Strong Combination of Ant Colony Optimization with Constraint Programming Optimization

Madjid Khichane, Patrick Albert, Christine Solnon

Strong Combination of Ant Colony Optimization with Constraint Programming Optimization

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

[1] IBM
9, rue de Verdun, Gentilly 94253, France
{madjid.khichane,palbert}@fr.ibm.com
[2] Université de Lyon
Université Lyon 1, LIRIS CNRS UMR5205, France
christine.solnon@liris.cnrs.fr

Abstract. We introduce an approach which combines ACO (Ant Colony Optimization) and IBM ILOG CP Optimizer for solving COPs (Combinatorial Optimization Problems). The problem is modeled using the CP Optimizer modeling API. Then, it is solved in a generic way by a two-phase algorithm. The first phase aims at creating a hot start for the second: it samples the solution space and applies reinforcement learning techniques as implemented in ACO to create pheromone trails. During the second phase, CP Optimizer performs a complete tree search guided by the pheromone trails previously accumulated. The first experimental results on knapsack, quadratic assignment and maximum independent set problems show that this new algorithm enhances the performance of CP Optimizer alone.

1 Introduction

Combinatorial Optimization Problems (COPs) are of high importance for the scientific world as well as for the industrial world. Most of these problems are *NP*-hard so that they cannot be solved exactly in polynomial time (unless $P = NP$). Examples of *NP*-hard COPs include timetabling, telecommunication network design, traveling salesman problems, Multi-dimensional Knapsack Problems (MKPs), and Quadratic Assignment Problems (QAPs). To solve COPs, two main dual approaches may be considered, i.e., Branch and Propagate and Bound (B&P&B) approaches, and metaheuristic approaches.

B&P&B approaches combine an exhaustive tree-based exploration of the search space with constraint propagation and bounding techniques which reduce the search space. These approaches ensure to find an optimal solution in bounded time. As a counterpart, they might need exponential computation time in the worst case [1, 2]. Constraint Programming (CP) is one of the most popular generic B&P&B approaches for solving COPs modeled by means of constraints:

⋆ This work has been partially financed by IBM ILOG under the research collaboration contract IBM ILOG/UCBL-LIRIS.

it offers high-level languages for modeling COPs and it integrates constraint propagation and search algorithms for solving them in a generic way. Hence, solving COPs with CP does not require a lot of programming work. CP is usually very effective when constraints are tight enough to eliminate large infeasible regions by propagating constraints. However, as it is generally based on B&P&B approach, it fail to found a high solution quality with an acceptable computational time limit.

Metaheuristics have shown to be very effective for solving many COPs. They explore the search space in an incomplete way and sacrifice optimality guarantees, but gets good solutions in reasonable computational times. However, solving a new problem with a metaheuristic usually requires a lot of programming work. In particular, handling constraints is not an easy task and requires designing appropriate incremental data structures for quickly evaluating constraint violations before making a decision.

Many metaheuristics are based on a local search framework such that the search space is explored by iteratively perturbing combinations. Among others, local search-based metaheuristics include simulated annealing [3], tabu search [4], iterated local search [5], and variable neighborhood search [6]. To ease the implementation of local search-based algorithms for solving COPs, Van Hentenryck and Michel have designed a high level constraint-based language, named Comet [7]. In particular, Comet introduces incremental variables, thus allowing the programmer to declaratively design data structures which are able to efficiently evaluate neighborhoods.

In this paper, we propose a generic approach for solving COPs which combines CP Optimizer —a B&P&B-based solver developed by IBM ILOG— with the Ant Colony Optimization (ACO) metaheuristic [8]. ACO is a constructive approach (and not a local search-based one): it explores the search space by iteratively constructing new combinations in a greedy randomized way. ACO has shown to be very effective to quickly find good solutions to COPs, but it suffers from the same drawbacks as other metaheuristics, i.e., there are no optimality guarantees and quite a lot of programming is required to solve new COPs with ACO.

By combining ACO with CP Optimizer, we take the best of both approaches. In particular, we use the CP Optimizer modeling API to describe the problem to solve. Hence, to solve a new COP with our approach, one only has to model the problem to solve by means of a set of constraints and an objective function to optimize, and then ask the solver to search for the optimal solution. This search is decomposed in two phases. In a first phase, ACO is used to sample the space of feasible solutions and gather useful information about the problem by means of pheromone trails. During this first phase, CP Optimizer is used to propagate constraints and provide feasible solutions to ACO, while in a second phase, it performs a complete B&P&B to search for the optimal solution. During this second phase, pheromone trails collected during the first phase are used by CP Optimizer as value ordering heuristics, allowing it to quickly focus on the most

promising areas of the space of feasible solutions. In both phases, we also use impacts [9] as an ordering heuristic.

