Integration of ACO in a Constraint Programming Language

Madjid Khichane, Patrick Albert, Christine Solnon

Integration of ACO in a Constraint Programming Language

Madjid Khichane[1,2], Patrick Albert[1], and Christine Solnon[2]

[1] ILOG SA, 9 rue de Verdun, 94253 Gentilly cedex, France
{mkhichane,palbert}@ilog.fr
[2] LIRIS CNRS UMR 5205, University of Lyon I
Nautibus, 43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
christine.solnon@liris.cnrs.fr

Abstract. We propose to integrate ACO in a Constraint Programming (CP) language. Basically, we use the CP language to describe the problem to solve by means of constraints and we use the CP propagation engine to reduce the search space and check constraint satisfaction; however, the classical backtrack search of CP is replaced by an ACO search. We report first experimental results on the car sequencing problem and compare different pheromone strategies for this problem.

1 Introduction

Our motivations mainly come from the two following observations:

- Ant Colony Optimization (ACO) has been successfully applied to a wide range of combinatorial optimization problems [1]; however most works have focused on designing efficient ACO algorithms for solving specific problems, but not on integrating these algorithms within declarative languages so that solving a new problem with this approach usually implies a lot of procedural programming;
- Constraint Programming (CP) languages provide high level features to declaratively model problems by means of constraints; however, most CP solvers are based on a systematic "Branch and Propagate" exploration of the search space, and fail to solve some hard problems within a reasonable time limit.

Hence, we investigate the integration of ACO within a CP language. Our research is based upon ILOG Solver [2], and we use its modeling language to describe the problem to solve by means of constraints and its propagation engine to reduce the search space and check constraint satisfaction; however, the search of solutions is guided by ACO. This approach has the benefit of reusing all the work done by ILOG at the modeling level as well as the code dedicated to constraint propagation and verification. We can as well test different variations of our ideas on a large benchmark library. Note that this work could be easily extended to other CP languages, such as, e.g., CHOCO or GECODE.

It is worth reporting that an hybridization of ACO and CP has already been proposed in [3] to solve a timetabling problem which contains hard constraints,

that must be satisfied, and has an objective function to optimize. In this approach, constraint propagation is used to build feasible solutions that satisfy all hard constraints while ACO is used to find high quality solutions with respect to the objective function. Our approach in this paper is rather different as ACO is used to guide a search procedure aiming at satisfying all the constraints.

The paper is organized as follows. In the next section, we briefly recall some definitions and terminology about CP. Section 3 describes the basic *Ant-CP* algorithm for solving constraint satisfaction problems. Section 4 shows how this algorithm may be used to solve the car sequencing problem and introduces different pheromone strategies and different heuristics for this problem. Section 5 experimentally compares these different variants of *Ant-CP*.

2 Background

A Constraint Satisfaction Problem (CSP) [4] is defined by a triple (X, D, C) such that X is a finite set of variables, D is a function that maps every variable $x_i \in X$ to its domain $D(x_i)$, that is, the finite set of values that can be assigned to x_i, and C is a set of constraints, that is, relations between some variables which restrict the set of values that can be assigned simultaneously to these variables.

Solving a CSP involves assigning values to variables so that constraints are satisfied. More formally, an *assignment* is a set of variable-value couples, noted $\langle x_i, v \rangle$ and corresponding to the assignment of a value $v \in D(x_i)$ to a variable x_i. The variables assigned in an assignment $\mathcal{A}$ are denoted by $var(\mathcal{A})$. An assignment $\mathcal{A}$ is *partial* if some variables are not assigned in $\mathcal{A}$, i.e., $var(\mathcal{A}) \subset X$; it is *complete* if all variables are assigned, i.e., $var(\mathcal{A}) = X$. An assignment $\mathcal{A}$ is *consistent* if it does not violate any constraint. A *solution of a CSP* (X, D, C) is a complete and consistent assignment.

CSPs are solved in a generic way by constraint solvers which are embedded within CP languages. These constraint solvers are usually based on a systematic exploration of the search space: starting from an empty assignment, they incrementally extend a partial consistent assignment by choosing a non assigned variable and a consistent value for it until either the current assignment is complete (a solution has been found) or the current assignment cannot be extended without violating constraints (the search must backtrack to a previous choice point and try another extension). To reduce the search space, this exhaustive exploration of the search space is combined with constraint propagation techniques: each time a variable is assigned to a value, constraints are propagated to filter the domains of the variables that are not yet assigned, i.e., to remove values that are not consistent with respect to the current assignment. If constraint propagation detects an inconsistency or if it removes all values from a domain, the search must backtrack. Different levels of consistency may be considered (e.g., node consistency or arc consistency); some consistencies are stronger than others, removing more values from the domains, but have also higher time complexities.

