International Journal of the Physical Sciences Vol. 6(19), pp. 4667-4673, 16 September, 2011
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.771
ISSN 1992 - 1950 ©2011 Academic Journals

Full Length Research Paper

A hybrid Ant algorithm for the set covering problem

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

[1]Pontificia Universidad Católica de Valparaíso, Chile.
[2]Universidad Técnica Federico Santa María, Chile.
[3]Universidad Autónoma de Chile, Chile.
[4]CNRS, LINA, Université de Nantes, France.
[5]Escuela de Ingeniería Industrial, Universidad Diego Portales, Santiago, Chile.

Accepted 28 June, 2011

Set covering problem is the model for many important industrial applications. In this paper, we solve some benchmarks of this problem with ant colony optimization algorithms using a new transition rule. A look-ahead mechanism was incorporated to check constraint consistency in ant computing. Computational results are presented showing the advantages to use this additional mechanism to ant system and ant colony system.

Key words: Set covering problem, ant colony optimization, look-ahead techniques.

INTRODUCTION

Set covering problem (SCP) is a type of problem that can model several real life situations, including crew scheduling in railway and mass transit companies (Feo and Resende, 1989). In this work, we solve several benchmarks of SCP with ant colony optimization (ACO) algorithms and some hybridizations of ACO with a constraint programming (CP) look-ahead technique: Forward checking (Bessiere, 2006; Rossi et al., 2006).

There exist problems for which ACO is of limited effectiveness. Among them, a prominent role is played by very strongly constrained problems. They are problems for which neighborhoods contain a few solutions or none at all, and local search is of very limited use. SCP and set partitioning problem (SPP) are of such problems.

A direct implementation of the basic ACO framework is incapable of obtaining feasible solutions for many standard tested instances of SPP (Maniezzo and Milandri, 2002). The root of the problem is that simply following the random-proportional rule, that is, learning-reforcing good paths is no longer enough, as this does not check for constraint consistency.

There already exist some early approaches applying ACO to the SCP. In Leguizamon and Michalewicz (1999), ACO has been used only as a construction algorithm and the approach has only been tested on some small SCP instances. More recent works (Rahoual et al., 2002; Lessing et al, 2004; Gandibleux et al., 2004) apply ant systems to the SCP and related problems using techniques to remove redundant columns and local search to improve solutions.

In this paper, we explore the addition to the ACO algorithm of a look-ahead mechanism usually used in complete techniques. Trying to solve larger instances of SPP with the original ant system (AS) or ant colony system (ACS) implementation derives in a lot of unfeasible labeling of variables, and the ants can not obtain complete solutions using the classic transition rule when they move in their neighborhood.

We propose the addition of a look-ahead mechanism in the construction phase of ACO in order that only feasible solutions are generated. The look-ahead mechanism allows the incorporation of information about the instantation of variables after the current decision.

The idea differs from that proposed by (Michel and Middendorf, 1998; Gagne et al., 2001). These authors proposed a look ahead function evaluating the pheromone in the shortest common super-sequence problem and estimating the quality of a partial solution of an industrial

*Corresponding author. E-mail: broderick.crawford@ucv.cl.

```
Begin
InitParameter();
while (remain iterations) do
for (k:=1 to nAnt s) do
while (solution is not completed and  TabuList <> J) do
Choose next column j with Transition  Rule Probability
for (each Row i covered by j) do
feasible(i):=Posting(j)
end for
if (feasible(i) for all i) then
AddColumnToSolution(j)
else
Backtracking(j) /*Set j uninstAnt iated
end if
AddColumnToTabuList(j);
end while
end for
UpdateOptimum();
UpdatePheromone();
end while
Return best solution founded
End
```

Algorithm 1. Procedure ACO for SCP and SPP.

scheduling problem.

Combining ACO and constraint programming, the work that is closest to this study is that of Khichane et al. (2010). They introduced an approach which combines ACO and CP optimizer for solving combinatorial optimization problems.