Let us point out that our main objective is not to compete with state-of-the-art algorithms which are dedicated to solving specific problems, but to show that sampling the search space with ACO can significantly improve the solution process of a generic B&P&B approach for solving COPs. For this, we chose CP Optimizer as our reference.

The rest of this paper is organized as follows. In Section 2, we recall some definitions about COP, CP, and ACO. Section 3 describes the *CPO-ACO* algorithm. In section 4, we give some experimental results on the multidimensional knapsack problem, the quadratic assignment problem and the maximum independent set problem. We conclude with a discussion on some other related work and further work.

2 Background

2.1 COP

A COP is defined by a tuple $P = (X, D, C, F)$ such that $X = \{x_1, \ldots, x_n\}$ is a set of n decision variables; for every variable $x_i \in X$, $D(x_i)$ is a finite set of integer values defining the domain of x_i; C is a set of constraints; and $F : D(x_1) \times \ldots \times D(x_n) \longrightarrow \mathbb{R}$ is an objective function to optimize.

An assignment $\mathcal{A}$ is a set of variable-value couples denoted $< x_i, v_i >$ which correspond to the assignment of a value $v_i \in D(x_i)$ to a variable x_i. An assignment $\mathcal{A}$ is complete if all variables of X are assigned in $\mathcal{A}$; it is partial otherwise. An assignment is inconsistent if it violates a constraint and it is consistent otherwise. A feasible solution is a complete consistent assignment. A feasible solution $\mathcal{A}$ of P is optimal if for every other feasible solution $\mathcal{A}^{'}$ of P, $F(\mathcal{A}) \leq F(\mathcal{A}^{'})$ if P is a minimization problem or $F(\mathcal{A}) \geq F(\mathcal{A}^{'})$ if P is a maximization problem.

2.2 Complete B&P&B approaches

B&P&B approaches solve COPs by building search trees: at each node, one chooses a non-assigned variable x_i and, for each value $v_i \in D(x_i)$, one creates a new node corresponding to the assignment of x_i to v_i. This tree search is usually combined with constraint propagation and bounding techniques. Constraint propagation filters variable domains by removing inconsistent values with respect to some local consistency such as, for example, arc consistency. Bounding techniques compute bounds on the objective function and prune the nodes for which this approximation is worse than the best feasible solution found so far. When constraint propagation or bounding techniques detect a failure, one backtracks to the last choice point to explore another branch. This method is effective and generic, although it fails to solve some COPs for which constraint propagation and bounding techniques are not able to reduce the search space to a reasonable size.

Algorithm 1: Generic ACO framework for solving a COP (X, D, C, F)

```
1 Initialize pheromone trails
2 while Stopping criteria not reached do
3 |   for Each ant do Construct a complete assignment
4 |   Update pheromone trails
```

2.3 Impact-Based search strategies

In constraint programming, as soon as a value v_i is assigned to a variable x_i, constraint propagation removes part of the infeasible space by reducing the domains of some variables. Refalo [9] has defined the *impact* of the assignment $x_i = v_i$ as the proportion of search space removed. He has defined the impact of a value as the average of its observed impacts and the impact of a variable as the average of the impact of its remaining values. He has shown that these impacts may be used to define valuable ordering heuristics.

2.4 Ant Colony Optimization (*ACO*)

There exist two main kinds of heuristic approaches, i.e., perturbative and constructive approaches. Perturbative heuristic approaches (such as local search) explore the search space by iteratively perturbating combinations. Constructive heuristic approaches sample the search space by iteratively constructing combinations in a greedy randomized way: starting from an empty combination, combination components are iteratively added until the combination is complete. At each step of these constructions, the combination component to be added is randomly chosen with respect to some probability.

Ant Colony Optimization (ACO) [8] is a constructive heuristic approach which borrows features from the collective behavior of ants to define the probability of adding a component to a combination: this probability depends on a quantity of pheromone which represents the past experience with respect to the choice of this component.

Algorithm 1 describes the generic ACO framework: at each cycle, each ant builds an assignment in a greedy randomized way using pheromone trails to progressively bias probabilities with respect to previous constructions; then pheromone trails are updated. We describe the main steps of this algorithm in the next paragraph, with a focus on COPs described by means of a tuple (X, D, C, F) so that the goal is to find the best feasible assignment.

Pheromone trails: Pheromone is used to guide the search and a key point lies in the choice of the components on which pheromone is laid. When the COP is defined by a tuple (X, D, C, F), one may associate a pheromone trail $\tau(x_i, v_i)$ with every variable $x_i \in X$ and every value $v_i \in D(x_i)$. Intuitively, this pheromone trail represents the desirability of assigning x_i to v_i. Such a pheromone structure

has shown to be effective to solve, for example, QAPs [10], CSPs [11], and car sequencing problems [12].