Algorithm 1: *Ant-CP* procedure

```
   Input: A CSP (X, D, C), a pheromone strategy Φ, a heuristic factor η
   Output: A consistent (partial or complete) assignment for (X, D, C)
 1 Initialize all pheromone trails of Φ to τ_max
 2 repeat
 3     foreach k in 1..nbAnts do
           /* Construction of a consistent assignment A_k                */
 4         A_k ← ∅
 5         repeat
 6             Select a variable x_i ∈ X so that x_i ∉ var(A_k)
 7             Choose a value v ∈ D(x_i)
 8             Add ⟨x_i, v⟩ to A_k
 9             Propagate constraints to filter domains of D
10         until var(A_k) = X or Failure ;
11     Update pheromone trails of Φ using {A_1, ..., A_nbAnts}
12 until var(A_i) = X for some i ∈ {1..nbAnts} or max cycles reached ;
13 return the largest constructed assignment
```

3 Description of *Ant-CP*

Some ACO algorithms have been previously proposed for solving CSPs [5–8]. In these algorithms, ants iteratively build complete assignments (that assign a value to every variable) that may violate constraints, and their goal is to minimize the number of constraint violations; a solution is found when the number of constraint violations is null.

In this paper, we investigate a new ACO framework for solving CSPs: ants iteratively build partial assignments (such that some variables may not be assigned to a value) that do not violate constraints, and their goal is to maximize the number of assigned variables; a solution is found when all variables are assigned. This new ACO framework may be combined with the propagation engine of ILOG Solver in a very straightforward way.

More precisely, the proposed algorithm for solving CSPs, called *Ant-CP*, is sketched in Algorithm 1. First, pheromone trails are initialized to some given value τ_{max}. Then, at each cycle (lines 2-12), each ant k constructs a consistent assignment $\mathcal{A}_k$ (lines 4-10): starting from an empty assignment, the ant iteratively chooses a variable which is not yet assigned and a value to assign to this variable; this variable assignment is added to $\mathcal{A}_k$, and constraints are triggered which might in turn narrow the domains of non assigned variables, trigger new assignments, or detect a failure; this process is iterated until either all variables have been assigned (i.e., a solution has been found) or the propagation step detects a failure. Once every ant has constructed an assignment, pheromone trails are updated. The algorithm stops iterating either when an ant has found a solution, or when a maximum number of cycles has been performed.

In the next paragraphs, we define the pheromone strategy Φ used to bias the search, and we describe the variable selection, value selection, propagation and pheromone updating steps.

Pheromone strategy. The pheromone strategy, denoted by Φ, is a parameter of *Ant-CP* and is defined by a triple $\Phi = (S, \tau, comp)$ such that:

- S is the set of components on which ants lay pheromone;
- τ is a function which defines how pheromone trails of S are used to bias the search. More precisely, given a partial assignment $\mathcal{A}$, a variable $x_i \notin var(\mathcal{A})$, and a value $v \in D(x_i)$, the function $\tau(\mathcal{A}, x_i, v)$ returns the value of the pheromone factor which evaluates the learnt desirability of adding $\langle x_i, v\rangle$ to the partial assignment $\mathcal{A}$;
- *comp* is a function which defines the set of components on which pheromone is laid when rewarding an assignment $\mathcal{A}$, i.e., the function $comp(\mathcal{A})$ returns the set of components associated with $\mathcal{A}$.