PROBLEM DESCRIPTION

The relevance to solve SCP and SPP lies in that they are models for many important applications in the field of Operational Research. For instance, they can be used to describe scheduling or timetabling problems.

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 to each of the eligible subsets.

In SPP, we have given a *mxn* matrix $A = a_{ij}$, in which all the matrix elements are either zero or one. Additionally, each column is given a non-negative cost c_j. We say that a column *j* can cover a row *i* if $a_{ij} = 1$. Let x_j be a binary decision 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 a 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).

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

$$\sum_{j=1}^{n} a_{ij} x_j = 1 \quad \forall i = 1,...,m \quad (2)$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i = 1,...,m \quad (3)$$

SOLVING SPP AND SCP WITH ACO METAHEURISTIC

We solve SPP and SCP instances with Ant colony optimization. Artificial Ant build problem solutions using a constructive procedure driven by a combination of artificial pheromone, heuristic information (problem data) and a transition rule used to evaluate successive constructive steps. For solving SPP and SCP, the columns are chosen as the solution components and have associated a cost and a pheromone trail (Dechter and Frost, 2002). 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 (Algorithm 1).

$$p_j^k(t) = \frac{\tau_j(\eta_j)^\beta}{\sum_{l \notin S^k} \tau_l(\eta_l)^\beta} \quad \text{if} \quad j \notin S^k \tag{4}$$

$$\eta_j = \frac{e_j}{c_j} \tag{5}$$

A pheromone trail τ_j and a heuristic information η_j are associated to each eligible column *j*. A column to be added is chosen with a probability that depends of the pheromone trail and the heuristic information.

The most common form of the ACO decision policy (Transition Rule Probability) when the Ant s work with components is defined in (4), where S^k is the partial solution of the Ant k. The β parameter controls how important η is in the probabilistic decision. In this work, the pheromone trail τ_j is put on the problems component (each eligible column *j*) instead of the problems connections. Setting a good pheromone quantity is not a trivial task either. The quantity of pheromone trail laid on columns is based on the idea: *The more pheromone trail in particular item, the more profitable that item is* (Leguizamon and Michalewicz, 1999). Then, the pheromone deposited in each component will be in relation to its frequency in the Ant solutions. In this work, we divided this frequency by the number of Ants obtaining better results.

We use dynamic heuristic information that depends on the partial solution of an Ant. It can be defined as (5), where *ej* 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*.

Algorithm 1 describes the basic structure of ACO algorithm to solve SCP and SPP. 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. We use two ACO instances: Ant system (AS) and Ant colony system (ACS) algorithms, the original and the most famous algorithms in the ACO family. Generally ACS improves the search of AS by using: a different transition rule in the constructive phase, exploiting the heuristic information in a more rude form (pseudorandom), a list of candidates to future labeling and a different treatment of the pheromone.

A direct implementation of the Basic ACO framework is incapable of obtaining feasible solution for many SPP instances (Crawford et al., 2006). 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 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 a instantiated. And in the worst case, in the iterative steps, it is possible to assign values to some variable that will make it impossible to obtain a complete solution. To improve it, we use a procedure similar to the constraint propagation technique from CP (Apt, 2003; Bessiere, 2006).

LOW LEVEL 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) (Meyer and Ernst, 2004; Khichane et al., 2010).

An hybridization of ACO and CP can be approached from two directions: We can either take ACO or CP as the base algorithm and then 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 the 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 use CP in the variable selection (when adding columns to partial solution). The CP algorithm used in this paper is forward checking with backtracking (Dechter and Frost, 2002). It performs arc consistency between pairs composed 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 forward checking procedure, taking into account the constraint network topology (that is, which sets of variables are linked by a constraint and which are not), guarantees that at each step of the search, all constraints between already assigned variables and not yet assigned variables are consistent; it means that columns are chosen if they do not produce any conflicts with the next column to be chose. Then, a new transition rule is developer adding forward checking to ACO.