Pheromone trails are used to intensify the search around the best assignments built so far. In order to balance intensification and diversification, Stützle and Hoos have proposed in [10] to bound pheromone trails between two parameters τ_{min} and τ_{max} so that the relative difference between pheromone trails is limited. Also, pheromone trails are initialized to τ_{max} at the beginning of an ACO search.

Construction of assignments by ants (line 3): Each assignment is built in a greedy randomized way: starting from an empty assignment, one iteratively chooses a non assigned variable and a value to assign to this variable, until all variables have been assigned. The next variable to assign is usually chosen with respect to some given ordering heuristic (e.g., in increasing order for the QAP or the car sequencing problem, or with respect to the min-domain heuristic for CSPs). Once a non assigned variable x_i has been chosen, the value $v_i \in D(x_i)$ to assign to x_i is chosen with respect to probability:

$$p(x_i, v_i) = \frac{[\tau(x_i, v_i)]^\alpha \cdot [\eta(x_i, v_i)]^\beta}{\sum_{v_j \in D(x_i)} [\tau(x_i, v_j)]^\alpha \cdot [\eta(x_i, v_j)]^\beta} \quad (1)$$

where $\eta(x_i, v_i)$ is the heuristic factor associated with the assignment of x_i to v_i. The definition of this factor depends on the considered application, and usually evaluates the impact of this assignment on the objective function. α and β are two parameters that allow the user to balance the influence of pheromone and heuristic factors in the transition probability.

The way constraints are handled may be different from a COP to another. For loosely constrained COPs, such as QAPs [10], maximum clique problems [13], or MKPs [14], constraints are propagated after each variable assignment in order to remove inconsistent values from the domains of non assigned variables, so that ants always build feasible solutions. However, when constraints are tighter so that it is actually difficult to build feasible solutions, constraint violations may be integrated in the heuristic factor and in the objective function so that ants may build inconsistent assignments [11].

Pheromone updating step (line 4): Once each ant has constructed an assignment, pheromone trails are updated. In a first step, all pheromone trails are decreased by multiplying them by a factor $(1 - \rho)$, where $\rho \in [0; 1]$ is the evaporation rate. This evaporation process allows ants to progressively forget older constructions and to emphasize more recent ones. In a second step, some assignments are rewarded by laying pheromone trails. These assignments may be the best of the cycle and/or the best since the beginning of the search. The goal is to increase the probability of selecting the components of these assignments during the next constructions. The pheromone is laid on the trails associated with the rewarded assignments. When pheromone trails are associated with variable/value couples, pheromone is laid on the variable/value couples of the assignment to reward. The quantity of pheromone laid usually is proportional to the quality of the rewarded

assignment. This quantity is often normalized between 0 and 1 by defining it as a ratio between the value of the assignment to reward and the optimal value (if it is known) or the best value found since the beginning of the search.

3 Description of $CPO - ACO$

ACO has shown to be very effective for quickly finding good solutions to many COPs. However, designing ACO algorithms for new COPs implies a lot of programming: if procedures for managing and exploiting pheromone are very similar from a COP to another so that one can easily reuse them, solving a new COP implies to write procedures for propagating and checking problem dependent constraints. Hence, a first motivation for combining ACO with CP is to reuse the numerous available procedures for managing constraints. Moreover, combining ACO with CP optimizer allows us to take the best of these two approaches:

- During a first phase, CP Optimizer is used to sample the space of feasible solutions, and pheromone trails are used to progressively intensify the search around the best feasible solutions.
- During a second phase, CP Optimizer is used to search for an optimal solution, and the pheromone trails collected during the first phase are used to guide CP Optimizer in this search.

3.1 First phase of $CPO - ACO$

Algorithm 2 describes the first phase of $CPO - ACO$, the main steps of which are described in the next paragraphs.

Pheromone structure: The pheromone structure is used in order to progressively intensify the search around the most promising areas, i.e., those that contain the best feasible solutions with respect to the objective function. This pheromone structure associates a pheromone trail $\tau(x_i, v_i)$ with each variable $x_i \in X$ and each value $v_i \in D(x_i)$. Each pheromone trail is bounded between two given bounds τ_{min} and τ_{max}, and is initialized to τ_{max} (line 1) as proposed in [10]. At the end of the first phase, the pheromone structure τ is returned so that it can be used in the second phase as a value ordering heuristic.

Construction of assignments: At each cycle (lines 2-14), each ant calls CP Optimizer in order to construct a feasible solution (line 4). Note that during this first phase, we do not ask CP Optimizer to optimize the objective function, but simply to find feasible solutions that satisfy all the constraints.