The goal of the pheromone strategy is to learn from previous constructions which decisions have allowed ants to build good assignments, and to use this information to bias further constructions. The default pheromone strategy, denoted by $\Phi_{default}$, is defined as follows:

- ants lay pheromone on variable-value couples, i.e.,

$$S = \{\tau_{\langle x_i, v\rangle} | x_i \in X, v \in D(x_i)\}$$

  so that each pheromone trail $\tau_{\langle x_i, v\rangle}$ represents the learnt desirability of assigning value v to x_i;
- the pheromone factor is defined by $\tau(\mathcal{A}, x_i, v) = \tau_{\langle x_i, v\rangle}$;
- the set of components associated with an assignment is

$$comp(\mathcal{A}) = \{\tau_{\langle x_i, v\rangle} | \langle x_i, v\rangle \in \mathcal{A}\}$$

For specific problems, the user may design other pheromone strategies. In this case, he must define the triple $(S, \tau, comp)$. We shall propose and compare two other pheromone strategies for the car sequencing problem in the next section.

Selection of a variable. When constructing an assignment, the order in which the variables are assigned is rather important and variable ordering heuristics have been studied widely in the context of backtrach search [4, 9]. These heuristics can be used as well in our context of greedy construction of assignments. For example, we can use the different variable ordering heuristics that are predefined in ILOG solver such as, e.g., the `IloChooseMinSizeInt` heuristic which selects the variable that has the smallest domain.

Choice of a value. Once a variable x_i has been selected, ants have to choose a value v in the domain $D(x_i)$ of x_i. Note that this domain may have been reduced by constraint propagation and may not contain all values of the initial domain of x_i. The main contribution of ACO for solving CSPs is to provide a generic heuristic for choosing values. The value v to be assigned to a variable x_i is randomly chosen within $D(x_i)$ with respect to a probability $p(x_i, v)$ which depends on a pheromone factor $\tau(\mathcal{A}, x_i, v)$ —which reflects the past experience of the colony regarding the addition of $\langle x_i, v\rangle$ to the partial assignment $\mathcal{A}$— and a heuristic factor $\eta(\mathcal{A}, x_i, v)$ —which is problem-dependent, i.e.,

$$p(x_i, v) = \frac{[\tau(\mathcal{A}, x_i, v)]^\alpha [\eta(\mathcal{A}, x_i, v)]^\beta}{\sum_{w \in D(x_i)} [\tau(\mathcal{A}, x_i, w)]^\alpha [\eta(\mathcal{A}, x_i, w)]^\beta} \quad (1)$$

where α and β are two parameters that determine the relative weights of pheromone and heuristic information.

Constraint propagation. Each time a variable is assigned to a value, a propagation algorithm is called. This algorithm filters the domains of the non assigned variables: it removes the values that are inconsistent with respect to some partial consistencies such as, e.g., node-consistency, arc-consistency or path-consistency [4]. If the domain of a variable becomes a singleton, then the partial assignment $\mathcal{A}_k$ is completed by the assignment of this variable and the propagation process is continued. If the domain of a variable becomes empty, then *Failure* is returned.

One may consider different propagation algorithms, that ensure different partial consistencies. In our preliminary experiments, we have used the default propagation algorithm integrated to ILOG solver.

Pheromone updating step. Once every ant has constructed an assignment, pheromone trails are updated according to ACO: first, they are decreased by multiplying them by $(1-\rho)$ (where $\rho \in [0; 1]$ is the evaporation rate); then, they are rewarded with respect to their contribution to the construction of good assignments. More precisely, let *BestOfCycle* be the set of all the best assignments constructed during the cycle, that is,

$$BestOfCycle = \{\mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\} \mid \#var(\mathcal{A}_i) \text{ is maximal}\}$$

For each assignment $\mathcal{A}_k \in BestOfCycle$, pheromone trails associated with pheromone components of $\mathcal{A}_k$ are increased by a quantity δ_τ which is proportionally inverse to the gap of sizes between $\mathcal{A}_k$ and the largest assignment $\mathcal{A}_{best}$ built since the beginning of the search (including the current cycle), i.e.,

$$\delta_\tau = 1/(1 + \#\mathcal{A}_{best} - \#\mathcal{A}_k)$$

The set of pheromone components associated with a given assignment depends on the considered pheromone strategy and is defined by *comp*.

Note that *Ant-CP* follows the MAX-MIN Ant System scheme [10] so that pheromone trails are bounded between two given bounds τ_{min} and τ_{max}.

