A Beam-Search Approach to the Set Covering Problem

Victor Reyes, Ignacio Araya, Broderick Crawford, Ricardo Soto and Eduardo Olguín

Abstract In this work we present a beam-search approach applied to the Set Covering Problem. The goal of this problem is to choose a subset of columns of minimal cost covering every row. Beam Search constructs a search tree by using a breadth-first search strategy, however only a fixed number of nodes are kept and the rest are discarded. Even though original beam search has a deterministic nature, our proposal has some elements that makes it stochastic. This approach has been tested with a well-known set of 45 SCP benchmark instances from OR-Library showing promising results.

Keywords SCP · Beam search · Branch-and-Bound · Greedy

V. Reyes (✉) · I. Araya · B. Crawford · R. Soto
Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile
e-mail: vareyesrod@gmail.com

I. Araya
e-mail: ignacio.araya@ucv.cl

B. Crawford
e-mail: broderick.crawford@ucv.cl

R. Soto
e-mail: ricardo.soto@ucv.cl

B. Crawford · E. Olguín
Universidad San Sebastián, Santiago Metropolitan Region, Chile

B. Crawford
Universidad Central de Chile, Santiago Metropolitan Region, Chile

R. Soto
Universidad Autónoma de Chile, Temuco, Chile

R. Soto
Universidad Cientifica Del Sur, Lima, Peru

R. Silhavy et al. (eds.), *Artificial Intelligence Perspectives in Intelligent Systems*,
Advances in Intelligent Systems and Computing 464,
DOI 10.1007/978-3-319-33625-1_35

1 Introduction

The Set Covering Problem (SCP) is a combinatorial problem that can be described as the problem of finding a subset of columns from a *m-row*, *n-column* zero-one matrix a_{ij} such that they can cover all the rows at minimal cost. The SCP can be formulated as follows:

$$Minimize\, Z = \sum_{j=1}^{n} c_j x_j \qquad j \in \{1, 2, 3, ..., n\} \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i \in \{1, 2, 3, ..., m\} \tag{2}$$

$$x_j \in \{0, 1\}, \tag{3}$$

where c_j represents the vector cost. The SCP is a NP-hard problem [9] that has been used to model many problems as scheduling, manufacturing, services planning, information retrieval, etc. [1, 7]. Several algorithms have been developed for solving SCP instances. *Exact algorithms* [6], even though they can reach the global optima, they require substantial time for solving large instances. *Greedy algorithms* [8] are a good approach for large instances, but rarely generates good solutions because of its myopic and deterministic nature. Another approach are *Probabilistic greedy algorithms* [10, 13], which often generates better quality solutions than the deterministic counterparts. *Metaheuristics* are commonly the best way to solve large SCP instances, some of them are: Genetic algorithms [3, 18], Neural Network algorithms [16], Simulated Annealing [11], Ant Colony Optimization [14], and many more.

In this work, we propose an algorithm for solving the SCP that is based in the well known beam-search algorithm. It has been used in many optimization problems [4, 5, 12, 19]. Beam-search is a fast and approximate branch and bound method, which operates in a limited search space to find good solutions for optimization problems. It constructs a search tree by using a breadth-first search, but selecting only the most promising nodes by using some rule. Our implementation selects these nodes using a simple greedy algorithm that can be seen as a Depth-first search. The greedy will find a solution and returns its fitness, which will be used to select and discard nodes from the search tree.

This paper is organized as follows: Sect. 2 describes our Beam-Search implementation for the SCP, Sect. 3 shows the result that we obtained by using a well known set of SCP benchmarks instances, finally conclusions and future work can be found in Sect. 4.

2 Beam Search

Beam Search [15] is a deterministic heuristic algorithm that constructs a search-tree. It begins with an empty solution at the root node and gradually construct solution candidates, level by level. At each level of the tree, two procedures are applied: *PromisingChildren* and *SelectBest*. While the first one expand each node by the n_p most promising children using some criteria, the second one choose the n_s most promising nodes from the current level. Given this, at the *level 0* the tree will have one node; at the *level 1* n_s nodes; from the *level 2* the algorithm will select n_s nodes from a pool of at most $n_s * n_p$ nodes. Beam Search lacks of completeness, because the optimal solution could be pruned during the search process. The Algorithm 1 corresponds to the classic beam-search described before.