CP Optimizer is used as a black-box with its default search parameters and each new call corresponds to a restart. In particular, the variable ordering heuristic is based on the variable impact heuristic of [9], i.e., at each step of the search tree, CP Optimizer chooses the variable with the highest impact. CP Optimizer

Algorithm 2: Phase 1 of $CPO - ACO$

Input: a COP $P = (X, D, C, F)$ and a set of parameters $\{t_{max1}, d_{min}, it_{max}, \alpha, \beta, \rho, \tau_{min}, \tau_{max}, nbAnts\}$

Output: A feasible solution $\mathcal{A}_{best}$ and a pheromone matrix $\tau : X \times D \rightarrow [\tau_{min}; \tau_{max}]$

foreach $x_i \in X$ **and foreach** $v_i \in D(x_i)$ **do** $\tau(x_i, v_i) \leftarrow \tau_{max}$
repeat
  `/* Step 1: Construction of` $nbAnts$ `feasible solutions */`
  foreach $k \in \{1, \ldots, nbAnts\}$ **do**
    Construct a feasible solution $\mathcal{A}_k$ using CP Optimizer
  `/* Step 2: Evaporation of all pheromone trails */`
  foreach $x_i \in X$ **and foreach** $v_i \in D(x_i)$ **do**
    $\tau(x_i, v_i) \leftarrow max(\tau_{min}, (1 - \rho) \cdot \tau(x_i, v_i))$
  `/* Step 3: Pheromone laying on good feasible solutions */`
  Let $\mathcal{A}_{best}$ be the best assignment built so far (including the current cycle)
  foreach $k \in \{1, \ldots, nbAnts\}$ **do**
    if $\forall l \in \{1, \ldots, nbAnts\}, \mathcal{A}_k$ *is at least as good as* $\mathcal{A}_l$ **then**
      foreach $< x_i, v_i > \in \mathcal{A}_k$ **do**
        $\tau(x_i, v_i) \leftarrow min(\tau_{max}, \tau(x_i, v_i) + \frac{1}{1+|F(\mathcal{A}_k) - F(\mathcal{A}_{best})|})$
  if $\mathcal{A}_{best}$ *is strictly better than all feasible solutions of* $\{\mathcal{A}_1, ..., \mathcal{A}_{nbAnts}\}$ **then**
    foreach $< x_i, v_i > \in \mathcal{A}_{best}$ **do** $\tau(x_i, v_i) \leftarrow min(\tau_{max}, \tau(x_i, v_i) + 1)$
until *time spent* $\geq t_{max1}$ **or** *number of cycles without improvement of* $\mathcal{A}_{best} \geq it_{max}$ **or** *average distance of* $\{\mathcal{A}_1, ..., \mathcal{A}_{nbAnts}\} \leq d_{min}$;
return $\mathcal{A}_{best}$ **and** τ

propagates constraints using predefined procedures, and when an inconsistency is detected, it backtracks until finding a feasible solution that satisfies all constraints. Also, if the given cutoff in the number of backtracks is met without having found a solution, CP Optimizer automatically restarts the search, as described in [9].

However, the value ordering heuristic procedure is given to CP Optimizer and it is defined according to ACO: let x_i be the next variable to be assigned; v_i is randomly chosen in $D(x_i)$ w.r.t. probability

$$p(v_i) = \frac{[\tau(x_i, v_i)]^\alpha \cdot [1/impact(v_i)]^\beta}{\sum_{v_j \in D(x_i)} [\tau(x_i, v_j)]^\alpha \cdot [1/impact(v_j)]^\beta}$$

where $impact(v_i)$ is the observed impact of value v_i as defined in [9], and α and β are two parameters that weight the pheromone and impact factors respectively. Hence, during the first cycle, values are randomly chosen with respect to impacts only as all pheromone trails are initialized to the same value (i.e., τ_{max}). However, at the end of each cycle, pheromone trails are updated so that these probabilities are progressively biased with respect to past constructions.

It is worth mentioning here that our CPO-ACO framework is designed to solve underconstrained COPs that have a rather large number of feasible solutions (such as, for example, MKPs or QAPs): when solving these problems, the difficulty is not to build a feasible solution, but to find the feasible solution that optimizes the objective function. Hence, on these problems CP Optimizer is able to build feasible solutions very quickly, with very few backtracks. Our CPO-ACO framework may be used to solve more tightly constrained COPs. However, for these problems, CP Optimizer may backtrack a lot (and therefore need more CPU time) to compute each feasible solution. In this case, pheromone learning will be based on a very small set of feasible solutions so that it may not be very useful and CPO-ACO will simply behave like CP Optimizer.

Pheromone evaporation: Once every ant has constructed an assignment, pheromone trails are evaporated by multiplying them by $(1-\rho)$ where $\rho \in [0;1]$ is the pheromone evaporation rate (lines 5-6).

