Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 716069, 12 pages
http://dx.doi.org/10.1155/2013/716069

Research Article

A Hybrid Soft Computing Approach for Subset Problems

Broderick Crawford,[1,2] Ricardo Soto,[1,3] Eric Monfroy,[4] Carlos Castro,[5] Wenceslao Palma,[1] and Fernando Paredes[6]

[1] *Pontificia Universidad Católica de Valparaíso, Valparaíso 2362807, Chile*
[2] *Universidad Finis Terrae, Santiago 7500000, Chile*
[3] *Universidad Autónoma de Chile, Santiago 7500000, Chile*
[4] *CNRS, LINA, Université de Nantes, Nantes 44322, France*
[5] *Universidad Técnica Federico Santa María, Valparaíso 2390123, Chile*
[6] *Escuela de Ingeniería Industrial, Universidad Diego Portales, Santiago 8370179, Chile*

Correspondence should be addressed to Broderick Crawford; broderick.crawford@ucv.cl

Received 7 April 2013; Revised 21 June 2013; Accepted 22 June 2013

Academic Editor: Ker-Wei Yu

Subset problems (set partitioning, packing, and covering) are formal models for many practical optimization problems. A set partitioning problem determines how the items in one set (S) can be partitioned into smaller subsets. All items in S must be contained in one and only one partition. Related problems are set packing (all items must be contained in zero or one partitions) and set covering (all items must be contained in at least one partition). Here, we present a hybrid solver based on ant colony optimization (ACO) combined with arc consistency for solving this kind of problems. ACO is a swarm intelligence metaheuristic inspired on ants behavior when they search for food. It allows to solve complex combinatorial problems for which traditional mathematical techniques may fail. By other side, in constraint programming, the solving process of Constraint Satisfaction Problems can dramatically reduce the search space by means of arc consistency enforcing constraint consistencies either prior to or during search. Our hybrid approach was tested with set covering and set partitioning dataset benchmarks. It was observed that the performance of ACO had been improved embedding this filtering technique in its constructive phase.

1. Introduction

Set covering problem (SCP) and set partitioning problem (SPP) have many applications, including those involving routing, scheduling, stock cutting, electoral redistricting, and other important real-life situations [1, 2]. Although the best known application of the SPP is airline crew scheduling [3, 4], several other applications exist, including vehicle routing problems (VRP) [5, 6] and query processing [7]. The main disadvantage of SPP-based models is the need to explicitly generate a large set of possibilities to obtain good solutions. Additionally, in many cases, a prohibitive time is needed to find the exact solution.

Furthermore, Set Partitioning Problems occur as subproblems in various combinatorial optimization problems [8]. In airline scheduling, a subtask called crew scheduling takes as input data a set of crew pairings, where the selection of crew pairings which cause minimal costs and ensure that each flight is covered exactly once can be modeled as a set partitioning problem [1, 9]. In [10, 11], solving a particular case of VRP, the dial-a-ride problem (DARP), also uses an SPP decomposition approach.

Because the SPP formulation has demonstrated to be useful modeling important industrial problems (or their phases), it is our interest to solve it with novel techniques. In this work, we solve some test instances of SPP and SCP (SCP is considered a relaxation of SPP) with ant colony optimization (ACO) algorithms and some hybridizations of ACO with a constraint programming (CP) technique: constraint propagation [12].

ACO is a swarm intelligence metaheuristic which is inspired from the foraging behavior of real ant colonies. The

ants deposit pheromone on the ground marking the path for identification by other members of the colony of the routes from the nest to food [13]. From the early nineties, ACO attracted the attention of researchers, and many successful applications solving optimization problems are done [14].

There exist already some good approaches applying ACO to subset problems [15]. In [16], ACO is applied on the set packing problem using two solution construction strategies based on exploration and exploitation. In general, the same occurs in relation with SCP, applying ACO only as a construction algorithm and testing the approach only on some small SCP instances. More recent works apply ant computing to the SCP and related problems using techniques to remove redundant columns and local search to improve solutions [17–22].

The best performing metaheuristics for SPP are genetic algorithms [23, 24]. Taking into account these results, it seems that the incomplete approach of ant computing could be considered as a good alternative to solve these problems when complete techniques are not able to get the optimal solution in a reasonable time.

ACO is of limited effectiveness solving very strongly constrained problems. They are problems for which neighborhoods contain few solutions, or none at all, and local search is of very limited use. Probably, the most significant of such problems is the SPP. A direct implementation of the basic ACO framework is incapable of obtaining feasible solutions for many standard tested instances of SPP [25–27].

Trying to solve larger instances of SPP with the original ant system (AS) [28] or ant colony system (ACS) [29] implementation derives in a lot of unfeasible labeling of variables, and the ants cannot obtain complete solutions using the classic transition rule when they move in their neighborhood. The root of the problem is that simply following the random proportional transition rule; that is, learning/reforcing good paths is no longer enough, as this does not check for constraint consistency.

In order to improve this aspect of ACO, we are working in the addition of a constraint programming mechanism in the construction phase of ACO; thus, only feasible partial solutions are generated. The CP mechanism allows the incorporation of information about the instantiation of variables after the current decision. In general, ACO algorithms are competitive with other optimization techniques when applied to problems that are not overly constrained. However, when solving highly constrained problems the performance of ACO algorithms degrades. When a problem is highly constrained, the difficulty is in finding feasible solutions. This is where CP comes into play, because these problems are the target problems for CP solvers. CP is a programming paradigm in which a combinatorial optimization problem is modeled as a discrete optimization problem, specifying the constraints that a feasible solution must meet. The CP approach to search for a feasible solution often works by the iteration of constraint propagation and the addition of additional constraints, and it transforms the problem without changing its solutions. Constraint propagation is the mechanism that reduces the domains of the decision variables with respect to the given set of constraints [30].

Although the idea of obtaining sinergy from hybridization of ACO with CP is not novel [31–36], our proposal is a bit different. We explore the addition to the ACO algorithm of a mechanism to check constraint consistency usually used in complete techniques: arc consistency. Other kinds of cooperation between ACO and CP are shown in [37], where combinatorial optimization problems are solved in a generic way by a two-phase algorithm. The first phase aims to create 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, a CP optimizer performs a complete tree search guided by the pheromone trails previously accumulated.

Here, we propose the addition of a lookahead mechanism in the construction phase of ACO in order that only feasible solutions are generated. The lookahead mechanism allows the incorporation of information about the instantiation of variables after the current decision. The idea differs from that proposed by [31, 32], and these authors proposed a lookahead function evaluating the pheromone in the shortest common supersequence problem and estimating the quality of a partial solution of an industrial scheduling problem, respectively.

This paper is organized as follows. In Section 2, we explain the problem. In Section 3, we describe the ACO framework. In Section 4, we present the definitions considered in constraint propagation. Our hybrid proposal is described in Section 5. In Section 6, we present the experimental results obtained. Finally, in Section 7, we conclude the paper and give some perspectives for future research.

2. Problem Description

SPP is the problem of partitioning a given set into manually independent subsets while minimizing a cost function defined as the sum of the costs associated with each of the eligible subsets.