4 Using *Ant-CP* to solve the Car Sequencing Problem

The car sequencing problem involves scheduling cars along an assembly line in order to install options (e.g., sun-roof or air-conditioning) on them. Each option is installed by a different station, designed to handle at most a certain percentage of the cars passing along the assembly line, and the cars requiring this option must be spaced so that the capacity of the station is never exceeded. More precisely, a car sequencing problem is defined by a tuple (C, O, p, q, r), where C is the set of cars to be produced and O is the set of different options. The two functions $p : O \rightarrow \mathbb{N}$ and $q : O \rightarrow \mathbb{N}$ define the capacity constraint associated with each option $o_i \in O$, i.e., for any sequence of $q(o_i)$ consecutive cars on the line, at most $p(o_i)$ of them may require o_i. The function $r : C \times O \rightarrow \{0, 1\}$ defines option requirements, i.e., for each car $c_i \in C$ and for each option $o_j \in O$, $r(c_i, o_j)$ returns 1 if o_j must be installed on c_i, and 0 otherwise.

Solving a car sequencing problem involves finding an arrangement of the cars in a sequence, defining the order in which they will pass along the assembly line, such that the capacity constraints are met. This problem is NP-hard [11]. A more general problem –which introduces paint batching constraints and priority levels for capacity constraints– has been proposed by Renault for the ROADEF challenge in 2005 [12].

4.1 CP model

The car sequencing problem has been first introduced in the CP community in 1988 [13]. Since then, it has been very often used to evaluate CP solvers and it is the first problem of the CSP library *CSPlib* [14]. To evaluate *Ant-CP*, we have considered a classical CP model for the car sequencing problem which basically corresponds to the first model described in the user's manual of ILOG solver. In order to reduce the search space, this model introduces the concept of car classes: all cars requiring a same set of options are grouped into a same car class.

There are two different kinds of variables:

- A slot variable x_i is associated with each position i in the sequence of cars. This variable corresponds to the class of the ith car in the sequence and its domain is the set of all car classes.
- An option variable y_i^j is associated with each position i in the sequence and each option j. This variable is assigned to 1 if option j has to be installed on the ith car of the sequence, and 0 otherwise, so that its domain is $\{0, 1\}$.

There are three different kinds of constraints:

- Link constraints specify the link between slot and option variables, i.e., $y_i^j = 1$ iff option j has to be installed on x_i.
- Capacity constraints specify that station capacities must not be exceeded, i.e., for each option j and each subsequence of q_j cars, a linear inequality specifies that the sum of the corresponding option variables must be smaller or equal to p_j.
- Demand constraints specify, for each car class, the number of cars of this class that must be sequenced.

4.2 Variable ordering heuristic

In experiments reported in Section 5 we have used a classical sequential variable ordering heuristic, which consists in assigning slot variables in the order defined by the sequence of cars, i.e., $x_1, x_2, x_3, \ldots$ Note that option variables y_i^j are assigned by propagation when the corresponding slot variable x_i is assigned.

4.3 Pheromone strategies

As pheromone is at the core of the efficiency of any ACO implementation, we explore the impact of its structure: besides the default pheromone strategy $\Phi_{default}$ (which associates a trail with every couple (x_i, j) such that x_i is the variable associated with position i and j is a car class), we consider two other strategies which will be experimentally compared in the next section.

Pheromone strategy $\boldsymbol{\Phi_{classes}}$. This pheromone strategy has been introduced in [15]. The set S associates a trail $\tau_{(v,w)}$ with every couple of car classes (v, w). This pheromone trail represents the learnt desirability of sequencing a car of class w just after a car of class v or, in other words, of assigning the value w to a variable x_i when x_{i-1} has just been assigned to v.

For this pheromone strategy, the pheromone factor is equal to the pheromone trail between the last assigned car class and the candidate car class (this factor is equal to one when assigning the first variable), i.e., if $i > 1$ and $\langle x_{i-1}, w\rangle \in \mathcal{A}$, then $\tau(\mathcal{A}, x_i, v) = \tau_{(w,v)}$, otherwise $\tau(\mathcal{A}, x_i, v) = 1$.

When updating pheromone trails, pheromone is laid on couples of consecutively assigned values, i.e., $comp(\mathcal{A}) = \{\tau_{(v,w)} | \{\langle x_i, v\rangle, \langle x_{i+1}, w\rangle\} \subseteq \mathcal{A}\}$.

Pheromone strategy $\boldsymbol{\Phi_{cars}}$. This pheromone strategy has been introduced in [8]. The set S associates a trail $\tau_{(v,j,w,k)}$ with each couple of car classes (v, w) and each $j \in [1; \#v]$ and $k \in [1; \#w]$ where $\#v$ and $\#w$ are the number of cars within the classes v and w respectively. This trail represents the learnt desirability of sequencing the k^{th} car of class w just after the j^{th} car of class v.