Pheromone laying step: At the end of each cycle, good feasible solutions (with respect to the objective function) are rewarded in order to intensify the search around them. Lines 8-11, the best feasible solutions of the cycle are rewarded. Lines 12-13, the best feasible solution built so far is rewarded if it is better than the best feasible solutions of the cycle (otherwise it is not rewarded as it belongs to the best feasible solutions of the cycle that have already been rewarded). In both cases, a feasible solution $\mathcal{A}$ is rewarded by increasing the quantity of pheromone laying on every couple $< x_i, v_i >$ of $\mathcal{A}$, thus increasing the probability of assigning x_i to v_i. The quantity of pheromone added is inversely proportional to the gap between $F(\mathcal{A})$ and $F(\mathcal{A}_{best})$.

Termination conditions: The first phase is stopped either if the CPU time limit of the first phase t_{max1} has been reached, or if $\mathcal{A}_{best}$ has not been improved since it_{max} iterations, or if the average distance between the assignments computed during the last cycle is smaller than d_{min}, thus indicating that pheromone trails have allowed the search to converge. We define the distance between two assignments with respect to the number of variable/value couples they share, i.e., the distance between $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\frac{|X|-|\mathcal{A}_1 \cap \mathcal{A}_2|}{|X|}$

3.2 Second phase of $CPO-ACO$

At the end of the first phase, the best constructed feasible solution $\mathcal{A}_{best}$ and the pheromone structure τ are forwarded to the second phase. $\mathcal{A}_{best}$ is used to bound the objective function with its cost. Then, we ask CP Optimizer to find a feasible solution that optimizes the objective function F: in this second phase, each time CP Optimizer finds a better feasible solution, it adds a constraint to bound the objective function with respect to its cost, and it backtracks to find better feasible solutions, or prove the optimality of the last computed bound.

Like in the first phase, CP Optimizer is used as a black-box with its default search parameters: the restart of [9] is used as the search type parameter, and impacts are used as variable ordering heuristic. However, the value ordering heuristic is defined by the pheromone structure τ: given a variable x_i to be assigned, CP Optimizer chooses the value $v_i \in D(x_i)$ which maximizes the formula: $[\tau(x_i, v_i)]^\alpha \cdot [1/impact(v_i)]^\beta$.

Note that we have experimentally compared different other frameworks which are listed below:

- We have considered a framework where, at the end of the first phase, we only return $\mathcal{A}_{best}$ (which is used to bound the objective function at the beginning of the second phase) and we do not return the pheromone structure (so that the value ordering heuristic used in the second phase is only defined with respect to impacts). This framework obtains significantly worse results, showing us that the pheromone structure is a valuable ordering heuristic.
- We have considered a framework where, during the first phase, the sampling is done randomly, without using pheromone for biasing probabilities (i.e., α is set to 0). This framework also obtains significantly worse results, showing us that it is worth using an ACO learning mechanism.
- We have considered a framework where, during the second phase, the value ordering heuristic is defined by the probabilistic rule used in the first phase, instead of selecting the value that maximizes the formula. This framework obtains results that are not significantly different on most instances.

4 Experimental evaluation of CPO-ACO

4.1 Considered problems

We evaluate our CPO-ACO approach on three well known COPs, i.e., Multidimensional Knapsack problem (MKP), Quadratic Assignment Problem (QAP) and the Maximum Independent Set (MIS).

The Multidimensional Knapsack problem (MKP) involves selecting a subset of objects in a knapsack so that the capacity of the knapsack is not exceeded and the profit of the selected objects is maximized. The associated CP model is such that

- $X = \{x_1, \ldots, x_n\}$ associates a decision variable x_i with every object i;
- $\forall x_i \in X$, $D(x_i) = \{0, 1\}$ so that $x_i = 0$ if i is not selected, and 1 otherwise;
- $C = \{C_1, \ldots, C_m\}$ is a set of m capacity constraints such that each constraint $C_j \in C$ is of the form $\sum_{i=1}^{n} c_{ij} \cdot x_i \leq r_j$ where c_{ij} is the amount of resource j required by object i and r_j is the available amount of resource j;
- the objective function to maximize is $F = \sum_{i=1}^{n} u_i \cdot x_i$ where u_i is the profit associated with object i.

We have considered academic instances with 100 objects which are available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html. We have considered the first 20 instances with 5 resource constraints (5-100-00 to 5-100-19); the first 20 instances with 10 resource constraints (10-100-00 to 10-100-19) and the first 20 instances with 30 resource constraints (30-100-00 to 30-100-19).

The Quadratic Assignment Problem (QAP) involves assigning facilities to locations so that a sum of products between facility flows and location distances is minimized. The associated CP model is such that

- $X = \{x_1, \ldots, x_n\}$ associates a decision variable x_i with every facility i;
- $\forall x_i \in X$, $D(x_i) = \{1, \ldots, n\}$ so that $x_i = j$ if facility i is assigned to location j;
- C only contains a global all different constraint among the whole set of variables, thus ensuring that every facility is assigned to a different location;
- the objective function to minimize is $F = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{x_i x_j} b_{ij}$ where $a_{x_i x_j}$ is the distance between locations x_i and x_j, and b_{ij} is the flow between facilities i and j.