Algorithm 1 `Original Beam-Search`(n_p, n_s, P); **out:** *Solution*

```
S ← {emptySolution}
Solution ← NULL
while S ≠ ∅ do
  S' ← { }
  for all s ∈ S do
    S' ← S' ∪ PromisingChildren(s, n_s, P)
  end for
  if is − solution(S') then
    Return(Solution)
  end if
  S ← SelectBest(S', n_s)
end while
```

2.1 Our Implementation

For adapting this algorithm to the SCP, we consider the following: *PromisingChildren* determinates the n_p most promising children from the current node. This is achieved by calculating, for each non-instantiated variable, a value using one of the following functions: c_j/k_j, c_j/k_j^2, $c_j/(k_j \log(1+k_j))$, $c_j^{1/2}/k_j$, $c_j/k_j^{1/2}$ and $c_j/\log(k_j+1)$ [8, 13]. The variable k_j represents the number of currently uncovered rows that could be covered by the column j. The function is selected in a random way and it is used for all the nodes of the current level. Then, the n_p variables with the lowest values are instanciated. After that, we run a greedy algorithm for each of the new candidates nodes by using the procedure *Greedy-SelectBest*. This greedy attempts to construct a branch (one node per level), using the same function selected in *PromisingChildren*, until a solution is reached. At the end of this process, each

node will have an associate solution. The procedure will select the n_s nodes with the best objective function value. The best solution founded in the search it is used to discard nodes with a worst objetive function value.

Unlike the classic algorithm, the search does not stop when a solution is founded or all nodes are discarded, instead, we set a fixed number of nodes to be generated (See Algorithm 2).

Algorithm 2 `Beam-Search+Greedy`(n_p, n_s, P); **out:** $Best - Solution$

$S \leftarrow \{emptySolution\}$
$Best - Solution \leftarrow NULL$
while `FixedNumberOfNodesReached` **do**
  $S' \leftarrow \{\}$
  for all $s \in S$ **do**
    $S' \leftarrow S' \cup$ `PromisingChildren`$(s, n_s, P, BestSolution)$
  end for
  $S \leftarrow$ `Greedy` $-$ `SelectBest`$(S', n_s, BestSolution)$
end while

2.2 Preprocessing

Preprocessing is a popular method to speedup the algorithm. A number of preprocessing methods have been proposed for the SCP [2]. In our implementation, we used the most effective ones:

- Column domination: Any column j whose rows I_j can be covered by other columns for a cost less than c_j can be deleted from the problem, however this is an NP complete problem [9]. Instead, we used the rule described in [17].
- Column inclusion: If a row is covered by only one column after the above domination, this column must be included in the optimal solution.

3 Experiments

Our approach has been implemented in `C++`, on an 2.4GHz CPU Intel Core i7-4700MQ with 8gb RAM computer using Ubuntu 14.04 LTS x86_64. In order to test it, we used 45 SCP instances from OR-Library[1] which are described in Table 1. Optimal solutions are known for all of these instances.

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html.

Table 1 Detail of the test instances

Instance set	No. of instances	Rows	Columns	Cost range
4	10	200	1000	[1, 100]
5	10	200	2000	[1, 100]
6	5	200	1000	[1, 100]
A	5	300	3000	[1, 100]
B	5	300	3000	[1, 100]
C	5	400	4000	[1, 100]
D	5	400	4000	[1, 100]