In the SPP matrix formulation, we are given a $m \times n$ matrix $A = (a_{ij})$ in which all the matrix elements are either zero or one. Additionally, each column is given a nonnegative cost c_j.

Let $I = 1, \ldots, m$ and $J = 1, \ldots, n$ be the row set and column set, respectively.

We say that a column j covers a row i if $a_{ij} = 1$. Let x_j be a binary variable which is one if column j is chosen and zero otherwise. The SPP can be defined formally as *minimize* (1) *subject to* (2). These constraints enforce that each row is covered by exactly one column. The SCP is an SPP relaxation. The goal in the SCP is to choose a subset of the columns of minimal weight formally using constraints to enforce that each row is covered by at least one column as (3):

$$f(x) = \sum_{j=1}^{n} c_j x_j, \tag{1}$$

$$\sum_{j=1}^{n} a_{ij} x_j = 1; \quad \forall i = 1, \ldots, m, \tag{2}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1; \quad \forall i = 1, \ldots, m, \tag{3}$$

$$x_j \in \{0, 1\}; \quad \forall j = 1, \ldots, n. \tag{4}$$

The notations in (5) are often used to complete the description of the problem:

$$\begin{aligned} J_i &= \left\{ j \in J \mid a_{ij} = 1 \right\} : \text{subset of columns covering row } i, \\ I_j &= \left\{ i \in I \mid a_{ij} = 1 \right\} : \text{subset of rows covered by column } j, \\ d &= \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}}{mn} : \text{density, that is,} \\ &\quad \text{the ratio of non-zero entries in } A. \end{aligned} \tag{5}$$

3. Ant Colony Optimization for Set Partitioning Problems

In this section, we briefly present ACO algorithms and give a description of their use to solve SPP. More details about ACO algorithms can be found in [13].

The basic idea of ACO algorithms comes from the capability of real ants to find the shortest paths between the nest and food source. From a combinatorial optimization point of view, the ants are looking for *good solutions*. Real ants cooperate in their search for food by depositing pheromone on the ground. An artificial ant colony simulates this behavior implementing artificial ants as parallel processes whose role is to build solutions using a randomized constructive search driven by pheromone trails and heuristic information of the problem.

An important topic in ACO is the adaptation of the pheromone trails during algorithm execution to take into account the cumulated search experience: reinforcing the pheromone associated with good solutions and considering the *evaporation* of the pheromone on the components over time in order to avoid premature convergence. ACO can be applied in a very straightforward way to SPP. The columns are chosen as the solution components and have associated a cost and a pheromone trail [13]. Each column can be visited by an ant only once, and then a final solution has to cover all rows. A walk of an ant over the graph representation corresponds to the iterative addition of columns to the partial solution obtained so far. Each ant starts with an empty solution and adds columns until a cover is completed. A pheromone trail τ_j and a heuristic information η_j are associated with each eligible column j. A column to be added is chosen with a probability that depends on pheromone trail and the heuristic information. The most common form of the ACO decision policy (*Transition Rule Probability*) when ants work with components is

$$p_j^k(t) = \frac{\tau_j^{\alpha} \eta_j^{\beta}}{\sum_{l \notin S^k} \tau_l [\eta_l]^{\beta}} \quad \text{if } j \notin S^k, \tag{6}$$

where S^k is the partial solution of the ant k. α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information in the probabilistic decision [13, 18].

3.1. Pheromone Trail τ_j. One of the most crucial design decisions to be made in ACO algorithms is the modeling of the set of pheromones. In the original ACO implementation for TSP, the choice was to put a pheromone value on every link between a pair of cities, but for other combinatorial problems pheromone values can be often assigned to the decision variables (first-order pheromone values) [13]. In this work, the pheromone trail is put on the problem's component (each eligible column j) instead of the problems connections. And setting a good pheromone quantity is not a trivial task either. The quantity of pheromone trail laid on columns is based on the idea that *the more pheromone trail on a particular item, the more profitable the item is* [15]. Then, the pheromone deposited in each component will be in relation to its frequency in the ants solutions (In this work, we divided this frequency by number of ants.).

3.2. Heuristic Information η_j. In this paper, we use a dynamic heuristic information that depends on the partial solution of an ant. It can be defined as $\eta_j = e_j/c_j$, where e_j is the so-called cover value, that is, the number of additional rows covered when adding column j to the current partial solution, and c_j is the cost of column j. In other words, the heuristic information measures the unit cost of covering one additional row. An ant ends the solution construction when all rows are covered.

3.3. AS and ACS. In this work, we hybridize with CP two instances of ACO: ant system (AS) [28] and ant colony system (ACS) [29] algorithms, the original and the most famous algorithms in the ACO family.

ACS differs from AS in the following aspects. First, it exploits the search experience accumulated by the ants more strongly than AS does through the use of a more aggressive action choice rule. Second, pheromone evaporation and pheromone deposit take place only on the columns belonging to the best so far solution. Third, each time an ant chooses a column j, it removes some pheromone from the component increasing the exploration. ACS has demonstrated better performance than AS in a wide range of problems.

ACS exploits a pseudorandom transition rule in the solution construction; ant k chooses the next column j with criteria

$$\underset{l \notin S^k}{\text{Argmax}} \left\{ \tau_l [\eta_l]^{\beta} \right\} \quad \text{if } q \leq q_o, \tag{7}$$

following the Transition Rule Probability (6) or otherwise, where q is a random number uniformly distributed in $[0, 1]$ and q_0 is a parameter that controls how strongly the ants exploit deterministically the pheromone trail and the heuristic information. It should be mentioned that ACS uses a candidate list to restrict the number of available choices to be considered at each construction step. The candidate list

```
(1) Begin
(2)   InitParameters()
(3)   While remain iterations do
(4)     Foreach ant do
(5)       While solution is not completed and (TabuList <> j) do
(6)         Choose next column j with Transition Rule Probability
(7)         AddColumnToTabuList(j)
(8)       End While
(9)     End Foreach
(10)    UpdateOptimum()
(11)    UpdatePheromone()
(12)  End While
(13)  Return BestSolution
(14) End
```

Algorithm 1: ACO for SCP and SPP.

contains a number of the best rated columns according to the heuristic criterion.

Trying to solve larger instances of SPP with the original AS or ACS implementation derives in a lot of unfeasible labeling of variables, and the ants cannot obtain complete solutions. In this paper we explore, the addition of an arc consistency mechanism in the construction phase of ACO; thus, only feasible solutions are generated. A direct implementation of the basic ACO framework is incapable of obtaining feasible solution for many SPP instances.

3.4. The ACO Framework. Each ant starts with an empty solution and adds column until a cover is completed. But to determine if a column actually belongs or not to the partial solution ($j \neq S^k$) is not good enough.

The traditional ACO decision policy (4) does not work for SPP because the ants, in this traditional selection process of the next columns, ignore the information of the problem constraint when a variable is instantiated. And in the worst case, in the iterative steps, it is possible to assign values to some variables that will make it impossible to obtain complete solution (see Algorithm 1).