We have considered instances of the QAPLIB which are available at http://www.opt.math.tu-graz.ac.at/qaplib/inst.html.

The Maximum Independent Set (MIS) involves selecting the largest subset of vertices of a graph such that no two selected vertices are connected by an edge (this problem is equivalent to searching for a maximum clique in the inverse graph). The associated CP model is such that

- $X = \{x_1, \ldots, x_n\}$ associates a decision variable x_i with every vertex i;
- $\forall x_i \in X$, $D(x_i) = \{0, 1\}$ so that $x_i = 0$ if vertex i is not selected, and 1 otherwise;
- C associates a binary constraint c_{ij} with every edge (i, j) of the graph. This constraint ensures that i and j have not been both selected, i.e., $c_{ij} = (x_i + x_j < 2)$.
- the objective function to maximize is $F = \sum_{i=1}^{n} x_i$.

We have considered instances of the MIS problem which are available at http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

4.2 Experimental settings

For each problem, the CP model has been written in C++ using the CP Optimizer modeling API. It is worth mentionning here that this CP model is straightforwardly derived from the definition of the problem as given in the previous section: it basically declares the variables together with their domains, the constraints, and the objective function.

We compare CPO-ACO with CP Optimizer (denoted CPO). In both cases, we used the version V2.3 of CP Optimizer with its default settings. However,

for CPO-ACO, the value ordering heuristic is given to CPO (see Section 3). For CPO, the default value ordering heuristic is defined with respect to impacts as proposed in [9].

For all experiments, the total CPU time has been limited to 300 seconds on a 2.2 Gz Pentium 4. For CPO-ACO, this total time is shared between the two phases as follows: $t_{max1} = 25\%$ of the total time (so that phases 1 cannot spend more than 75s); $d_{min} = 0.05$ (so that phase 1 is stopped as soon as the average distance is smaller than 5%); and $it_{max} = 500$ (so that phase 1 is stopped if $\mathcal{A}_{best}$ has not been improved since 500 cycles). The number of ants is $nbAnts = 20$; pheromone and impact factors are respectively weighted by $\alpha = 1$ and $\beta = 2$; and pheromone trails are bounded between $\tau_{min} = 0.01$ and $\tau_{max} = 1$.

However, we have not considered the same pheromone evaporation rate for all experiments. Indeed, both MKP and MIS have $0 - 1$ variables so that, at each cycle, one value over the two possible ones is rewarded, and ACO converges rather quickly. For the QAP, all domains contain n values (where n is equal to the number of variables) so that, at each cycle, one value over the n possible ones is rewarded. In this case, we speed-up the convergence of ACO by increasing the evaporation rate. Hence, $\rho = 0.01$ for MKP and MIS whereas $\rho = 0.1$ for QAP.

Note that we have not yet extensively studied the influence of parameters on the solution process so that the parameter setting considered here is probably not optimal. Further work will include the use of a racing algorithm based on statistical tests in order to automatically tune parameters [15].

For both CPO and CPO-ACO, we performed 30 runs per problem instance with a different random-seed for each run.

4.3 Experimental results

Table 1 gives experimental results obtained by CPO and CPO-ACO on the MKP, the QAP and the MIS. For each class of instances and each approach, this table gives the percentage of deviation from the best known solution. Let us first note that CPO and CPO-ACO (nearly) never reach the best known solution: indeed, best known solutions have usually been computed with state-of-the-art dedicated approaches. Both CPO and CPO-ACO are completely generic approaches that do not aim at competing with these dedicated approaches which have often required a lot of programming and tuning work. Also, we have chosen a reasonable CPU time limit (300 seconds) in order to allow us to perform a significant number of runs per instance, thus allowing us to use statistical tests. Within this rather short time limit, CPO-ACO obtains competitive results with dedicated approaches on the MKP (less than 1% of deviation from best known solutions); however, it is rather far from best known solutions on many instances of the QAP and the MIS.

