ICIC Express Letters
Volume **10**, Number **4**, April **2016**
 ISSN 1881-803X
pp. **799–805**

A SIMULATED ANNEALING BASED MULTISTART HEURISTIC FOR THE SET COVERING PROBLEM

Lingxiao Xue[1] and Geng Lin[2,*]

[1]College of Computer and Information
Fujian Agriculture and Forestry University
No. 15, Shangxiadian Road, Cangshan District, Fuzhou 350002, P. R. China
xuelingxiao@fafu.edu.cn

[2]Department of Mathematics
Minjiang University
No. 200, Xiyuangong Road, Shangjie, Minhou, Fuzhou 350108, P. R. China
*Corresponding author: lingeng413@163.com

Received October 2015; accepted January 2016

Abstract. *The set covering problem is an NP-hard combinatorial optimization problem with lots of applications. In this paper, we propose a simulated annealing based multistart heuristic (SAMSH) for the set covering problem. The SAMSH uses an adaptive memory mechanism to generate initial solutions. It guides the search toward promising region. Then, a simulated annealing procedure is developed to improve the initial solutions. The proposed algorithm is tested on a set of 45 benchmark instances from the literature. Experimental results and comparisons show that the proposed algorithm is able to get high quality solutions.*
Keywords: Set covering problem, Simulated annealing, Heuristic, Combinatorial optimization

1. **Introduction.** The set covering problem (SCP) is a classical NP-hard problem of combinatorial optimization [1]. It seeks to cover the rows of an m-row, n-column, zero-one matrix (a_{ij}) by a subset of the columns at minimal cost. The SCP can be formulated as follows [2]:

$$(SCP)\begin{cases} \min & f(x)=\sum_{j\in J} c_j x_j, \\ \text{s.t.} & \sum_{j\in J} a_{ij}x_j \geq 1, \quad \text{for } i\in I, \\ & x_j \in \{0,1\}, \quad \text{for } j\in J, \end{cases}$$

where $J=\{1,\cdots,n\}$, and $I=\{1,\cdots,m\}$ are the sets of columns and rows, respectively. c_j is the cost of column j. $x_j \in \{0,1\}$ is the binary decision variable indicating if column j is selected, or not.

As the wide practical applications of the SCP [6, 7], it has received a great deal of attention in the past two decades. Several exact methods for the SCP, such as branch and bound [8], and Gomory f-cuts [9], are developed to produce optimal solutions. Due to the NP-hardness of the SCP, exact algorithms are difficult to produce high quality solutions with a reasonable computational effort for problems of a moderate size. For larger instances, many researches have been interested in applying heuristic methods including genetic algorithm [2, 3], cultural algorithm [4], binary firefly algorithm [5], the 3-flip neighborhood local search method [10], electromagnetism metaheuristic [11], and row weighting local search algorithm [12], to getting near optimal solutions.

Simulated annealing (SA) was introduced by Kirkpatrick et al. [13]. It is a local search based heuristic capable of escaping from local optimum by accepting, with small probability, worse solutions during the search process. SA has been successfully applied to a lot of hard combinatorial optimization problems [13, 14, 15] to produce high quality solutions.

Inspired by the potential application of SA to solve combinatorial optimization problems, this paper proposes a simulated annealing based multistart heuristic (SAMSH) for solving the set covering problem. Extensive experiments are done on a set of 45 instances. The computational results are compared with other existing heuristics, including genetic algorithm, culture algorithm, and binary firefly algorithm. The comparisons show that the SAMSH is very efficient.

This paper is organized as follows. The details of the proposed algorithm are given in Section 2. Experimental results and comparisons are provided in Section 3. Finally, we give our conclusions in Section 4.

2. The Multistart SA Algorithm for the SCP.

2.1. The SA procedure.

In this section, a simulated annealing based multistart heuristic is presented. The proposed algorithm follows the standard SA procedure which is based on the definitions of fitness function and neighborhood. During the search process, we use the exact penalty function $g(x)$ to evaluate the quality of an obtained solution x. More formally,

$$g(x) = f(x) + \lambda \times \omega(x), \tag{1}$$

where $\lambda > 0$ is a penalty parameter, and

$$\omega(x) = \sum_{i \in I} \max\left(1 - \sum_{j \in J} a_{ij} x_j, 0\right)$$

is the number of rows which are not covered by x. Denote the set of neighboring solutions of the current solution x as $N(x)$. We define $N(x)$ as the set of solutions obtained by adding an unselected column or removing a selected column, i.e., each time one variable is flipping. More formally,

$$N(x) = \left\{ y : \sum_{j \in J} |x_j - y_j| \leq 1 \right\}. \tag{2}$$

Let x^* and x^{best} be the current best solution found so far, and the best solution obtained by an SA procedure, respectively. Our SA procedure starts from an initial solution x. The pseudo code of this procedure is given in Algorithm 1. Initially, the current temperature T is set to T_0 (line 1). At each iteration, a neighborhood solution y in $N(x)$ is selected by flipping one variable. We choose a variable to flip according to its move value. The move value $v(j)$ of x_j is defined as the fitness function value would decrease if x_j is flipped, which can be calculated by the following formula:

$$v(j) = g(x) - g(x_1, \cdots, 1 - x_j, \cdots, x_n). \tag{3}$$

The SA procedure calculates the move value $v(j)$ of each variable. We can choose the variable with the highest move value to flip directly. In order to diversify the search, a restricted candidate list (RCL) is constructed by the variables with move values larger than a threshold value. Specifically, let

$$RCL = \{j \in J : v(j) \geq \alpha \times v_{\max}\}, \tag{4}$$

where $v_{\max} = \max\{v(j), j \in J\}$, and $\alpha \in [0, 1]$ is a parameter. $\alpha = 0$ means that SA procedure selects a random variable to flip; $\alpha = 1$ means that SA procedure selects the variable with the highest move value to flip. Then, a variable x_t is randomly selected from RCL, and is flipped. Let x' be the obtained solution, i.e., $x' = (x_1, \cdots, 1 - x_t, \cdots, x_n)$. If x' is not worse than x, then it replaces x as the new current solution (line 9). Otherwise, x' is accepted with a small probability p.

