A hybrid Ant algorithm for the set covering problem

Broderick Crawford\textsuperscript{1,2,*}, Ricardo Soto\textsuperscript{1,3}, Eric Monfroy\textsuperscript{2,4}, Fernando Paredes\textsuperscript{5} and Wenceslao Palma\textsuperscript{1}

\textsuperscript{1}Pontificia Universidad Católica de Valparaíso, Chile.
\textsuperscript{2}Universidad Técnica Federico Santa María, Chile.
\textsuperscript{3}Universidad Autónoma de Chile, Chile.
\textsuperscript{4}CNRS, LINA, Université de Nantes, France.
\textsuperscript{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 instantiation 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
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 uninstantiated
      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.

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 \(x_i\) 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).

\[
f(x) = \sum_{j=1}^{n} c_jx_j \quad (1)
\]

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

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

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).
in the worst case, it is possible to
in the iterative steps, it is possible to
in the ACO family. Generally ACS improves the search of
AS by using: a different transition rule in the constructive
algorithms, the original and the most famous algorithms
Ant system (AS) and Ant colony system (ACS)
problem constraint when a variable is instantiated. And
labeling and a different treatment of the pheromone.
phase, exploiting the heuristic information in a more rude
instances (Crawford et al., 2006). Each Ant starts with an
incapable of obtaining feasible solution for many 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 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 $\beta = 0.5$ and evaporation rate $\rho = 0.4$.

The number of Ants was set to 100 and the maximum

$$p_j^k(t) = \frac{\tau_j(\eta_j)^\beta}{\sum_{i\in S^k} \tau_i(\eta_i)^\beta} \quad \text{if} \quad j \notin S^k \quad \text{(4)}$$

$$\eta_j = \frac{e_j}{c_j} \quad \text{(5)}$$

A pheromone trail $\tau_j$ and a heuristic information $\eta_j$ 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 $\beta$ parameter controls how
important $\eta$ is in the probabilistic decision. In this work,
the pheromone trail $\tau_j$ 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 $e_j$ 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 $j$.

Algorithm 1 describes the basic structure of ACO
game 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 instantiated. And
in the worst case, in the iterative steps, it is possible to
### Table 1. Experimental results of SCP benchmarks.