Let us now compare CPO with CPO-ACO. Table 1 shows us that using ACO to guide CPO search improves the search process on all classes except two. However, this improvement is more important for the MKP than for the two other problems. As the two approaches have obtained rather close results on some instances, we have used statistical tests to determine if the results are significantly

Results for the MKP

			CPO			$CPO-ACO$		
Name	# I	# X	avg (sd)	$>_{avg}$	$>_{t-test}$	avg (sd)	$>_{avg}$	$>_{t-test}$
5.100-*	20	100	1.20 (0.30)	0%	0%	**0.46** (0.23)	**100%**	**100%**
10.100-*	20	100	1.53 (0.31)	0%	0%	**0.83** (0.34)	**100%**	**100%**
30.100-*	20	100	1.24 (0.06)	5%	0%	**0.86** (0.08)	**95%**	**85%**

Results for the QAP

			CPO			$CPO-ACO$		
Name	# I	# X	avg (sd)	$>_{avg}$	$>_{t-test}$	avg (sd)	$>_{avg}$	$>_{t-test}$
bur*	7	26	1.17 (0.43)	0%	0%	**0.88** (0.43)	**100%**	**57 %**
chr*	11	19	12.11 (6.81)	45%	9%	**10.99** (6.01)	**55 %**	**45 %**
had*	5	16	1.07 (0.89)	0%	0%	**0.54** (1.14)	**100%**	**60 %**
kra*	2	30	17.46 (3.00)	0%	0%	**14.99** (2.79)	**100%**	**100%**
lipa*	6	37	22.11 (0.82)	0%	0%	**20.87** (0.75)	**100%**	**100%**
nug*	15	20	8.03 (1.59)	7%	0%	**5.95** (1.44)	**93 %**	**80 %**
rou*	3	16	5.33 (1.15)	33%	0%	**3.98** (1.00)	**67 %**	**67 %**
scr*	3	16	**4.60** (2.4)	33%	0%	5.12 (2.60)	**67 %**	0 %
tai*	4	16	6.06 (1.35)	25%	25%	**4.84** (1.25)	**75 %**	**50 %**

Results for the MIS

			CPO			$CPO-ACO$		
Name	# I	# X	avg (sd)	$>_{avg}$	$>_{t-test}$	avg (sd)	$>_{avg}$	$>_{t-test}$
frb-30-15-*	5	450	9.83 (1.86)	0%	0%	**9.46** (2.00)	**80%**	**20%**
frb-35-17-*	5	595	**11.62** (2.05)	**60%**	0%	11.82 (2.31)	40%	0%
frb-40-19-*	5	760	13.47 (1.92)	20%	0%	**12.85** (2.22)	**80%**	**20%**
frb-45-21-*	5	945	15.40 (2.43)	0%	0%	**14.35** (1.82)	**100%**	**80%**
frb-50-23-*	5	1150	16.24 (2.32)	20%	0%	**15.84** (2.00)	**80%**	**20%**
frb-53-24-*	5	1272	18.15 (2.55)	0%	0%	**16.86** (1.84)	**100%**	**80%**
frb-56-25-*	5	1400	17.85 (2.37)	20%	0%	**16.89** (1.08)	**80%**	**40%**
frb-59-26-*	5	1534	18.40 (2.44)	40%	0%	**18.37** (2.16)	**60%**	**20%**

Table 1. Comparison of CPO and CPO-ACO on the MKP, the QAP and the MIS. Each line successively gives: the name of the class, the number of instances in the class ($\#I$), the average number of variables in these instances ($\#X$), the results obtained by CPO (resp. CPO-ACO), i.e., the percentage of deviation from the best known solution (average (avg) and standard deviation (sd)), the percentage of instances for which CPO (resp. CPO-ACO) has obtained strictly better average results ($>_{avg}$), and the percentage of instances for which CPO (resp. CPO-ACO) is significantly better w.r.t. the statistical test.

different or not: we have performed the Student's t-test with significance level of 0.05, using the R Stats Package available at http://sekhon.berkeley.edu/doc-/html/index.html. For each class, we report the percentage of instances for which an approach has obtained significantly better results than the other one (column $>_{t-test}$ of table 1). For the MKP, CPO-ACO is significantly better than CPO for 57 instances, whereas it is not significantly different for 3 instances. For the QAP, CPO-ACO is significantly better than CPO on a large number of instances. However, CPO is better than CPO-ACO on one instance of the class tai* of the QAP. For the MIS, CPO-ACO is significantly better than CPO on 35% of instances, but it is not significantly different on all other instances.

5 Conclusion

We have proposed CPO-ACO, a generic approach for solving COPs defined by means of a set of constraints and an objective function. This generic approach combines a complete B&P&B approach with ACO. One of the main ideas behind this combination is the utilization of the effectiveness of (i) ACO to explore the search space and quickly identify promising areas (ii) CP Optimizer to strongly exploit the neighborhood of the best solutions found by ACO. This combination allows us to reach a good balance between diversification and intensification of the search: diversification is mainly ensured during the first phase by ACO; intensification is ensured by CP optimizer during the second phase.

It is worth noting that thanks to the modular nature of IBM ILOG CP Optimizer that clearly separates the modeling part of the problem from its resolution part, the proposed combination of ACO and CP was made in natural way. Hence, the CPO-ACO program used was exactly the same for the experiments on the different problems used in this work.