EXPERIMENTAL RESULTS

The computational experiments showed that AS+FC and ACS+FC out performed AS and ACS. We have tested SCP and SPP benchmark instances of Beasley or-library (Beasley, 1990). Tables 1 and 2 and Figures 1 to 4 show the result for solving SCP and SPP with AS and ACS, respectively. The algorithms ran with the following parameters settings: Influence of heuristic information β = 0.5 and evaporation rate ρ = 0.4.

The number of Ants was set to 100 and the maximum

Table 1. Experimental results of SCP benchmarks.

Problem	m	n	Opt	AS	ACS	AS+FC	ACS+FC
scp410	200	1000	514	539	669	556	664
scpa1	300	3000	253	592	348	288	331
scpa2	300	3000	252	531	378	285	376
scpa3	300	3000	232	473	319	270	295
scpa4	300	3000	234	375	333	278	301
scpa5	300	3000	236	349	353	272	335
scpb1	300	3000	69	196	101	75	115
scpb2	300	3000	76	243	117	87	110
scpb3	300	3000	80	207	112	89	117
scpc1	400	4000	227	442	305	261	317
scpc2	400	4000	219	484	309	260	311
scpc3	400	4000	243	551	367	268	328
scpc4	400	4000	219	523	324	259	303
scpcyc07	672	448	144	272	321	148	321
scpcyc08	1792	1024	344	512	769	364	769
scpcyc09	4608	2304	780	1297	1723	816	1445
scpcyc10	11520	5120	1792	3123	4097	1969	3506
scpd1	400	4000	60	184	92	72	105
scpd2	400	4000	66	209	96	74	113
scpd3	400	4000	72	221	111	83	119

M, Number of rows (constraints); n, number of columns (decision variables); Opt, the best known cost value for each instance (IP optimal), when applying Ant algorithms, AS and ACS, and combining them with forward checking.

Table 2. Experimental results of SPP benchmarks.

Problem	m	n	Opt	AS	ACS	AS+FC	ACS+FC
sppaa01	8233	8904	56137	96256	94270	60246	84435
sppaa02	531	5198	30494	39883	57632	37452	52211
sppaa03	825	8627	49649	63734	93304	55082	81177
sppaa05	801	8308	55839	61703	91134	58158	84362
sppaa06	646	7292	27040	42015	54964	33524	48703
sppnw06	50	6774	7810	9200	9788	8160	8038
sppnw08	24	434	35894	x	x	35894	36682
sppnw09	40	3103	67760	70462	x	70222	69332
sppnw10	24	853	68271	x	x	x	x
sppnw12	27	626	14118	15406	16060	14466	14252
sppnw15	31	467	67743	67755	67746	67743	67743
sppnw18	124	10757	51624	51624	73006	60224	62832
sppnw19	40	2879	10898	11678	12350	11060	11858
sppnw23	19	711	12534	14304	14604	13932	12880
sppnw26	23	771	6796	6976	6956	6880	6880
sppnw32	19	294	14877	14877	14886	14877	14877
sppnw34	20	899	10488	13341	11289	10713	10797
sppnw39	25	677	10080	11670	10758	11322	10545
sppnw41	17	197	11307	11307	11307	11307	11307

m, number of rows (constraints); n, number of columns (decision variables); Opt, the best known cost value for each instance (IP optimal), when applying Ant Algorithms, AS and ACS, and combining them with forward checking.