4. Constraint Propagation

Constraint propagation is crucial in CP, and it appears under different names: constraint relaxation, filtering algorithms, narrowing algorithms, constraint inference, simplification algorithms, label inference, local consistency enforcing, rules iteration, and chaotic iteration [38].

Constraint propagation embeds any reasoning which consists in explicitly forbidding values or combinations of values for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise. Arc consistency is the most well-known way of propagating constraints.

4.1. Arc Consistency. Arc-consistency is one of the most used filtering techniques in constraint satisfaction for reducing the combinatorial space of problems. Arc-consistency is formally defined as a local consistency within the constraint programming field [39]. A local consistency defines properties that the constraint problem must satisfy after constraint propagation. Constraint propagation is simply the process when the given local consistency is enforced to the problem. In the following, some necessary definitions are stated [38].

Definition 1 (constraint). A constraint c is a relation defined on a sequence of variables $X(c) = (x_{i_1}, \ldots, x_{i_{|X(c)|}})$, called the scheme of c. c, is the subset of $\mathbb{Z}^{|X(c)|}$ that contains the combinations of values (or tuples) $\tau \in \mathbb{Z}^{|X(c)|}$ that satisfy c. $|X(c)|$ is called the arity of c. A constraint c with scheme $X(c) = (x_1, \ldots, x_k)$ is also noted as $c(x_1, \ldots, x_k)$.

Definition 2 (constraint network). A constraint network also known as constraint satisfaction problem (CSP) is defined by a triple $N = \langle X, D, C \rangle$, where

(i) X is a finite sequence of integer variables $X = (x_1, \ldots, x_n)$,

(ii) D is the corresponding set of domains for X, that is, $D = D(x_1) \times \cdots \times D(x_n)$, where $D(x_i) \subset \mathbb{Z}$ is the finite set of values that variable x_i can take,

(iii) C is a set of constraints $C = \{c_1, \ldots, c_e\}$, where variables in $X(c_j)$ are in X.

Definition 3 (projection). A projection of c on Y is denoted as $\pi_{Y(c)}$, which defines the relation with scheme Y that contains the tuples that can be extended to a tuple on $X(c)$ satisfying c.

As previously mentioned, arc-consistency is one of the most used ways of propagating constraints. Arc-consistency was initially defined for binary constraint [40, 41], that is, constraints involving two variables. We here give the more general definition for nonarbitrary constraints named generalized arc-consistency (GAC).

Definition 4 ((generalized) arc consistency). Given a network $N = \langle X, D, C \rangle$, a constraint $c \in C$, and a variable $x_i \in X(c)$,

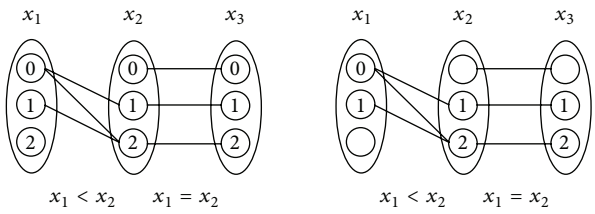

Figure 1: Enforcing arc-consistency.

Input: x_i, c
Output: *CHANGE*
(1) *CHANGE* ← false
(2) **Foreach** $v_i \in D(x_i)$ **do**
(3) **If** $\nexists \tau \in c \cap \pi_{X(c)}(D)$ with $\tau[x_i] = v_i$ **then**
(4) remove v_i from $D(x_i)$
(5) *CHANGE* ← true
(6) **End If**
(7) **End Foreach**
(8) **Return** *CHANGE*

Algorithm 2: Revise3.

(i) a value $v_i \in D(x_i)$ is consistent with $c \in D$ if and only if there exists a valid tuple τ satisfying c such that $v_i = \tau[\{x_i\}]$. Such a tuple is called a support for (x_i, v_i) on c,

(ii) the domain D is (generalized) arc-consistent on c for x_i if and only if all the values in $D(x_i)$ are consistent with c in D, that is, $D(x_i) \subseteq \pi_{x_i}(c \cap \pi_{X(c)}(D))$,

(iii) the network N is (generalized) arc-consistent if and only if D is (generalized) arc-consistent for all variables in X on all constraints in C.

As an example, let us consider the non-arc-consistent network N depicted on left side of Figure 1. It considers three variables x_1, x_2, and x_3; domains $D(x_1) = D(x_2) = D(x_3) = \{0, 1, 2\}$; and constraints c_{12}: $(x_1 < x_2)$ and c_{23}: $(x_2 = x_3)$. Enforcing arc-consistency allows one to eliminate some inconsistent values. For instance, when constraint c_{12} is verified, the value 2 from $D(x_1)$ is removed since there is no value greater than it in $D(x_2)$. The value 0 from $D(x_2)$ is also removed since no support for it exists in $D(x_1)$. Removing 0 from $D(x_2)$ leads to the removal of 0 from $D(x_3)$ when c_{23} is checked. The resulting arc-consistent network is depicted on the right side of Figure 1.

Such a filtering process can be carried out by using Algorithms 2 and 3.

As previously illustrated, the main idea of this process is the revision of arcs, that is, to eliminate every value in $D(X_i)$ that is inconsistent with a given constraint c. This notion is encapsulated in the function `Revise3`. This function takes each value v_i in $D(x_i)$ (line 2) and analyses the space $\tau \in c \cap \pi_{X(c)}(D)$, searching for a support on constraint c (line 3). If support does not exist, the value v_i is eliminated from $D(x_i)$. Finally, the function informs if $D(X_i)$ has been changed by returning true or false otherwise (line 8).

Algorithm 3 is responsible for ensuring that every domain is consistent with the set of constraints. This is done by using a loop that verifies arcs until no change happens. The function begins by filling a list Q with pairs (x_i, c) such that $x_i \in X(c)$. The idea is to keep the pairs for which $D(x_i)$ is not ensured to be arc-consistent with respect to c. This allows to avoid useless calls to `Revise3` as done in more basic algorithms such as AC1 and AC2. Then, a loop that takes the pairs (x_i, c) from Q (line 2) and `Revise3` is called (line 4). If `Revise3` returns true, $D(X_i)$ is checked whether it is an empty set. If so, the algorithm returns false. Otherwise, normally, a value for another variable x_j has lost its support on c. Thus, all pairs (x_i, c) such that $x_i \in X(c)$ must be reinserted in the list Q. The algorithm ends once Q is empty, and it returns true when all arcs have been verified and remaining values of domains are arc-consistency with respect to all constraints.

5. Hybridization of Ants and Constraint Programming

Hybrid algorithms provide appropriate compromises between exact (or complete) search methods and approximate (or incomplete) methods, some efforts have been done in order to integrate constraint programming (exact methods) to ants algorithms (stochastic local search methods) [31–36].

A hybridization of ACO and CP can be approached from two directions: we can either take ACO or CP as the base algorithm and try to embed the respective other method into it. A form to integrate CP into ACO is to let it reduce the possible candidates among the not yet instantiated variables participating in the same constraints that current variable. A different approach would be to embed ACO within CP. The point at which ACO can interact with CP is during the labeling phase using ACO to learn a value ordering that is more likely to produce good solutions.