We have shown through experiments on three different COPs that CPO-ACO is significantly better than CP Optimizer.

5.1 Related Works

Recent research has focused on the integration of ACO in classical branch and bound algorithms, but most of them were applied on specific problems and/or proposed a combination based on an incomplete search.

In particular, B.Meyer has proposed in [16] two hybrid algorithms where the metaheuristic Ant Colony Optimization (ACO) was coupled with CP. In his work, Meyer has proposed a loose coupling where both components run in parallel, exchanging only (partial) solutions and bounds. Then, he has proposed a tight coupling where both components collaborate in an interleaved fashion so that, the constraint propagation was embedded in ACO in order to allow an ant to backtrack when an association of a value v with a given variable fails. However, the backtrack procedure was limited at the level of the last chosen variable. This means that, if all the possible values of the last chosen variable have been tried without success, the search of an ant ends with failure. The results of this work

show on the machine scheduling problem with sequence-dependent setup time that the tight coupling is better. But unfortunately, the proposed tight coupling is not based on a complete search and in the both proposed algorithms; the author has assumed that the variable ordering is fixed.

Also, we have proposed in [12] an hybrid approach, denoted Ant-CP, which combines ACO with a CP solver in order to solve constraint satisfaction problems (without objective function to optimize). Like CPO-ACO, Ant-CP uses the CP modeling language to define the problem, and ants use predefined CP procedures to check and propagate constraints. However, unlike CPO-ACO, Ant-CP performs an incomplete search which never backtracks.

5.2 Further work

There are several points which are worth mentioning as further improvements to CPO-ACO. At the moment, CPO-ACO works well (if we compare it with CP Optimizer) on the problems for which finding a feasible solution is relatively easy. In this paper, we have applied CPO-ACO on three different problems without using problem-dependent heuristics. We plan to study the interest of adding problem-dependent heuristics, that may improve the efficiency of CPO-ACO and allow it to become competitve with state-of-the-art approches.

Parameter tuning is another interesting topic. For the moment the parameters of CPO-ACO are roughly tuned using our experience, but we believe that an adaptive version which dynamically tunes the parameters during the execution should significantly increase the algorithm's efficiency and robustness.

6 Acknowledgments

The authors thank Renaud Dumeur, Jérôme Rogerie, Philippe Refalo and Philippe Laborie for the fruitful and animated discussions about combinatorial optimization in general case and, particularly, the search strategies. Also, we thank Renaud Dumeur and Paul Shaw for proofreading drafts of this paper.

References

1. Nemhauser, G., Wolsey, A.: Integer and combinatorial optimization. New York: Jhon Wiley and & Sons; (1988)
2. Papadimitriou, C., Steiglitz, K.: Combinatorial optimization–Algorithms and complexity. New York:Dover; (1982)
3. Kirkpatrick, S., Gellat, C., Vecchi, M.: Optimization by simulated annealing. science **220** (1983) 671–680
4. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997)
5. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In F. Golver and G. Kochenberger, editors, Handbook of Metaheuristics, volume 57 of International Series in Operations Research Management Science. Kluwer Academic Publisher (2001) pages 321–353.

6. Hensen, P., Mladenović, N.: An introduction to variable neighborhood search. In S. Voβ, S. Martello, I. Osman, and C. Roucairol, editors, Meta-heuristics: advances and trends in local search paradigms for optimization, pages 433–438. Kluwer Academic Publisher (1999)
7. Hentenryck, P.V., Michel, L.: Constraint-Based Local Search. MIT Press (2005)
8. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
9. Refalo, P.: Impact-based search strategies for constraint programming. In: In CP04, 10th International Conference on Principales and Practice of Constraint Programming. Volume 3258 of LNCS., Springer (2004) 557–571
10. Stützle, T., Hoos, H.: $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Journal of Future Generation Computer Systems **16** (2000) 889–914
11. Solnon, C.: Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation **6**(4) (2002) 347–357
12. Khichane, M., Albert, P., Solnon, C.: Integration of ACO in a constraint programming language. In: 6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS'08). Volume 5217 of LNCS., Springer (2008) 84–95
13. Solnon, C., Fenet, S.: A study of ACO capabilities for solving the Maximum Clique Problem. Journal of Heuristics **12**(3) (2006) 155–180
14. Alaya, I., Solnon, C., Ghedira, K.: Ant Colony Optimization for Multi-objective Optimization Problems. In: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), IEEE Computer Society (2007) 450–457
15. Birattari, M., Stutzle, T., Paquete, L., Varrentrapp, K.: A Racing Algorithm for Configuring Metaheuristics. In: GECCO. (2002) 11–18
16. Meyer, B.: Hybrids of constructive meta-heuristics and constraint programming: A case study with ACO. In Chr. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, Hybrid Metaheuristics-An emergent approach for optimization. Springer Verlag, New York (2008)