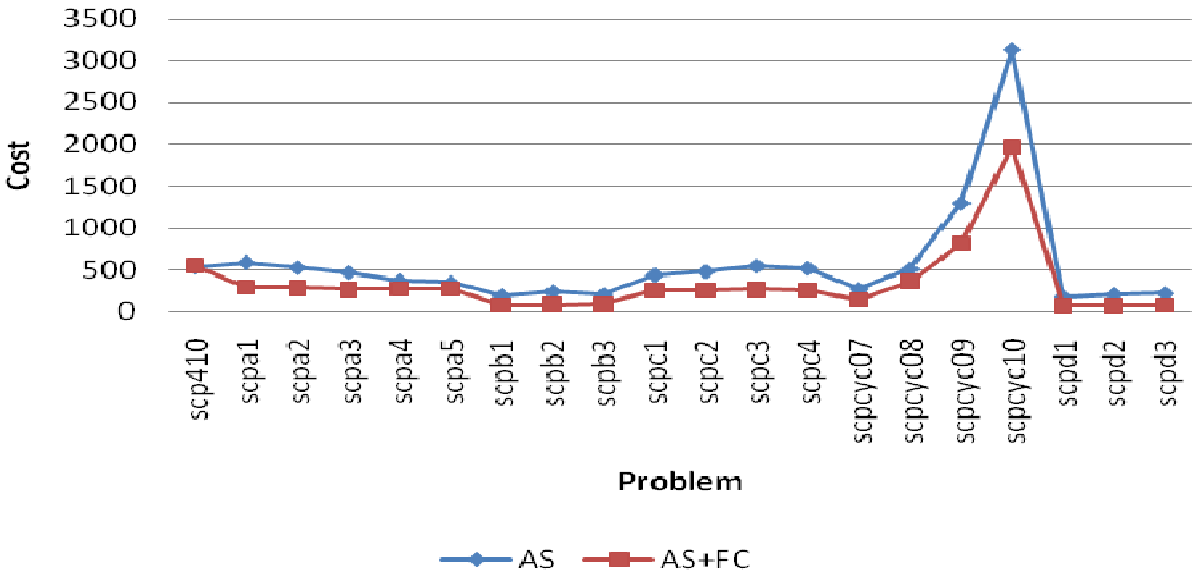

Figure 1. Experimental results for SCP with AS and AS+FC.

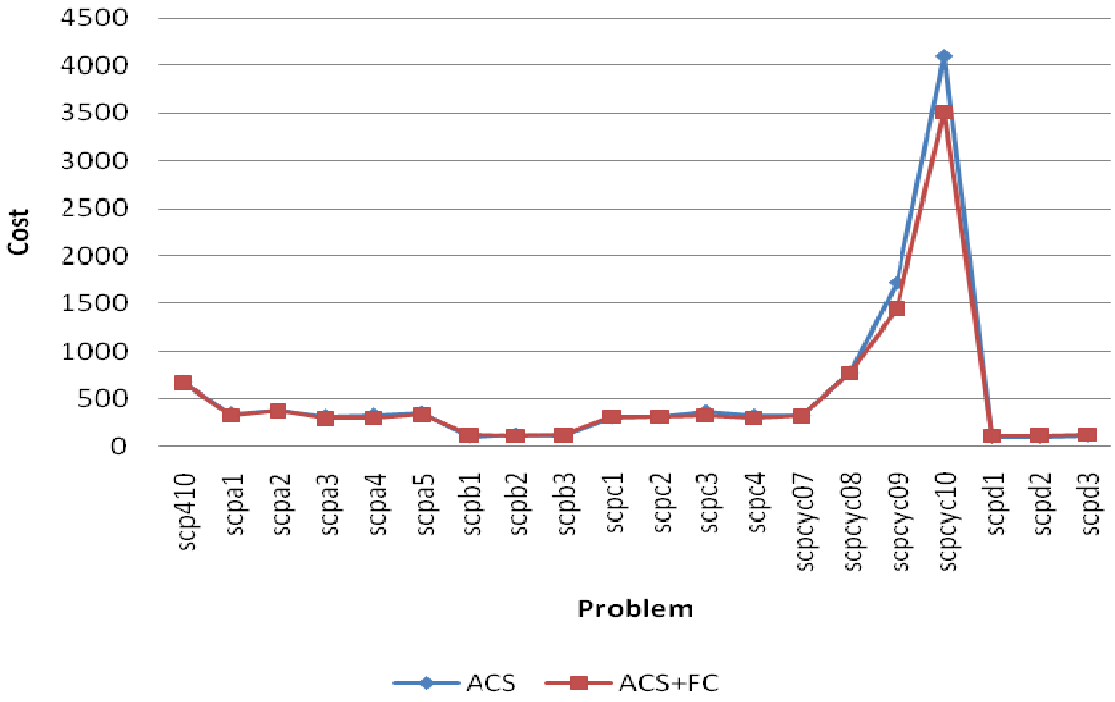

Figure 2. Experimental results for SCP with ACS and ACS+FC.