In this work, ACO uses CP in the variable selection (when ACO adds a column to partial solution). The CP algorithm used in this paper is the AC3 filtering procedure [42]. It performs consistency between pairs of a not yet instantiated variable and an instantiated variable; that is, when a value is assigned to the current variable, any value in the domain of a future variable which conflicts with this assignment is removed from the domain.

The AC3 filtering procedure, taking into account the constraints network topology (i.e., which sets of variables are linked by a constraint and which are not), guaranties that at each step of the search, all constraints between ready assigned variables and not yet assigned variables are consistent; it means that columns are chosen if they do not generate any conflicts with the next column to be chosen. Then, a new transition rule is developed embedding AC3 in the ACO framework (see Algorithm 4, lines 7 to 12).

5.1. ACO + CP to SCP. In Figure 2, we present an explanation of how ACO + CP works solving SCP. Here, x_i represents the

Input: X, D, C
Output: Boolean
(1) $\{Q \leftarrow (x_i, c) \mid c \in C, x_i \in X(c)\}$
(2) **While** $Q \neq \emptyset$ **do**
(3) select and remove (x_i, c) from Q
(4) **If** Revise3(x_i, c) **then**
(5) **If** $D(x_i) = \emptyset$ **then**
(6) **Return** false
(7) **Else**
(8) $Q \leftarrow Q \cup \{(x_j, c') \mid c' \in C \wedge c' \neq c \wedge x_i, x_j \in X(c') \wedge j \neq i\}$
(9) **End If**
(10) **End If**
(11) **End While**
(12) **Return** true

Algorithm 3: AC3/GAC3.

(1) **Begin**
(2) InitParameters()
(3) **While** remain iterations **do**
(4) **Foreach** ant **do**
(5) **While** solution is not completed **and** (*TabuList* <> j) **do**
(6) Choose next column j with Transition Rule Probability
(7) feasible(i):= AC3$\left(X \notin S^k, D_{X \notin S^k}, C_{I_j}\right)$ {/* Filtering $i \in I_j^*$ */}
(8) **If** feasible$(i) \forall i$ **then**
(9) AddColumnToSolution(j)
(10) **Else**
(11) Backtracking(j) {/* Set j uninstantiated */}
(12) **End If-Else**
(13) AddColumnToTabuList(j)
(14) **End While**
(15) **End Foreach**
(16) UpdateOptimum()
(17) UpdatePheromone()
(18) **End While**
(19) **Return** *BestSolution*
(20) **End**

Algorithm 4: Hybrid ACO + CP for SCP and SPP.

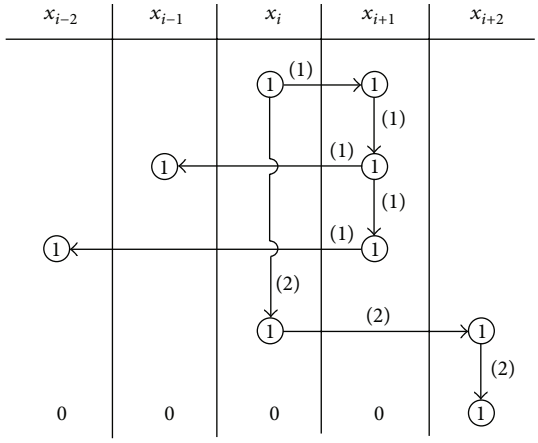

Figure 2: ACO + CP to SCP.

current variable, x_{i-1} and x_{i-2} are variables already instantiated, and x_{i+1} and x_{i+2} are not yet instantiated variables (The sequence $\cdots x_{i-1}, x_{i-2}, x_i, x_{i+1}, x_{i+2} \cdots$ is the order of variable instantiations given by the enumeration strategy in use.).

(1) *Constraint Propagation Assigning Value 0 in Future Variables.* When instantiating a column for each row that can cover that column, the other columns that can also cover it can be put to 0, as long as all rows that can cover each of these columns are already covered (i.e., the search space is reduced and also favoring the optimization).

(2) *Constraint Propagation Assigning Value 1 in Future Variables.* If there is a column, which is the only one that can cover a row, this column can be put to 1.

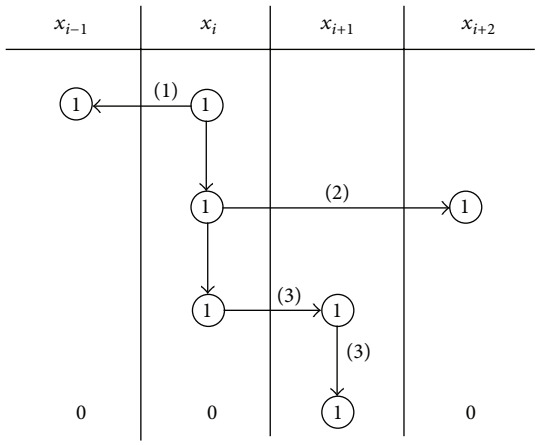

Figure 3: ACO + CP to SPP.

5.2. ACO + CP to SPP. In Figure 3, we present an explanation of how ACO + CP works solving SPP. Here, x_i represents the current variable, x_{i-1} is a variable already instantiated, and x_{i+1} and x_{i+2} are not yet instantiated variables.

(1) *Backtracking in Current Variable.* If any of the rows that can cover a column is already been covered, the current column cannot be instantiated because it violates a constraint, and then it should be done by using backtracking.

(2) *Constraint Propagation Assigning Value 0 in Future Variables.* When instantiating a column for each uncovered row that can cover that column, the other columns that can also cover (i.e., by constraint propagation it reduces the search space should be put to 0, and in practice, for the uninstantiated variable, the value 1 of its domain was eliminated).

(3) *Backtracking in Current Variable and Constraint Propagation Assigning Value 1 in Future Variables.* If any of these columns (which were assigned the value 0) is the only column that can cover another row, the current column cannot be instantiated because it does not lead to a solution, and then it should be done a backtracking.

6. Experimental Evaluation

We have implemented AS and ACS and the proposed AS + CP and ACS + CP algorithms. The effectiveness of the proposed algorithms was evaluated experimentally using SCP and SPP test instances from Beasley's OR-library [43]. Each instance was solved (else it is indicated with *n.f.*) 12 times, and the algorithms have been run with 100 ants and a maximum number of 200 iterations. Table 1 shows the value considered for each standard ACO parameter: α is the relative influence of the pheromone trail, β is the relative influence of the the heuristic information, ρ is the pheromone evaporation rate, q_0 is used in ACS pseudorandom proportional action choice rule, ϵ is used in ACS local pheromone trail update, and

Table 1: Parameter settings for ACO algorithms.

ACO algorithm	α	β	ρ	q_0	ϵ	ls
AS	1	0.5	0.4	—	—	—
ACS	1	0.5	0.99	0.5	0.1	300

the ACS list size *ls* was 300. For each instance, the initial pheromone τ_0 was calculated as follows:

$$\tau_0 = \frac{2m}{\left(\text{Opt} + \sum_{j=1}^{n} c_j\right)}. \tag{8}$$

Algorithms were implemented using ANSI C, GCC 3.3.6, under a 2.0 GHz Intel Core2 Duo T5870 with 1 Gb RAM running Microsoft Windows XP Professional.

