems are such a context.

For those situations, we proposed MacDS, a distributed algorithm that performs discrepancy-based search and allows the agents to work concurrently. It outperforms

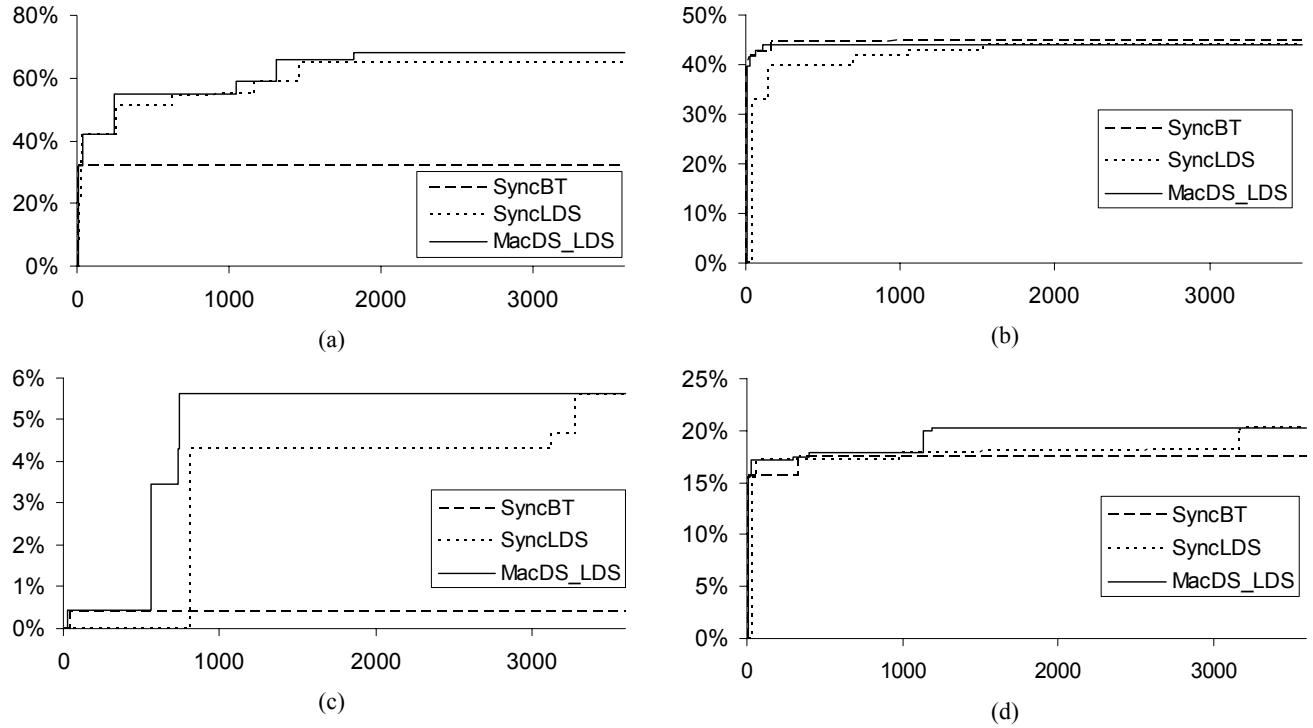

Figure 6: Reduction of the objective function, according to computation time (in seconds) for cases (a), (b), (c) and (d).

basic distributed search algorithms by two ways: (1) by applying backtracking policies based on discrepancies and (2) by allowing the agents to work concurrently, which reduces idle time for the agents.

We demonstrated the impact of different methods by applying them to real industrial problems. They all show considerable reduction of lateness for a real network of companies, MacDS being the winner most of the time. For an equal solution quality, MacDS also shows considerable reduction of computation time, when compared to a non-concurrent synchronous algorithm applying the same backtracking policy (LDS).

In the near future, it would be interesting to measure if discrepancy-based search could be helpful in situations where bound computation is possible. Discrepancy-based backtracking coupled with pruning based on bounds has shown good results in a centralized environment (Beck and Perron 2000).

References

Beck, J.C., Perron, L. 2000. Discrepancy-Bounded Depth First Search. *Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems*, 7-17. Paderborn, Germany.

Gershman, A., Meisels, A., Zivan, R. 2006. Asynchronous Forward-Bounding for Distributed Constraints Optimization. *European Conference on Artificial Intelligence*, 137-139. Amsterdam: IOS Press.

Harvey, W.D., Ginsberg, M.L. 1995. Limited discrepancy search. *International Joint Conference on Artificial Intelligence*, 607-613. Montreal, Can: Morgan Kaufmann.

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

Horling, B., Lesser, V. 2004. A survey of multi-agent organizational paradigms. *Knowledge Engineering Review*. 19(4): 281-316.

Le Pape, C., Baptiste, P. 1999. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*. 5(3): 305-325.

Lynch, N.A. 1996. *Distributed algorithms*. San Francisco, Calif.: Morgan Kaufmann.

Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M. 2005. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*. 161(1-2): 149-180.

Modi, P.J., Veloso, M. 2005. Bumping strategies for the multiagent agreement problem. *International Conference on Autonomous Agents and Multiagent Systems*, 527-533. New York: ACM Press.

Moyaux, T., Chaib-draa, B., D'Amours, S. 2006. Supply Chain Management and Multiagent Systems: An Overview. In: *Multiagent-Based Supply Chain Management*. Chaib-draa, B. and Müller, J.P. eds. New York : Springer.

Petcu, A., Faltings, B. 2005. DPOP: A Scalable Method for Multiagent Constraint Optimization. *International Joint Conference on Artificial Intelligence*. Edinburg, Scotland.

SCHNEEWEISS, C. 2003. *Distributed Decision Making*. New York: Springer.

Sun, J., Zhang, Y.F., Nee, A.Y.C. 2001. A distributed multi-agent environment for product design and manufacturing planning. *International Journal of Production Research*. 39(4): 625-645.

Walsh, T. 1997. Depth-bounded discrepancy search. *International Joint Conference on Artificial Intelligence*, 1388-1393. Nagoya, Japan: Morgan Kaufmann.

Yokoo, M., Ishida, T., Durfee, E.H., Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. *International Conference on Distributed Computing Systems*, 614-21. Yokohama, Japan: IEEE Press.

Zivan, R., Meisels, A. 2006. Concurrent search for distributed CSPs. *Artificial Intelligence*. 170(4-5): 440-61.