Table 2 Experiments using $n_p = 20$ and $n_s = 10$

Instance	Optima	Min-value	Max-value	Avg	RPD	Instance	Optima	Min-value	Max-value	Avg	RPD
scp41	429	430	434	432.0	0.23	scpA1	253	256	259	257.3	1.19
scp42	512	517	527	524.8	0.98	scpA2	252	257	263	262.1	1.98
scp43	516	520	530	525.9	0.78	scpA3	232	238	240	238.6	2.59
scp44	494	501	510	504.9	1.42	scpA4	234	236	241	238.7	0.85
scp45	512	515	525	521.6	0.59	scpA5	236	236	239	237.6	0.00
scp46	560	570	576	572.4	1.79	scpB1	69	69	78	75.1	0.00
scp47	430	432	435	433.6	0.47	scpB2	76	76	81	78.0	0.00
scp48	492	493	498	495.2	0.20	scpB3	80	80	82	80.5	0.00
scp49	641	658	667	662.7	2.65	scpB4	79	79	82	81.0	0.00
scp410	514	514	519	517.3	0.00	scpB5	72	72	73	72.1	0.00
scp51	253	256	262	259.9	1.19	scpC1	227	234	237	235.8	3.08
scp52	302	308	313	309.8	1.99	scpC2	219	222	230	227.0	1.37
scp53	226	230	234	233.4	1.77	scpC3	243	244	251	248.5	0.41
scp54	242	243	244	243.6	0.41	scpC4	219	223	235	234.0	1.83
scp55	211	215	219	217.8	1.90	scpC5	215	215	217	215.5	0.00
scp56	213	213	219	216.9	0.00	scpD1	60	60	61	60.2	0.00
scp57	293	298	303	301.1	1.71	scpD2	66	68	70	68.6	3.03
scp58	288	291	298	294.7	1.04	scpD3	72	74	75	74.3	2.78
scp59	279	282	287	285.0	1.08	scpD4	62	62	63	62.3	0.00
scp510	265	265	271	268.4	0.00	scpD5	61	61	64	62.9	0.00
scp61	138	140	143	141.9	1.45						
scp62	146	148	150	149.1	1.37						
scp63	145	149	151	149.7	2.76						
scp64	131	132	133	132.5	0.76						
scp65	161	165	170	167.1	2.48						

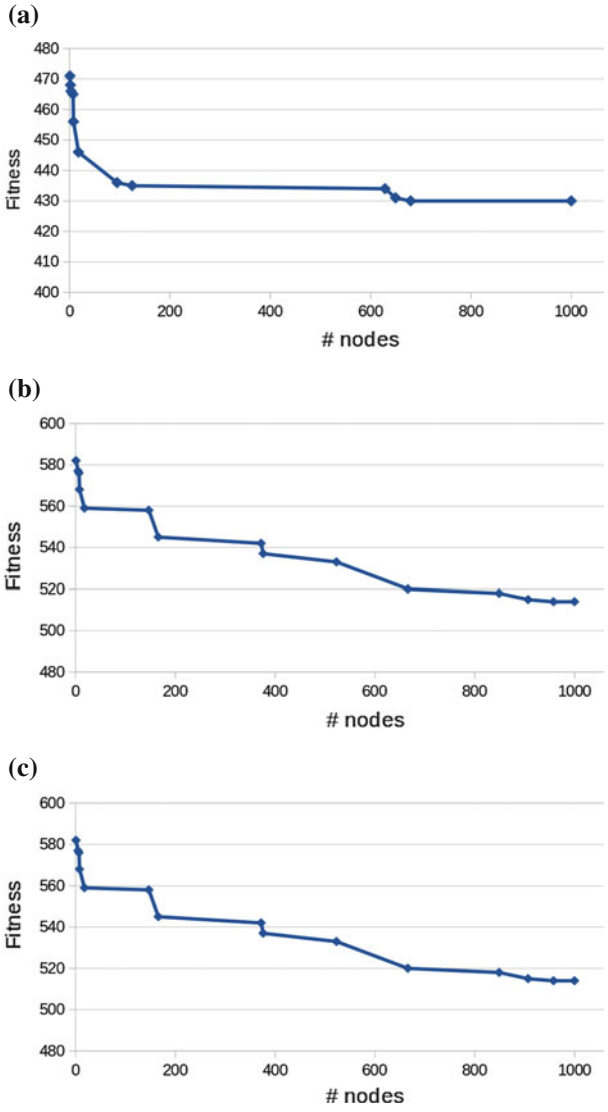

Fig. 1 Convergence plots for the **a** scp41, **b** scp42 and **c** scp43 instances

Our algorithm was configured before perform the search. Each of these instances were executed 20 times, with several values of n_p and n_s. The best results (related to the avg. value) were obtained by using $n_p = 20$ and $n_s = 10$. We set as stop criteria a maximum of 1000 nodes in the search tree. After reaching this value, the algorithm did not show a big improvement in the solutions. Table 2 shows the results by using this configuration.