number of iterations to 150, so the number of generated candidate solutions was limited to 15000. The performance of our previous work was improved due a better parameters setting. For ACS Q_0 = 0.5 and the list size was 300. Algorithms were implemented using ANSI C, GCC 3.3.6, under Microsoft Windows XP Professional version 2002.

DISCUSSION

We solved SCP and SPP using a new ACO transition rule algorithm. Results obtained show that a good idea is to use both incomplete (ACO) and complete (CP) techniques together. In general, when problems are easy enough to allow searching for the optimal solution,

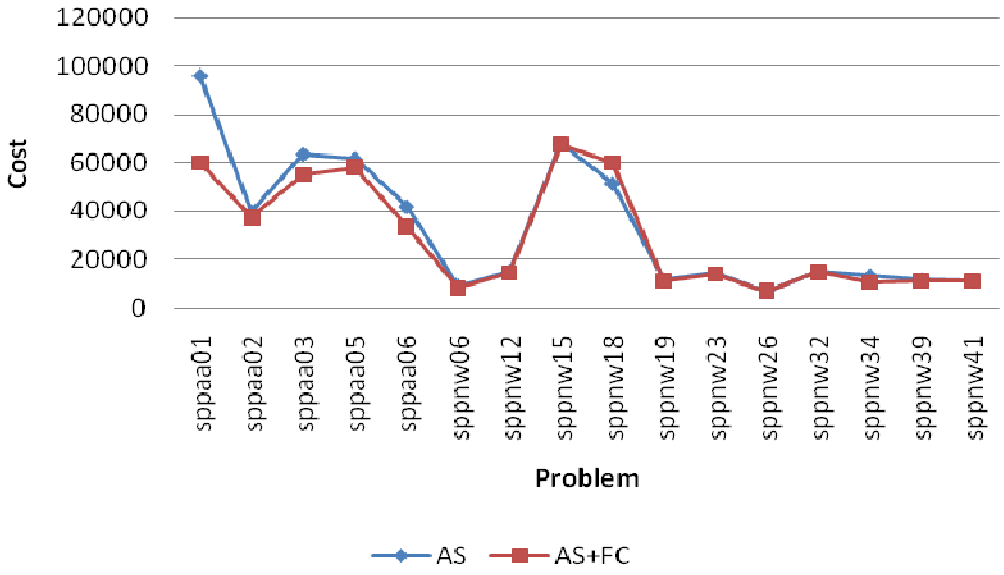

Figure 3. Experimental results for SPP with AS and AS+FC.