Tables 2(a) and 2(b) describe problem instances, and they show the problem code, the number of rows m (constraints), the number of columns n (decision variables), the *Density* (i.e., the percentage of nonzero entries in the problem matrix), and the best known cost value for each instance Opt (IP optimal) of the SCP and SPP instances used in the experimental evaluation.

Computational results (best cost obtained) are shown in Tables 3(a), 3b), 4(a), 4(b), 5(a), 5(b), 6(a), and 6(b) and in Figures 4 and 5. The quality of a solution is evaluated using the relative percentage deviation (RPD) and the relative percentage improvement (RPI) measures [19]. The RPD value quantifies the deviation of the objective value Z from Z_{opt} which in our case is the best known cost value for each instance (see the third column), and the RPI value quantifies the improvement of Z from an initial solution Z_I (see the fourth column).

These measures are computed as follows:

$$\text{RPD} = \frac{\left(Z - Z_{\text{opt}}\right)}{Z_{\text{opt}}} \times 100,$$
$$\text{RPI} = \frac{\left(Z_I - Z\right)}{\left(Z_I - Z_{\text{opt}}\right)} \times 100. \tag{9}$$

For all the implemented algorithms, the solution quality and computational effort (Secs) are related using the marginal relative improvement (MIC) measure (see the fifth column). This measure quantifies the improvement achieved per unit of CPU time (RPI/Secs). The solution time is measured in CPU seconds, and it is the time that each algorithm takes to first reach the final best solution.

The results expressed in terms of the average RPD, average RPI, and average MIC show the effectiveness of AS + CP and ACS + CP over AS and ACS to solve SCP (see Tables 3(a), 3(b), 4(a), and 4(b)). Our hybrid solver provides high quality near optimal solutions, and it has the ability to generate them for a variety of instances.

From Tables 5(a), 5(b), 6(a), and 6(b), it can be observed that AS + CP and ACS + CP solving SPP obtained better results than AS and ACS (viewing average RPD and average

Table 2: SCP and SPP instances.

(a) Details of the SCP test instances

Problem	m	n	Density	Opt
scp410	200	1000	2	514
scpa1	300	3000	2	253
scpa2	300	3000	2	252
scpa3	300	3000	2	232
scpa4	300	3000	2	234
scpa5	300	3000	2	236
scpb1	300	3000	5	69
scpb2	300	3000	5	76
scpb3	300	3000	5	80
scpc1	400	4000	2	227
scpc2	400	4000	2	219
scpc3	400	4000	2	243
scpc4	400	4000	2	219
scpcyc07	672	448	2	144
scpcyc08	1792	1024	2	344
scpcyc09	4608	2304	2	780
scpcyc10	11520	5120	2	1792
scpd1	400	4000	5	60
scpd2	400	4000	5	66
scpd3	400	4000	5	72

(b) Details of the SPP test instances

Problem	m	n	Density	Opt
sppaa01	823	8904	1	56137
sppaa02	531	5198	1.3	30494
sppaa03	825	8627	1	49649
sppaa05	801	8308	0.99	55839
sppaa06	646	7292	1.1	27040
sppnw06	50	6774	18.2	7810
sppnw08	24	434	22.4	35894
sppnw09	40	3103	16.2	67760
sppnw10	24	853	21.2	68271
sppnw12	27	626	20	14118
sppnw15	31	467	19.5	67743
sppnw18	124	10757	6.8	340160
sppnw19	40	2879	21.9	10898
sppnw23	19	711	24.8	12534
sppnw26	23	771	23.8	6796
sppnw32	19	294	24.3	14877
sppnw34	20	899	28.1	10488
sppnw39	25	677	26.6	10080
sppnw41	17	197	22.1	11307

Table 3: Experimental results of SCP benchmarks using AS and AS + CP.

(a) AS experimental results

Problem	AS	RPD	RPI	MIC	Secs
scp410	539	4.86	99.9	9.7	10.3
scpa1	592	133.99	96.6	1.18	82
scpa2	531	110.71	97.2	0.74	130.5
scpa3	473	103.88	97.3	0.78	125
scpa4	375	60.26	98.5	0.87	113.3
scpa5	349	47.88	98.8	9.41	10.5
scpb1	243	184.06	95.3	3.29	29
scpb2	207	219.74	94.4	5.33	17.7
scpb3	442	158.75	95.9	0.5	190.5
scpc1	484	94.71	97.6	1.39	70
scpc2	551	121	96.9	4.73	20.5
scpc3	523	126.75	96.8	0.74	130.5
scpc4	272	138.81	96.4	1.6	60.2
scpcyc07	512	88.89	97.7	1.38	70.7
scpcyc08	1297	48.84	98.8	0.79	124.6
scpcyc09	3123	66.28	98.3	0.7	140.5
scpcyc10	1969	74.27	98.1	0.65	150.3
scpd1	184	206.67	94.7	9.28	10.2
scpd2	209	216.67	94.4	1.04	90.7
scpd3	221	206.94	94.7	0.5	190.6
avg		120.7	96.9	2.73	

(b) AS + CP experimental results

Problem	AS + CP	RPD	RPI	MIC	Secs
scp410	556	8.17	99.79	10.29	9.7
scpa1	288	13.83	99.65	1.25	79.5
scpa2	285	13.1	99.66	0.83	120
scpa3	270	16.38	99.58	1.99	50.1
scpa4	278	18.8	99.52	4.29	23.2
scpa5	272	15.25	99.61	6.55	15.2
scpb1	75	8.7	99.78	8.32	12
scpb2	87	14.47	99.63	9.67	10.3
scpb3	89	11.25	99.71	0.76	130.4
scpc1	261	14.98	99.62	1.43	69.8
scpc2	260	18.72	99.52	6.55	15.2
scpc3	268	10.29	99.74	0.68	147.3
scpc4	259	18.26	99.53	1.94	51.3
scpcyc07	148	2.78	99.93	1.99	50.3
scpcyc08	364	5.81	99.85	1	100.1
scpcyc09	816	4.62	99.88	1	100.3
scpcyc10	1969	9.88	99.75	1.1	90.7
scpd1	72	20	99.49	10.15	9.8
scpd2	74	12.12	99.69	1.42	70.2
scpd3	83	15.28	99.61	0.53	187.9
avg		12.64	99.68	3.59	

RPI), but they are a bit worse in the average MIC. It indicates that the computational effort is slightly higher using the hybridization, but our approach can obtain optimal solutions in some instances where AS or ACS failed. Our hybrid approach shows an excellent tradeoff between the quality of the solutions obtained and the computational effort required.

Table 4: Experimental results of SCP benchmarks using ACS and ACS + CP.

