ACO with Lookahead Procedures for Solving Set Partitioning and Covering Problems

Broderick Crawford [1, 2], Carlos Castro [1]

[1] Departamento de Informática, Universidad Técnica Federico Santa María, Chile

[2] Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, Chile
broderick.crawford@ucv.cl, carlos.castro@inf.utfsm.cl

Abstract. Set Covering Problem and Set Partitioning Problem can model several real life situations. In this paper we solve some benchmarks of them with Ant Colony Optimization algorithms and some hybridizations of them with Constraint Programming techniques. The Lookahead mechanism allows the incorporation of information on the anticipated decisions that are beyond the immediate choice horizon. Computational results are presented showing the advantages to use additional mechanisms to Ant Colony Optimization in strongly constrained problems like Set Partitioning Problem.

Keywords. Ant Colony Optimization (ACO), Forward Checking (FC), Full Lookahead (FLA), Set Covering Problem (SCP), Set Partitioning Problem (SPP).

1 Introduction

Set Covering Problem (SCP) and Set Partitioning Problem (SPP) are two types of problems that can model different real life situations [2, 9]. In this work, we solve some benchmarks of them with Ant Colony Optimization (ACO) algorithms and some hybridizations of ACO with Constraint Programming techniques like Forward Checking and Full Lookahead. There exist problems for which ACO is of limited effectiveness [13]. Among them a prominent role is played by 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. The best performing metaheuristic for SPP is a genetic algorithm due to Chu and Beasley [5, 3]. There exist already some approaches applying ACO to the SCP. In [1, 11], ACO has been used only as a construction algorithm and the approach has only been tested on some small SCP instances. More recent works [10, 12] apply ACO to the SCP and use techniques to remove redundant columns and local search to improve solutions.

In this paper we explore the addition of a lookahead mechanism (usually used in complete techniques) to the ACO algorithm. In Section 2, we describe the applicability of the ACO algorithm for solving SCP and SPP. In Section 3, we present the basic concepts to adding Constraint Programming techniques to the two basic ACO algorithms: Ant System (AS) and Ant Colony System (ACS). In Section 4, we present results when applying these algorithms for solving standard benchmarks from Beasley`s ORLIB. Finally, in Section 5 we conclude the paper and give some perspectives for future research.

2 Ant Colony Optimization for SCP and SPP

In the following description the reader is expected to be familiar with the problems and ACO algorithms proposed in [7, 6]. ACO can be applied in a very straightforward way to the SCP. The columns are chosen as the solution components and have associated a cost and a pheromone trail. Constraints say that each column can be visited by an ant once and only once and that 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 and a heuristic information are associated to each eligible column. A column to be added is chosen with a probability that depends of pheromone trail and the heuristic information. In this paper we use a dynamic heuristic information that depends on the partial solution of an ant. It can be defined as the quotient between the so called cover value, that is, the number of additional rows covered when adding one column to the current partial solution, and the cost of column added [8]. 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. Figure 1 describes two basic ACO algorithms to solve SCP and SPP.

```
Procedure ACO_for_SCP
 Begin
  InitParameters();
   While (remain iterations) do
    For k := 1 to nants do
     While (solution is not completed) do
      AddColumnToSolution(election)
      AddToTabuList(k);
     EndWhile
    EndFor
    UpdateOptimum();
    UpdatePheromone();
   EndWhile
  Return best_solution_founded
 End.
```

```
Procedure ACO_for_SPP
 Begin
  InitParameters();
  While (remain iterations) do
   For k := 1 to nants do
    While (solution is not completed) do
     AddColumnToSolution(election)
     AddToTabuList(k);
     For each Row f covered by k do
      AddToTabuList(column that cover to f);
     EndFor
    EndWhile
   EndFor
   UpdateOptimum();
   UpdatePheromone();
  EndWhile
  Return best_solution_founded
 End.
```

Fig1. ACO algorithms for SCP and SPP

In this work, we use two instances of ACO: Ant System (AS) and Ant Colony System (ACS) algorithms, the original and most famous algorithms in the ACO family [8]. ACS improves the search of AS using: a different transition rule in the constructive phase, exploting the heuristic information in a more rude form, using a list of candidates to future labeling and using a different treatment of pheromone. ACS has demonstrated better performance than AS in a wide range of problems [7].

3 ACO with Constraint Programming

Recently, some efforts have been done in order to integrate Constraint Programming techniques to ACO algorithms [14]. Forward Checking (FC) seems to be the easiest way to prevent future conflicts. Instead of performing arc consistency to the instantiated variables, it performs a restricted form of arc consistency to the not yet instantiated variables. This reduces the search tree and the overall amount of work done. But it should be noted that Forward Checking does more work when each assignment is added to the current partial solution. Adding Forward Checking to ACO means that columns are chosen if they do not produce any conflict with respect to the next column to be chosen. Forward checking checks only the constraints between the current variable and the future variables. So, why not to perform full arc consistency that will further reduces the domains and removes possible conflicts. This approach is called Full Lookahead (FLA) or maintaining arc consistency. The advantage of look ahead is that it detects also the conflicts between future variables and therefore allows branches of the search tree that will lead to failure to be pruned earlier than with Forward Checking. Adding Full Lookahead to ACO means that columns are chosen using recursively the same ideas that Forward Checking and so we detect conflicts before a solution is completed.