<table>
<thead>
<tr>
<th>Problem</th>
<th>m</th>
<th>n</th>
<th>Opt</th>
<th>AS</th>
<th>ACS</th>
<th>AS+FC</th>
<th>ACS+FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>scp410</td>
<td>200</td>
<td>1000</td>
<td>514</td>
<td>539</td>
<td>669</td>
<td>556</td>
<td>664</td>
</tr>
<tr>
<td>scp41</td>
<td>300</td>
<td>3000</td>
<td>253</td>
<td>592</td>
<td>348</td>
<td>288</td>
<td>331</td>
</tr>
<tr>
<td>scp42</td>
<td>300</td>
<td>3000</td>
<td>252</td>
<td>531</td>
<td>378</td>
<td>285</td>
<td>376</td>
</tr>
<tr>
<td>scp43</td>
<td>300</td>
<td>3000</td>
<td>232</td>
<td>473</td>
<td>319</td>
<td>270</td>
<td>295</td>
</tr>
<tr>
<td>scp44</td>
<td>300</td>
<td>3000</td>
<td>234</td>
<td>375</td>
<td>333</td>
<td>278</td>
<td>301</td>
</tr>
<tr>
<td>scp45</td>
<td>300</td>
<td>3000</td>
<td>236</td>
<td>349</td>
<td>353</td>
<td>272</td>
<td>335</td>
</tr>
<tr>
<td>scp61</td>
<td>300</td>
<td>3000</td>
<td>69</td>
<td>196</td>
<td>101</td>
<td>75</td>
<td>115</td>
</tr>
<tr>
<td>scp62</td>
<td>300</td>
<td>3000</td>
<td>80</td>
<td>207</td>
<td>112</td>
<td>89</td>
<td>117</td>
</tr>
<tr>
<td>scp63</td>
<td>300</td>
<td>3000</td>
<td>227</td>
<td>442</td>
<td>305</td>
<td>261</td>
<td>317</td>
</tr>
<tr>
<td>scp64</td>
<td>300</td>
<td>3000</td>
<td>219</td>
<td>484</td>
<td>309</td>
<td>260</td>
<td>311</td>
</tr>
<tr>
<td>scp65</td>
<td>300</td>
<td>3000</td>
<td>243</td>
<td>551</td>
<td>367</td>
<td>268</td>
<td>328</td>
</tr>
<tr>
<td>scp64</td>
<td>300</td>
<td>3000</td>
<td>219</td>
<td>523</td>
<td>324</td>
<td>259</td>
<td>303</td>
</tr>
<tr>
<td>scp67</td>
<td>672</td>
<td>448</td>
<td>144</td>
<td>272</td>
<td>321</td>
<td>148</td>
<td>321</td>
</tr>
<tr>
<td>scp68</td>
<td>1792</td>
<td>1024</td>
<td>344</td>
<td>512</td>
<td>769</td>
<td>364</td>
<td>769</td>
</tr>
<tr>
<td>scp69</td>
<td>4608</td>
<td>2304</td>
<td>780</td>
<td>1297</td>
<td>1723</td>
<td>816</td>
<td>1445</td>
</tr>
<tr>
<td>scp70</td>
<td>11520</td>
<td>5120</td>
<td>1792</td>
<td>3123</td>
<td>4097</td>
<td>1969</td>
<td>3506</td>
</tr>
<tr>
<td>scp61</td>
<td>400</td>
<td>4000</td>
<td>60</td>
<td>184</td>
<td>92</td>
<td>72</td>
<td>105</td>
</tr>
<tr>
<td>scp62</td>
<td>400</td>
<td>4000</td>
<td>66</td>
<td>209</td>
<td>96</td>
<td>74</td>
<td>113</td>
</tr>
<tr>
<td>scp63</td>
<td>400</td>
<td>4000</td>
<td>72</td>
<td>221</td>
<td>111</td>
<td>83</td>
<td>119</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>m</th>
<th>n</th>
<th>Opt</th>
<th>AS</th>
<th>ACS</th>
<th>AS+FC</th>
<th>ACS+FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>sppa01</td>
<td>8233</td>
<td>8904</td>
<td>56137</td>
<td>96256</td>
<td>94270</td>
<td>60246</td>
<td>84435</td>
</tr>
<tr>
<td>sppa02</td>
<td>531</td>
<td>5198</td>
<td>30494</td>
<td>39883</td>
<td>57632</td>
<td>37452</td>
<td>52211</td>
</tr>
<tr>
<td>sppa03</td>
<td>825</td>
<td>8627</td>
<td>49649</td>
<td>63734</td>
<td>93304</td>
<td>55082</td>
<td>81177</td>
</tr>
<tr>
<td>sppa05</td>
<td>801</td>
<td>8308</td>
<td>55839</td>
<td>61703</td>
<td>91134</td>
<td>58158</td>
<td>84362</td>
</tr>
<tr>
<td>sppa06</td>
<td>646</td>
<td>7292</td>
<td>27040</td>
<td>42015</td>
<td>54964</td>
<td>33524</td>
<td>48703</td>
</tr>
<tr>
<td>sppnw06</td>
<td>50</td>
<td>6774</td>
<td>7810</td>
<td>9200</td>
<td>9788</td>
<td>8160</td>
<td>8038</td>
</tr>
<tr>
<td>sppnw08</td>
<td>24</td>
<td>434</td>
<td>35894</td>
<td>x</td>
<td>x</td>
<td>35894</td>
<td>36682</td>
</tr>
<tr>
<td>sppnw09</td>
<td>40</td>
<td>3103</td>
<td>67760</td>
<td>70462</td>
<td>x</td>
<td>70222</td>
<td>69332</td>
</tr>
<tr>
<td>sppnw10</td>
<td>24</td>
<td>853</td>
<td>68271</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>sppnw12</td>
<td>27</td>
<td>626</td>
<td>14118</td>
<td>15406</td>
<td>16060</td>
<td>14466</td>
<td>14252</td>
</tr>
<tr>
<td>sppnw15</td>
<td>31</td>
<td>467</td>
<td>67743</td>
<td>67755</td>
<td>67746</td>
<td>67743</td>
<td>67743</td>
</tr>
<tr>
<td>sppnw18</td>
<td>124</td>
<td>10757</td>
<td>51624</td>
<td>51624</td>
<td>73006</td>
<td>60224</td>
<td>62832</td>
</tr>
<tr>
<td>sppnw19</td>
<td>40</td>
<td>2879</td>
<td>10898</td>
<td>11678</td>
<td>12350</td>
<td>11060</td>
<td>11858</td>
</tr>
<tr>
<td>sppnw23</td>
<td>19</td>
<td>711</td>
<td>12534</td>
<td>14304</td>
<td>14604</td>
<td>13932</td>
<td>12880</td>
</tr>
<tr>
<td>sppnw26</td>
<td>23</td>
<td>771</td>
<td>6796</td>
<td>6976</td>
<td>6956</td>
<td>6880</td>
<td>6880</td>
</tr>
<tr>
<td>sppnw32</td>
<td>19</td>
<td>294</td>
<td>14877</td>
<td>14877</td>
<td>14886</td>
<td>14877</td>
<td>14877</td>
</tr>
<tr>
<td>sppnw34</td>
<td>20</td>
<td>899</td>
<td>10488</td>
<td>13341</td>
<td>11289</td>
<td>10713</td>
<td>10797</td>
</tr>
<tr>
<td>sppnw39</td>
<td>25</td>
<td>677</td>
<td>10080</td>
<td>11670</td>
<td>10758</td>
<td>11322</td>
<td>10545</td>
</tr>
<tr>
<td>sppnw41</td>
<td>17</td>
<td>197</td>
<td>11307</td>
<td>11307</td>
<td>11307</td>
<td>11307</td>
<td>11307</td>
</tr>
</tbody>
</table>

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.
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 to 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,
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