(a) ACS experimental results

Problem	ACS	RPD	RPI	MIC	Secs
scp410	669	30.16	99.23	12.72	7.8
scpa1	348	37.55	99.04	1.21	81.6
scpa2	378	50	98.72	0.83	118.9
scpa3	319	37.5	99.04	2.07	47.9
scpa4	333	42.31	98.92	5.18	19.1
scpa5	353	49.58	98.73	8.66	11.4
scpb1	101	46.38	98.81	9.78	10.1
scpb2	117	53.95	98.62	13.89	7.1
scpb3	112	40	98.97	0.64	153.8
scpc1	305	34.36	99.12	1.49	66.4
scpc2	309	41.1	98.95	6.87	14.4
scpc3	367	51.03	98.69	0.65	151.3
scpc4	324	47.95	98.77	1.93	51.1
scpcyc07	321	122.92	96.85	2.48	39
scpcyc08	769	123.55	96.83	1.75	55.4
scpcyc09	1723	120.9	96.9	1.34	72.3
scpcyc10	4097	128.63	96.7	0.86	112
scpd1	92	53.33	98.63	20.13	4.9
scpd2	96	45.45	98.83	1.64	60.2
scpd3	111	54.17	98.61	0.56	177.5
avg		60.54	98.45	4.73	

(b) ACS + CP experimental results

Problem	ACS + CP	RPD	RPI	MIC	Secs
scp410	664	29.18	99.25	9.93	10
scpa1	331	30.83	99.21	1.79	55.3
scpa2	376	49.21	98.74	1.64	60.2
scpa3	295	27.16	99.3	2.98	33.3
scpa4	301	28.63	99.27	5.37	18.5
scpa5	335	41.95	98.92	9.89	10
scpb1	115	66.67	98.29	8.06	12.2
scpb2	110	44.74	98.85	14.98	6.6
scpb3	117	46.25	98.81	1.14	87
scpc1	317	39.65	98.98	2.23	44.4
scpc2	311	42.01	98.92	19.02	5.2
scpc3	328	34.98	99.1	0.97	101.7
scpc4	303	38.36	99.02	3.56	27.
scpcyc07	321	122.92	96.85	3.2	30.3
scpcyc08	769	123.55	96.83	2.03	47.7
scpcyc09	1445	85.26	97.81	1.76	55.5
scpcyc10	3506	95.65	97.55	1.96	49.7
scpd1	105	75	98.08	18.86	5.2
scpd2	113	71.21	98.17	2.18	45
scpd3	119	65.28	98.33	1.09	90.3
avg		57.92	98.5	5.63	

Table 5: Experimental results of SPP benchmarks using AS and AS + CP.

(a) AS experimental results

Problem	AS	RPD	RPI	MIC	Secs
sppaa01	96256	71.47	98.71	0.09	1100.3
sppaa02	39883	30.79	99.21	0.33	300.7
sppaa03	63734	28.37	99.27	0.11	900.6
sppaa05	61703	10.5	99.73	0.13	778.3
sppaa06	42015	55.38	98.58	0.25	400.5
sppnw06	9200	17.8	99.54	1.1	90.8
sppnw08	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw09	70462	3.99	99.9	7.04	14.2
sppnw10	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw12	15406	9.12	99.77	12.96	7.7
sppnw15	67755	0.02	100	14.71	6.8
sppnw18	385596	13.36	99.66	0.5	200.2
sppnw19	11678	7.16	99.82	1.43	69.7
sppnw23	14304	14.12	99.64	14.65	6.8
sppnw26	6976	2.65	99.93	13.32	7.5
sppnw32	14877	0	100	10.31	9.7
sppnw34	13341	27.2	99.3	13.24	7.5
sppnw39	11670	15.77	99.6	16.06	6.2
sppnw41	11307	0	100	14.93	6.7
avg		18.1	99.54	7.13	

(b) AS + CP experimental results

Problem	AS + CP	RPD	RPI	MIC	Secs
sppaa01	60246	7.32	99.81	0.1	1000.7
sppaa02	37452	22.82	99.41	0.4	250.4
sppaa03	55082	10.94	99.72	0.12	800.3
sppaa05	58158	4.15	99.89	0.15	680
sppaa06	33524	23.98	99.39	0.32	313.5
sppnw06	8160	4.48	99.89	1.18	84.7
sppnw08	35894	0	100	0.14	700.3
sppnw09	70222	3.63	99.91	7.99	12.5
sppnw10	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw12	14466	2.46	99.94	9.52	10.5
sppnw15	67743	0	100	18.18	5.5
sppnw18	345762	1.65	99.96	0.9	110.8
sppnw19	11060	1.49	99.96	1.27	78.9
sppnw23	13932	11.15	99.71	16.9	5.9
sppnw26	6880	1.24	99.97	12.04	8.3
sppnw32	14877	0	100	9.35	10.7
sppnw34	10713	2.15	99.94	8.84	11.3
sppnw39	11322	12.32	99.68	21.21	4.7
sppnw41	11307	0	100	15.38	6.5
avg		6.1	99.8	6.89	

7. Conclusion

Highly constrained combinatorial optimization problems have proved to be a challenge for constructive metaheuristic. In this paper, ACO framework has been modified so that it may be applied to any constraint satisfaction problem in general and hard constrained instances in particular.

Table 6: Experimental results of SPP benchmarks using ACS and ACS + CP.

(a) ACS experimental results

Problem	ACS	RPD	RPI	MIC	Secs
sppaa01	94720	67.93	98.26	0.16	600.8
sppaa02	57632	88.99	97.72	0.36	270.4
sppaa03	93304	87.93	97.75	0.13	780.5
sppaa05	91134	63.21	98.38	0.14	712.3
sppaa06	54964	103.27	97.35	0.38	254.9
sppnw06	9788	25.33	99.35	1.26	78.7
sppnw08	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw09	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw10	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw12	16060	13.76	99.65	8.1	12.3
sppnw15	67746	0	100	17.54	5.7
sppnw18	365398	7.42	99.81	0.83	120.7
sppnw19	12350	13.32	99.66	1.35	73.7
sppnw23	14604	16.52	99.58	17.47	5.7
sppnw26	6956	2.35	99.94	12.81	7.8
sppnw32	14886	0.06	100	8.06	12.4
sppnw34	11289	7.64	99.8	14.68	6.8
sppnw39	10758	6.73	99.83	19.2	5.2
sppnw41	11307	0	100	16.67	6
avg		31.53	99.2	7.45	

(b) ACS + CP experimental results

Problem	ACS + CP	RPD	RPI	MIC	Secs
sppaa01	88435	50.41	98.71	0.14	707.1
sppaa02	52211	71.22	98.17	0.32	309.4
sppaa03	81177	63.5	98.37	0.16	600.5
sppaa05	84362	51.08	98.69	0.17	590.7
sppaa06	48703	80.11	97.95	0.33	300.9
sppnw06	8038	2.92	99.93	3.3	30.3
sppnw08	36682	2.2	99.94	0.32	309.7
sppnw09	69332	2.32	99.94	2.45	40.8
sppnw10	n.f.	n.f.	n.f.	n.f.	n.f.
sppnw12	14252	0.95	99.98	9.34	10.7
sppnw15	67743	0	100	16.13	6.2
sppnw18	345130	1.46	99.96	1.47	67.8
sppnw19	11858	8.81	99.77	1.42	70.1
sppnw23	12880	2.76	99.93	16.66	6
sppnw26	6880	1.24	99.97	15.87	6.3
sppnw32	14877	0	100	9.8	10.2
sppnw34	10797	2.95	99.92	17.53	5.7
sppnw39	10545	4.61	99.88	15.85	6.3
sppnw41	11307	0	100	19.23	5.2
avg		19.25	99.5	7.25	