4 Experiments and Results

Table 1 presents results when adding FC and FLA techniques to the basic ACO algorithms for solving standard benchmarks taken from the Beasley`s ORLIB. The first four columns present the problem code, the number of rows, the number of columns, and the best known solution for each instance, respectively. The remainder columns present the cost obtained when applying the algorithms. Considering several tests and published experimental results [8, 12, 11] we use the following parameters for the algorithms: *?* = 0.4, number of iterations = 160, number of ants = 120, *ß* = 0.5, for ACS the list was = 500 (in scp41, scp42, scp48, scp61, scp62, and scp63), for ACS Q_o = 0.5. Algorithms were implemented using ANSI C, GCC 3.3.6, under Microsoft Windows XP Professional version 2002.

Table 1. Results (Cost obtained)

Problem	Rows	Columns	Optimum	AS	AS+FC	AS+FLA	ACS	ACS+FC	ACS+FLA
sppnw39	25	677	10080	11670	11322	10722	10758	10545	11322
sppnw34	20	899	10488	13341	10713	10713	11289	10797	10713
sppnw26	23	771	6796	6976	6880	7192	6956	6880	6850
sppnw23	19	711	12534	14304	13932	13254	14604	12880	13400
scp41	200	1000	429	473	458	2115	463	683	842
scp42	200	1000	512	594	574	1990	590	740	752
scp48	200	1000	492	524	537	1952	522	731	752
scp51	200	1000	253	289	289	1975	280	464	526
scp61	200	1000	138	157	155	1081	154	276	352
scp62	200	1000	146	169	170	1004	163	280	352
scp63	200	1000	145	161	161	763	157	209	267

The effectiveness of Constraint Programming is showed to the SPP and in some instances of SCP solving with AS+FC. The strongly constrained problem characteristic of SPP does the stochastic behavior of ACO improved with FC techniques in the construction phase, so that almost only feasible solutions are induced. In the original ACO implementation the SPP solving derives in a lot of unfeasible labeling of variables, and many ants can not complete solutions. For SCP, the huge size of the search space and the relaxation of the constraints do original ACO algorithms work better than ACO with Constraint Programming considering the same execution conditions.

5 Conclusions and Future Directions

Computational results confirm that the performance of ACO is possible to improve with some classes of hybridization. In a restricted problem like SPP, ACO with Lookahead techniques showed an interesting performance, we demonstrated that this integration improves the process, mainly with respect to success costs instead running times. But the trade off in all cases is very convenient. The ACO metaheuristic can produce good quality solutions to this class of problems if it is applied correctly.

Future versions of the algorithm can study the pheromone representation and will try to incorporate it into look ahead techniques. Furthermore, considering that the ant´s solutions may contain redundant components which can be eliminated by a fine tuning after the solution, then we will explore Post Processing procedures too, which consist in the identification and replacement of the columns of the ACO solution in each iteration by more effective columns. Part of this involves the study of available local search techniques in order to reduce the input problem and improve the solutions given by the ants.

References

1. D. Alexandrov and Y. Kochetov. The behavior of the ant colony algorithm for the set covering problem. In *Operations Research*, pp 255–260. SV, 1999.
2. E. Balas and M. Padberg. Set Partitioning: A Survey. *SIAM Review*, 18:710–760, 1976.
3. J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404, 1996.
4. M. J. Brusco, L. W. Jacobs, and G. M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost and coverage correlated set covering problems. *Annals of Operations Research*, 86:611–627, January 1999.
5. P. C. Chu and J. E. Beasley. Constraint handling in genetic algorithms: the set partitoning problem. *Journal of Heuristics*, 4:323–357, 1998.
6. M. Dorigo, G. D. D. Caro, and L. M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5:137–172, 1999.
7. M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
8. M. Dorigo and T. Stutzle. *Ant Colony Optimization*. MIT Press, USA, 2004.
9. A. Feo, G. Mauricio, and A. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *OR Letters*, 8:67–71, 1989.
10. R. Hadji, M. Rahoual, E. Talbi, and V. Bachelet. Ant colonies for the set covering problem. In M. D. et al., editor, *ANTS 2000*, pp 63–66, 2000.
11. G. Leguizamón and Z. Michalewicz. A new version of Ant System for subset problems. In *Congress on Evolutionary Computation*, *CEC'99*, pp 1459–1464, Piscataway, NJ, USA, 1999. IEEE Press.
12. L. Lessing, I. Dumitrescu, and T. Stutzle. A Comparison Between ACO Algorithms for the Set Covering Problem. In M. D. et al., editor, *ANTS 2004*, vol 3172 of *LNCS*, pp 1–12. SV, 2004.
13. V. Maniezzo and M. Milandri. An Ant-Based Framework for Very Strongly Constrained Problems. In M. D. et al., editor, *ANTS 2002*, vol 2463 of *LNCS*, pp 222–227. SV, 2002.
14. B. Meyer and A. Ernst. Integrating ACO and Constraint Propagation. In M. D. et al., editor, *ANTS 2004*, vol 3172 of *LNCS*, pp 166–177. SV, 2004.