In this case, the pheromone factor $\tau(\mathcal{A}, x_i, v)$ is equal to 1 for the first car, i.e., $\tau(\mathcal{A}, x_i, v) = 1$ if $i = 1$. Otherwise $\tau(\mathcal{A}, x_i, v) = \tau_{(w,j,v,k+1)}$ where w is the value assigned to x_{i-1} in $\mathcal{A}$, j is the number of variables assigned to w in $\mathcal{A}$, and k is the number of variables assigned to v in $\mathcal{A}$.

When updating pheromone trails, pheromone is laid on couples of consecutively sequenced cars, i.e.,

$$comp(\mathcal{A}) = \{\tau_{(v,j,w,k+1)} |\ \{\langle x_l, v\rangle, \langle x_{l+1}, w\rangle\} \subseteq \mathcal{A} \text{ and } j = \#\{\langle x_m, v\rangle | m \leq l\} \text{ and } k = \#\{\langle x_m, w\rangle | m \leq l\}\}$$

4.4 Heuristic factors for the car sequencing problem

In the transition probability defined by eq. (1), the pheromone factor $\tau(\mathcal{A}, x_i, v)$ —which represents the past experience of the colony— is combined with a heuristic factor $\eta(\mathcal{A}, x_i, v)$ —which is problem-dependent. We now introduce two different problem-dependent heuristics for the car sequencing problem; these heuristics will be compared in Section 5.

Dynamic Sum of Utilisation rates (DSU). Smith [9] has introduced value ordering heuristics based on option utilization rates: the utilization rate of an option i is the ratio of the number of cars requiring i with respect to the maximum number of cars in the sequence which could have i while satisfying its capacity constraint. Gotlieb et al. [16] have compared different value ordering heuristics based on option utilization rates, and have shown that one of the best performing heuristics is the so-called Dynamic Sum of Utilization rates (DSU), i.e.,

$$\eta(\mathcal{A}, x_i, v) = \sum_{o_j \in \text{reqOptions}(v)} \frac{\text{reqSlots}(o_j, n_j)}{N}$$

where reqOptions(v) is the set of options that are required by cars of class v, N is the number of cars that have not yet been sequenced in $\mathcal{A}$, n_j is the number of cars that require option o_j and have not yet been sequenced in $\mathcal{A}$, and reqSlots(o_j, n_j) is a lower bound of the number of slots needed to sequence n_j cars that require option o_j without violating capacity constraints. For defining reqSlots(o_j, n_j), we have considered the formula introduced in [17], i.e., if $n_j \ \% \ p_j = 0$ then $\text{reqSlots}(o_j, n_j) = q_j * n_j/p_j - (q_j - p_j)$, otherwise $\text{reqSlots}(o_j, n_j) = q_j * (n_j - n_j \ \% \ p_j)/p_j + n_j \ \% \ p_j$.

Dynamic Sum of Utilisation rates with Propagation (DSU+P) We can exploit utilization rates to detect inconsistencies and filter variable domains:

- when the utilization rate of an option becomes greater than 1 (i.e., when $reqSlots(o_j, n_j)$ becomes greater than N) one can conclude that it is not possible to complete the sequence without violating constraints so that one can stop the current assignment construction on a failure;
- when the utilization rates of one or more options become equal to 1, we can remove from the domain of the next variable every car class which does not require all options that have an utilization rate equal to 1.

5 Experimental results

Test suite. Satisfiable instances of the library *CSPlib* [14] with 100 cars (4 instances described in [18]) and 200 cars (70 instances described in [19]) are all easily solved by *Ant-CP*. Hence, we consider a harder test suite which is described in [20]. All instances have 8 options and 20 car classes; capacity constraints are

randomly generated in such a way that $\forall o_i \in O, 1 \leq p(o_i) \leq 3$ and $p(o_i) < q(o_i) \leq p(o_i)+2$. This test suite contains 32 instances with 100 cars, 21 instances with 300 cars and 29 instances with 500 cars. All these instances are satisfiable.