A direct implementation of the basic ACO framework is incapable of obtaining feasible solutions for many strongly constrained problems. In order to improve this aspect of ACO, we integrated a constraint programming mechanism

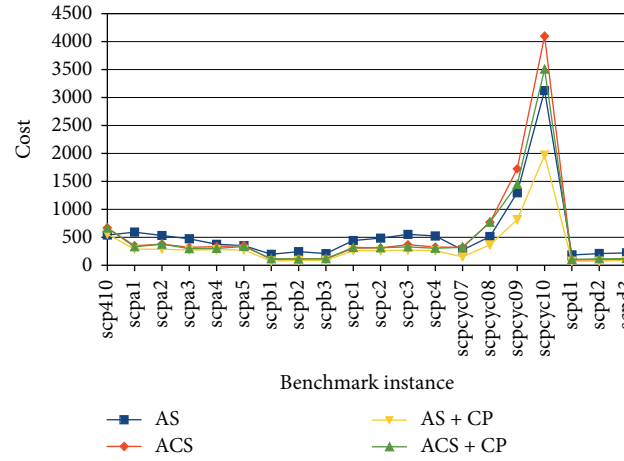

Figure 4: Experimental results for SCP.

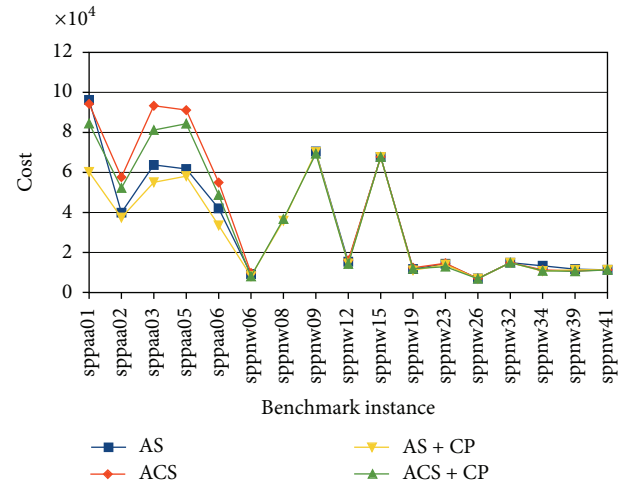

Figure 5: Experimental results for SPP.

in the construction phase of ACO; thus, only feasible partial solutions are generated.

The effectiveness of the proposed rule was tested on benchmark problems, and we solved SCP and SPP with ACO using constraint propagation in its transition rule, and results were compared with pure ACO algorithms. About efficiency, the computational effort required is almost the same.

An interesting extension of this work would be related to hybridization of AC3 with other metaheuristics [44]. The use of autonomous search (AS) [45] in conjunction with constraint programming would be also a promising direction to follow. AS represents a new research field, and it provides practitioners with systems that are able to autonomously self-tune their performance while effectively solving problems. Its major strength and originality consist in the fact that problem solvers can now perform self-improvement operations based on analysis of the performances of the solving process. In [46, 47], the order in which the variables are selected for instantiation is determined by a choice function that dynamically selects from a set of variable ordering heuristics

the one that best matches the current problem state, and this combination can accelerate the resolution process, especially in harder instances. In [48], the results show that some phases of reactive propagation are beneficial to the main hybrid algorithm, and the hybridization strategies are thus crucial in order to decide when to perform, or not, constraint propagation.

Furthermore, we are considering to use different preprocessing steps from the OR literature, which allow to reduce the problem size [49, 50].

Acknowledgment

The author Fernando Paredes is supported by FONDECYT-Chile Grant 1130455.

References

[1] E. Balas and M. Padberg, "Set partitioning: a survey," Management Sciences Research Report, Defense Technical Information Center, Fort Belvoir, Va, USA, 1976.

[2] T. A. Feo and M. G. C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, no. 2, pp. 67–71, 1989.

[3] A. Mingozzi, M. A. Boschetti, S. Ricciardelli, and L. Bianco, "A set partitioning approach to the crew scheduling problem," *Operations Research*, vol. 47, no. 6, pp. 873–888, 1999.

[4] M. Mesquita and A. Paias, "Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem," *Computers and Operations Research*, vol. 35, no. 5, pp. 1562–1575, 2008, special issue: algorithms and computational methods in feasibility and infeasibility.

[5] J. P. Kelly and J. Xu, "A set-partitioning-based heuristic for the vehicle routing problem," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 161–172, 1999.

[6] M. L.] Balinski and R. E. Quandt, "On an integer program for a delivery problem," *Operations Research*, vol. 12, no. 2, pp. 300–304, 1964.

[7] R. D. Gopal and R. Ramesh, "Query clustering problem: a set partitioning approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 6, pp. 885–899, 1995.

[8] G. B. Alvarenga and G. R. Mateus, "A two-phase genetic and set partitioning approach for the vehicle routing problem with time windows," in *Proceedings of the 4th International Conference on Hybrid Intelligent Systems (HIS '04)*, M. Ishikawa, S. Hashimoto, M. Paprzycki et al., Eds., pp. 428–433, IEEE Computer Society, 2004.

[9] F. Barahona and R. Anbil, "On some difficult linear programs coming from set partitioning," *Discrete Applied Mathematics*, vol. 118, no. 1-2, pp. 3–11, 2002, special Issue devoted to the ALIO-EURO Workshop on Applied Combinatorial Optimization.

[10] R. Borndorfer, M. Grotschel, F. Klostermeier, and C. Kuttner, "Telebus berlin: vehicle scheduling in a dial-a-ride system," Tech. Rep. SC 9723, Konrad-Zuse-Zentrum fur Informationstechnik, Berlin, Germany, 1997.

[11] B. Crawford, C. Castro, and E. Monfroy, "Solving dial-a-ride problems with a low-level hybridization of ants and constraint programming," in *Proceedings of the 2nd International Work-Conference on the Interplay between Natural and Artificial Computation (IWINAC '07)*, J. Mira and J. R. Alvarez, Eds., vol. 4528 of *Lecture Notes in Computer Science*, pp. 317–327, Springer, 2007.

[12] K. Apt, *Principles of Constraint Programming*, Cambridge University Press, New York, NY, USA, 2003.

[13] M. Dorigo and T. Stutzle, *Ant Colony Optimization*, MIT Press, Cambridge, Mass, USA, 2004.

[14] B. Chandra Mohan and R. Baskaran, "A survey: ant colony optimization based recent research and implementation on several engineering domain," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4618–4627, 2012.

[15] G. Leguizamon and Z. Michalewicz, "A new version of ant system for subset problems," in *Proceedings of Congress on Evolutionary Computation (CEC '99)*, P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds., IEEE Press, July 1999.