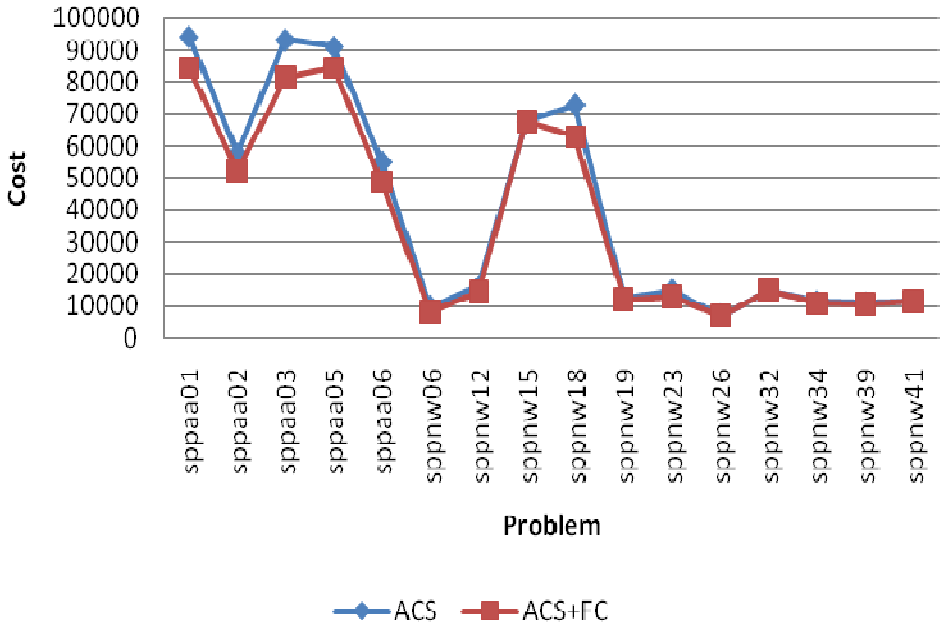

Figure 4. Experimental results for SPP with ACS and ACS+FC.

complete techniques (CP) can be used. When problems become harder, incomplete techniques (ACO) represent a good alternative in order to solve approximately the problem.

The effectiveness of the proposed rule was tested on benchmark problems and the results were compared with pure ACO algorithms.

About efficiency, the computational effort required is almost the same. Ongoing research will investigate a self-tuning parameter proposal.

REFERENCES

Apt K (2003). Principles of Constraint Programming. Cambridge University Press. ISBN: 0521125499 0511062494 9780521125499.

Beasley JE (1990). Or-library: distributing test problems by electronic mail. J. Oper. Res. Soc., 41(11):1069-1072.

Bessiere C (2006). Constraints propagation. In Handbook of Constraint Programming, pp. 29-84.

Crawford B, Castro C, Monfroy E (2006). A hybrid Ant algorithm for the airline crew pairing problem. Lecture Notes in Computer Science. 4293: 381-391.

Dechter R, Frost D (2002). Backjump-based backtracking for constraint satisfaction problems. Artif. Intell., 136(2): 147-188.

Feo T, Resende G (1989). A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett., 8(2): 67-71.
Gagne C, Gravel M, Price W (2001). A Look-Ahead Addition to the Ant Colony Optimization Metaheuristic and its Application to an Industrial Scheduling Problem. In Proceedings of the Fourth Metaheuristics International Conference (MIC'01), pp. 79-84.
Gandibleux X, Delorme X, T'Kindt V (2004). An Ant colony optimisation algorithm for the set packing problem. Lecture Notes in Computer Science. 3172: 49-60.
Khichane M, Albert P, Solnon C (2010). Strong Combination of Ant Colony Optimization with Constraint Programming Optimization. Lecture Notes in Computer Science. 6140: 232-245.
Leguizamon G, Michalewicz Z (1999). A new version of Ant system for subset problems. In Proceedings of Congress on Evolutionary Computation (CEC99). IEEE Press, 3(2): 124-141.
Lessing L, Dumitrescu I, Stutzle T (2004). A comparison between aco algorithms for the set covering problem. Lecture Notes in Computer Science. 3172: 1-12.
Maniezzo V, Milandri M (2002). An Ant -based framework for very strongly constrained problems. Lecture Notes in Computer Science. 2463: 222-227.
Meyer B, Ernst AT (2004). Integrating aco and constraint propagation. Lecture Notes in Computer Science. 3172:166-177.
Michel R, Middendorf M (1998). An island model based Ant system with look ahead for the shortest super sequence problem. Lecture Notes in Computer Science. 1498: 692-701.
Rahoual M, Hadji R, Bachelet V (2002). Parallel Ant system for the set covering problem. Lecture Notes in Computer Science. 2463: 262-267.
Rossi F, Van Beek P, Walsh T (2006). Handbook of Constraint Programming. Elsevier, pp. 29-84.