Considered *Ant-CP* instanciations. We compare the three pheromone strategies $\Phi_{default}$, $\Phi_{classes}$, and Φ_{cars}. To evaluate the influence of the pheromone on the solution process, we also consider a strategy without pheromone, denoted by $\Phi_{\emptyset}$: in this case, the set S is the empty set and the pheromone factor $\tau(\mathcal{A}, x_k, v)$ is set to 1 so that *Ant-CP* behaves like a greedy randomized approach which chooses values with respect to the heuristic factor only.

We also compare the two heuristic factors DSU and DSU+P. We note *Ant-CP*(Φ, h) the *Ant-CP* instanciation obtained with the pheromone strategy $\Phi \in \{\Phi_{\emptyset}, \Phi_{classes}, \Phi_{cars}, \Phi_{default}\}$ and the heuristic $h \in \{$DSU,DSU+P$\}$.

Parameter setting. Parameters have been set as follows: $\alpha = 1$, $\beta = 6$, $\rho = 0.02$, $nbAnts$=30, $\tau_{min} = 0.01$, and $\tau_{max} = 4$.

Comparison of *Ant-CP* instanciations. Fig. 1 compares the four different pheromone strategies when using the DSU heuristic (upper curves) and then when using the DSU+P heuristic (lower curves). One notes that after 3000 cycles, using pheromone increases the success rate from 66.32% for *Ant-CP*($\Phi_{\emptyset}$,DSU) to 79.39% for *Ant-CP*(Φ_{cars},DSU), 72.56% for *Ant-CP*($\Phi_{default}$,DSU) and 68.29% for *Ant-CP*($\Phi_{classes}$,DSU). Hence, on these instances, the best pheromone strategy is Φ_{cars}; the default pheromone strategy is worse than Φ_{cars}, but better than $\Phi_{classes}$.

The DSU+P heuristic usually obtains better results than DSU. However, the improvement depends on the considered pheromone strategy: it is not significant when pheromone is ignored (the success rate of $\Phi_{\emptyset}$ is increased from 66.32% to 66.97%); it is rather important for the pheromone strategies $\Phi_{classes}$ and $\Phi_{default}$ (success rates are increased from 68.29% to 74.39% for $\Phi_{classes}$ and from 72.56% to 78.54% for $\Phi_{default}$); it is not as important for Φ_{cars} (the success rate is increased from 79.39% to 82.32%).

Let us finally note that, given a heuristic, the four variants spend nearly the same CPU time to perform one cycle, as most of the time is spent by propagation procedures. However, cycles are performed quicker with the DSU+P heuristic than with the DSU heuristic. Indeed, DSU+P filters variable domains and detects earlier some inconsistencies. Hence, for the instances with 100 cars, 3000 cycles are performed in 5 minutes with the DSU heuristic whereas they are performed in 3 minutes with the DSU+P heuristic (on a 2GHz Intel Core Duo).

6 Conclusion

These first experiments on the Car Sequencing problem show that integrating an ACO search might be designed as a simple extension of a modular CP language