[16] X. Gandibleux, X. Delorme, and V. T'Kindt, "An ant colony optimisation algorithm for the set packing problem," in *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS '04)*, pp. 49–60, 2004.

[17] M. Rahoual, R. Hadji, and V. Bachelet, "Parallel ant system for the set covering problem," in *Ant Algorithms*, pp. 262–267, 2002.

[18] L. Lessing, I. Dumitrescu, and T. Stutzle, "A comparison between aco algorithms for the set covering problem," in *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS '04)*, pp. 1–12, 2004.

[19] Z.-G. Ren, Z.-R. Feng, L.-J. Ke, and Z.-J. Zhang, "New ideas for applying ant colony optimization to the set covering problem," *Computers and Industrial Engineering*, vol. 58, no. 4, pp. 774–784, 2010.

[20] R. M. D. A. Silva and G. L. Ramalho, "Ant system for the set covering problem," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 3129–3133, October 2001.

[21] R. Hadji, M. Rahoual, E. Talbi, and V. Bachelet, "Ant colonies for the set covering problem," in *Proceedings of 2nd International Workshop on Ant Algorithms (ANTS '00)*, M. Dorigo, Ed., pp. 63–66, Brussels, Belgium, 2000.

[22] M. H. Mulati and A. A. Constantino, "Ant-line: a lineoriented aco algorithm for the set covering problem," in *Proceedings of the 30th International Conference of the Chilean Computer Science Society (SCCC '11)*, pp. 265–274, Curico, Chile, November 2011.

[23] P. C. Chu and J. E. Beasley, "Constraint handling in genetic algorithms: the set partitioning problem," *Journal of Heuristics*, vol. 4, no. 4, pp. 323–357, 1998.

[24] D. Levine, "A parallel genetic algorithm for the set partitioning problem," Tech. Rep., Illinois Institute of Technology, Chicago, Ill, USA, 1994.

[25] V. Maniezzo and M. Milandri, "An ant-based framework for very strongly constrained problems," in *Ant Algorithms*, pp. 222–227, 2002.

[26] V. Maniezzo and M. Roffilli, "Very strongly constrained problems: an ant colony optimization approach," *Cybernetics and Systems*, vol. 39, no. 4, pp. 395–424, 2008.

[27] M. Randall and A. Lewis, "Modifications and additions to ant colony optimisation to solve the set partitioning problem," in *Proceedings of the 6th IEEE International Conference on e-Science Workshops (e-ScienceW '10)*, pp. 110–116, Los Alamitos, Calif, USA, December 2010.

[28] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on*

Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 26, no. 1, pp. 29–41, 1996.

[29] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[30] C. Blum, "Ant colony optimization: introduction and recent trends," *Physics of Life Reviews*, vol. 2, no. 4, pp. 353–373, 2005.

[31] R. Michel and M. Middendorf, "An island model based ant system with lookahead for the shortest supersequence problem," in *Parallel Problem Solving from Nature*, vol. 1498 of *Lecture Notes in Computer Science*, pp. 692–701, Springer, Berlin, Germany, 1998.

[32] C. Gagne, M. Gravel, and W. Price, "A look-ahead addition to the ant colony optimization metaheuristic and its application to an industrial scheduling problem," in *Proceedings of the 4th Metaheuristics International Conference (MIC '01)*, J. Sousa, Ed., pp. 79–84, Porto, Portugal, July 2001.

[33] B. Meyer and A. T. Ernst, "Integrating aco and constraint propagation," in *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS '04)*, pp. 166–177, 2004.

[34] B. Crawford and C. Castro, "Aco with lookahead procedures for solving set partitioning and covering problems," in *Proceedings of Workshop on Combination of Metaheuristic and Local Search with Constraint Programming Techniques*, Nantes, France, November 2005.

[35] B. Crawford and C. Castro, "Integrating lookahead and post processing procedures with aco for solving set partitioning and covering problems," in *Proceedings of the 8th International Conference on Artificial Intelligence and Soft Computing (ICAISC '06)*, L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds., vol. 4029 of *Lecture Notes in Computer Science*, pp. 1082–1090, Springer, 2006.

[36] D. R. Thiruvady, C. Blum, B. Meyer, and A. T. Ernst, "Hybridizing beam-aco with constraint programming for single machine job scheduling," in *Hybrid Metaheuristics*, M. J. Blesa, C. Blum, L. D. Gaspero, A. Roli, M. Sampels, and A. Schaerf, Eds., vol. 5818 of *Lecture Notes in Computer Science*, pp. 30–44, Springer, Berlin, Germany, 2009.

[37] M. Khichane, P. Albert, and C. Solnon, "Strong combination of ant colony optimization with constraint programming optimization," in *Proceedings of the 7th International Conference on Integration of Artificial Intelligence and Operations Research (CPAIOR '10)*, A. Lodi, M. Milano, and P. Toth, Eds., vol. 6140 of *Lecture Notes in Computer Science*, pp. 232–245, Springer, 2010.

[38] C. Bessiere, "Constraint propagation," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., pp. 29–84, Elsevier, Rio de Janeiro, Brazil, 2006.

[39] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, Elsevier, Rio de Janeiro, Brazil, 2006.

[40] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[41] A. Mackworth, "On reading sketch maps," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '77)*, pp. 598–606., 1977.

[42] R. Dechter and D. Frost, "Backjump-based backtracking for constraint satisfaction problems," *Artificial Intelligence*, vol. 136, no. 2, pp. 147–188, 2002.

[43] J. E. Beasley, "Or-library: distributing test problems by electronic mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[44] R. Soto, B. Crawford, C. Galleguillos, E. Monfroy, and F. Paredes, "A hybrid ac3-tabu search algorithm for solving sudoku puzzles," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5817–5821, 2013.

[45] Y. Hamadi, E. Monfroy, and F. Saubion, "An introduction to autonomous search," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds., pp. 1–11, Springer, Berlin, Germany, 2011.

[46] B. Crawford, C. Castro, E. Monfroy, R. Soto, W. Palma, and F. Paredes, "Dynamic selection of enumeration strategies for solving constraint satisfaction problems," *Romanian Journal of Information Science and Technology*, vol. 15, no. 2, pp. 106–128, 2012.

[47] B. Crawford, R. Soto, E. Monfroy, W. Palma, C. Castro, and F. Paredes, "Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1690–1695, 2013.

[48] E. Monfroy, C. Castro, B. Crawford, R. Soto, F. Paredes, and C. Figueroa, "A reactive and hybrid constraint solver," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 25, no. 1, pp. 1–22, 2013.

[49] T. Muller, "Solving set partitioning problems with constraint programming," in *Proceedings of the 6th International Conference on the Practical Application of Prolog and the 4th International Confernce on the Practical Application of Constraint Technology (PAPPACT '98)*, pp. 313–332, London, UK, March 1998.

[50] M. Krieken, H. Fleuren, and M. Peeters, "Problem reduction in set partitioning problems," Discussion Paper 2003-80, Tilburg University, Center for Economic Research, Tilburg, The Netherlands, 2003.