The column *Optima* represents the lowest objective function value for a particular instance. *Min-value* and *Max-value* represent the lowest and the maximum objective function value, respectively, obtained for our proposal in 20 executions. The mean value of these 20 executions are shown in the column *Avg*. The column *RPD* represents the Relative Percent Difference. This measure can be defined as follows:

$$RPD = \frac{(\text{Min-value} - \text{Optima})}{\text{Optima}} \times 100. \tag{4}$$

Convergence plots can be seen in Fig. 1.

4 Conclusion and Future Work

In this work we have presented a beam-search approach with a greedy algorithm to solve the SCP. Our approach applies a greedy algorithm in each node to find solutions by using a set of simple functions that choose promising variables. Experiments show very promising results, considering that the technique in not yet fully exploited. In a future work we plan to do a more guided search by using a nogood-like learning strategy,[2] that should reduce the size of the search tree. Also, we plan to adapt this technique for the bi-objective SCP formulation.

Acknowledgments Victor Reyes is supported by grant INF-PUCV 2015, Ricardo Soto is supported by grant CONICYT/FONDECYT/INICIACION/11130459, Broderick Crawford is supported by grant CONICYT/FONDECYT/REGULAR/1140897, and Ignacio Araya is supported by grant CONICYT/FONDECYT/INICIACION/11121366.

References

1. Balas, E., et al.: A class of location, distribution and scheduling problems: modeling and solution methods (1982)
2. Beasley, J.E.: An algorithm for set covering problem. Eur. J. Oper. Res. **31**(1), 85–93 (1987)
3. Beasley, J.E., Chu, P.C.: A genetic algorithm for the set covering problem. Eur. J. Oper. Res. **94**(2), 392–404 (1996)
4. Bennell, J.A., Song, X.: A beam search implementation for the irregular shape packing problem. J. Heuristics **16**(2), 167–188 (2010)
5. Blum, C.: Beam-acohybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput. Oper. Res. **32**(6), 1565–1591 (2005)
6. Caprara, A., Toth, P., Fischetti, M.: Algorithms for the set covering problem. Ann. Oper. Res. **98**(1–4), 353–371 (2000)
7. Ceria, S., Nobili, P., Sassano, A.: A lagrangian-based heuristic for large-scale set covering problems. Math. Program. **81**(2), 215–228 (1998)
8. Chvatal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. **4**(3), 233–235 (1979)

[2] also known as cutting planes.

9. Michael, R.G., David, S.J.: Computers and intractability: a guide to the theory of np-completeness. San Francisco, p. 1979. Freeman, LA (1979)
10. Haouari, M, Chaouachi, J.S.: A probabilistic greedy search algorithm for combinatorial optimisation with application to the set covering problem. J. Oper. Res. Soc. 792–799 (2002)
11. Jacobs, L.W., Brusco, M.J.: Note: a local-search heuristic for large set-covering problems. Nav. Res. Logist. (NRL) **42**(7), 1129–1140 (1995)
12. Kim, K.H., Kang, J.S., Ryu, K.R.: A beam search algorithm for the load sequencing of outbound containers in port container terminals. OR Spectr. **26**(1), 93–116 (2004)
13. Lan, G., DePuy, G.W., Whitehouse, G.E.: An effective and simple heuristic for the set covering problem. Eur. J. Oper. Res. **176**(3), 1387–1403 (2007)
14. Lessing, L., Dumitrescu, I., Stützle, T.: A comparison between aco algorithms for the set covering problem. Ant Colony Optimization and Swarm Intelligence, pp. 1–12. Springer, Berlin (2004)
15. Norvig, P.: Paradigms of Artificial Intelligence Programming: Case Studies in Common LISP. Morgan Kaufmann (1992)
16. Ohlsson, M., Peterson, C., Söderberg, B.: An efficient mean field approach to the set covering problem. Eur. J. Oper. Res. **133**(3), 583–595 (2001)
17. Ren, Z.-G., Feng, Z.-R., Ke, L.-J., Zhang, Z.-J.: New ideas for applying ant colony optimization to the set covering problem. Comput. Ind. Eng. **58**(4), 774–784 (2010)
18. Solar, M., Parada, V., Urrutia, R.: A parallel genetic algorithm to solve the set-covering problem. Comput. Oper. Res. **29**(9), 1221–1235 (2002)
19. Wang, F., Lim, A.: A stochastic beam search for the berth allocation problem. Decis. Support Syst. **42**(4), 2186–2196 (2007)