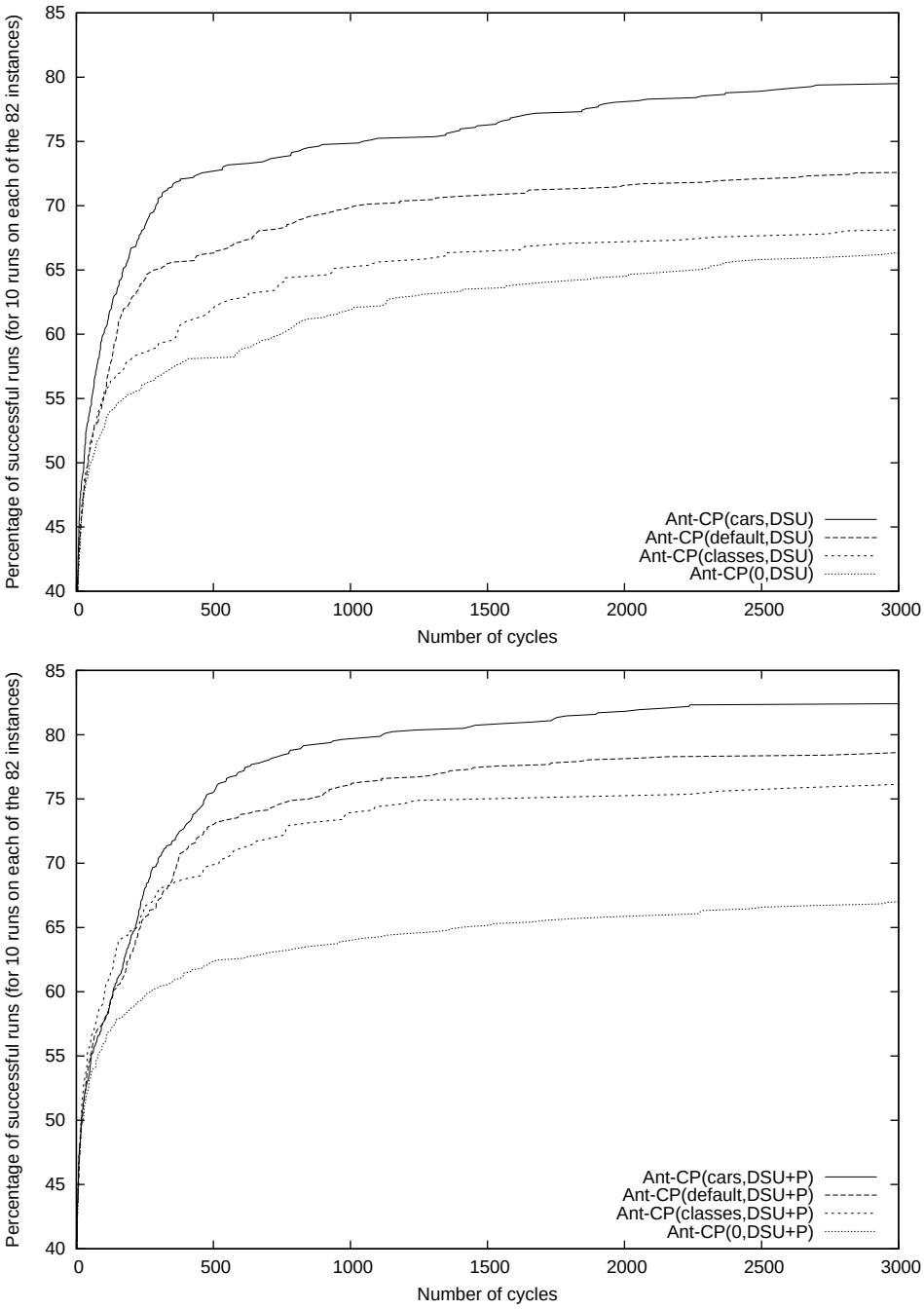

Fig. 1. Comparison of different instanciations of *Ant-CP*(Φ, h), with $\Phi \in \{\Phi_{default}, \Phi_{classes}, \Phi_{cars}, \Phi_{\emptyset}\}$ and $h \in \{DSU, DSU + P\}$: each curve plots the evolution of the percentage of runs that have found a solution with respect to the number of cycles (for 10 runs for each of the 82 instances).

such as ILOG Solver. It is worth noting that thanks to the modular nature of ILOG Solver that separates the modeling of the problem from the computation of its solution, the CP model used to describe the sequencing problem does not depend on the search technique: one can try or combine other search methods without changing the problem description.

First experimental results are very promising. Indeed, classical CP solvers based on exhaustive tree search approaches are still not able to solve within a reasonable amount of time all instances of *CSPLib* with 100 and 200 cars, even when using dedicated filtering algorithms such as, e.g., those proposed in [18, 21, 22]. All these instances are easily solved by *Ant-CP* (whatever the pheromone strategy is): the 70 instances with 200 cars (resp. 4 instances with 100 cars) are solved in less than a second (resp. less than a minute). As a comparison, filterings introduced in [22] for the "sequence" constraint can solve less than half of these instances in less than 100 seconds when they are combined with the default tree search of ILOG Solver on a Pentium 4 sequenced at 3.2 Ghz. Of course, these experiments should be pursued on other problems to further validate our approach.

These first results might be further enhanced by adapting the propagation algorithms to the specificities of ACO. Indeed, propagation algorithms integrated in most CP solvers have been designed to fit a procedure that enables backtracking on the choice points. Such algorithms have thus to maintain data structures enabling the restitution at each backtrack of the context of the previous choice point. The allocation and management of these data structures bring a cost both in terms of memory and Cpu time which is necessary in the context of a backtrack-based search procedure but not in the context of our ACO inspired search method which never backtracks.

A similar remark might be done with respect to a language such as Comet that does not rely on a backtracking tree search but on local search. Indeed, Van Hentenryck and Michel have shown in [23] that the Comet language may be used to implement an ACO algorithm in a very declarative way. However, Comet is dedicated to local search which explores the search space by iteratively applying elementary moves to a current configuration. In order to efficiently select the best move to be applied, Comet maintains a set of data structures that support the incremental evaluation of invariant properties after each elementary move. These data structures can be used to support an ACO search (they support the incremental evaluation of the heuristic factor at each step of the construction) but again, they maintain more information than necessary because choices made in ACO during the greedy construction of the solution are never revised.

Still, we did not explore in both cases (tree-search based solver such as ILOG Solver, or local-search based solver such as Comet) how well the cost of extra data structures could be compensated by an effective combination of the ACO search procedure and the default search of the host solver. For example one could do a limited tree-search based exploration of the remaining search space when a constraint triggers a failure in ILOG Solver, or perform some local search at the end of the exploration of each or some of the artificial ants.

References

1. Dorigo, M., Stuetzle, T.: Ant Colony Optimization. MIT Press (2004)
2. ILOG: Ilog solver user's manual. Technical report, ILOG (1998)
3. Meyer, B., Ernst, A.: Integrating aco and constraint propagation. In: ANTS'04. Number 3172 in LNCS, Springer (2004) 166–177
4. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, London, UK (1993)
5. Schoofs, L., Naudts, B.: Solving csps with ant colonies. In: ANTS. (2000)
6. Roli, A., Blum, C., Dorigo, M.: Aco for maximal constraint satisfaction problems. In: Meta–heuristics International Conference (MIC). (2001)
7. Solnon, C.: Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation **6**(4) (2002) 347–357
8. Solnon, C.: Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. EJOR (to appear) (2008)
9. Smith, B.: Succeed-first or fail-first: A case study in variable and value ordering heuristics. In: third Conference on the Practical Applications of Constraint Technology PACT'97. (1996) 321–330
10. Stützle, T., Hoos, H.: MAX-MIN Ant System. Journal of Future Generation Computer Systems, special issue on Ant Algorithms **16** (2000) 889–914
11. Kis, T.: On the complexity of the car sequencing problem. Operations Research Letters **32** (2004) 331–335
12. Solnon, C., Cung, V., Nguyen, A., Artigues, C.: The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. EJOR (to appear) (2008)
13. Dincbas, M., Simonis, H., van Hentenryck, P.: Solving the car-sequencing problem in constraint logic programming. In Kodratoff, Y., ed.: Proceedings of ECAI-88. (1988) 290–295
14. Gent, I., Walsh, T.: Csplib: a benchmark library for constraints. Technical report (1999) available from http://csplib.cs.strath.ac.uk/.
15. Gravel, M., Gagné, C., Price, W.: Review and comparison of three methods for the solution of the car-sequencing problem. JORS (2004)
16. Gottlieb, J., Puchta, M., Solnon, C.: A study of greedy, local search and aco approaches for car sequencing problems. In: EvoCOP. LNCS 2611, Springer (2003)
17. Boysen, N., Fliedner, M.: Comments on solving real car sequencing problems with ant colony optimization . EJOR **182**(1) (2007) 466–468
18. Regin, J.C., Puget, J.F.: A filtering algorithm for global sequencing constraints. In: CP97. Volume 1330 of LNCS. Springer-Verlag (1997) 32–46
19. Lee, J., Leung, H., Won, H.: Performance of a comprehensive and efficient constraint library using local search. In: 11th Australian JCAI. LNAI. Springer-Verlag (1998)
20. Perron, L., Shaw, P.: Combining forces to solve the car sequencing problem. In: Proceedings of CP-AI-OR'2004. Volume 3011 of LNCS., Springer (2004) 225–239
21. van Hoeve, W.J., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the sequence constraint. In: CP 2006. LNCS 4204, Springer (2006) 620–634
22. Brand, S., Narodytska, N., Quimper, C.G., Stuckey, P.J., Walsh, T.: Encodings of the sequence constraint. In: CP. Volume 4741 of LNCS., Springer (2007) 210–224
23. Hentenryck, P.V., Michel, L.: Constraint-based local search. MIT Press (2005)