The Boltzmann function is widely used to calculate the probability p. The research by Tiwari et al. [17] has shown that using the Cauchy function to calculate the probability p

has more opportunities to escape local optima. Our SA procedure uses Cauchy function to calculate the probability p. More specifically, let $\triangle E = g(y) - g(x)$, and the probability p is given by $p = \frac{T}{T^2+(\triangle E)^2}$.

After the above flip operator performs M iterations, line 20 decreases the current temperature T according to the rule $T = \gamma T$, $\gamma \in (0, 1)$. The search process is repeated until x^{best} has not improved for G_{no} consecutive temperature decreases.

Algorithm 1 SA procedure

Input: an initial solution x, G_{no}.
Output: an improved solution x^{best}.
1: Initial $T = T_0$, and $x^{best} = x$, $G = 0$.
2: **while** $G \leq G_{no}$ **do**
3: **for** $Iteration = 1$ to M **do**
4: Calculate the move value $v(j), j \in J$, according to (3).
5: Construct RCL according to (4).
6: Select a variable x_t from RCL at random.
7: Let $x' = (x_1, \cdots, 1 - x_t, \cdots, x_n)$.
8: **if** $g(x') < g(x)$ **then**
9: $x = x'$.
10: **if** $g(x') < g(x^{best})$ **then**
11: $x^{best} = x'$, $G = 0$.
12: **end if**
13: **else**
14: Generate a number $p \in (0, 1)$ at random.
15: **if** $p < \frac{T}{T^2+(g(x')-g(x))^2}$ **then**
16: Let $x = x'$.
17: **end if**
18: **end if**
19: **end for**
20: Let $T = \gamma T$, $G = G + 1$.
21: **end while**
22: **if** $g(x^{best}) < g(x^*)$ **then**
23: $x^* = x^{best}$.
24: **end if**

2.2. **Adaptive memory mechanism.** SA has been applied to solving many optimization problems. It usually starts from a randomly generated solution in each iteration. To enhance the performance of SA procedure, we use a memory mechanism in the construction of new starting solutions.

Suppose $E = \{x^1, \cdots, x^s\}$ is a set of elite solutions that has good solution quality. At the beginning, we use the greedy heuristic [18] to generate s solutions to form E. Our constructive procedure randomly selects two elite solutions from E, say x and y. A new solution is generated as follows.

First, a partial solution z' is generated. If $x_i = y_i$, the constructive procedure sets $z'_i = x_i$; otherwise, let $z'_i = 0$. Second, a greedy method is used to complete the partial solution z'. A greedy function is defined to evaluate the benefit of adding an unselected column into the current solution. We define the greedy function as follows: $\psi(j) = \frac{r_j}{c_j}$, where r_j is the number of uncovered rows which will be covered by column j. The constructive procedure iteratively adds an unselected column with the largest value $\psi(j)$ to the current solution z'. The add operator is repeated until z' is a feasible solution, i.e., all rows are covered by z'. Finally, a simple perturbation is used to diversify the search. μ

selected columns and μ unselected columns are determined at random. The constructive procedures swap these columns to form a new solution.

2.3. **The algorithm.** The proposed algorithm firstly uses the greedy heuristic [18] to generate an elite solutions set E. At each generation, a new starting solution is constructed by the constructive procedure, and is improved by the SA procedure (Algorithm 1). Then the obtained solution z is used to update the elite solutions set E. Let $x^{worst} = \arg\max\{g(x^i), x^i \in E\}$. If $g(z) < g(x^{worst})$, z is added to E, and x^{worst} is deleted from E. When the maximum number of generations $maxgeneration$ is reached, then we stop the proposed algorithm. The detailed algorithm is given in Algorithm 2.

Algorithm 2 SAMSH

Input: a matrix $A = (a_{ij})$.
Output: the best solution x^* found so far.

```
Initial generation = 0.
Generate elite solution set E by the greedy heuristic.
Let x* = arg min{g(x^i), x^i ∈ E}.
while generation ≤ maxgeneration do
    A new solution is generated by the constructive procedure, and is further improved
    by the SA procedure (Algorithm 1). The obtained solution is denoted by z.
    if g(z) < g(x*) then
        Let x* = z.
    end if
    Let x^worst = arg max{g(x^i), x^i ∈ E}.
    if g(z) < g(x^worst) then
        Let E = E ∪ {z} − {x^worst}.
    end if
    generation = generation + 1.
end while
```

3. **Computational Results.** The proposed algorithm SAMSH was coded in the C programming language and the tests were carried out on an AMD processor with 3.4 GHz clockpulse and 2.0 GB RAM under Windows XP. A collection of 45 benchmarks from OR Library are used to test the proposed algorithm. It consists of 7 groups: SCP4, SCP5, SCP6, SCPa, SCPb, SCPc, and SCPd.

Table 1. Settings of parameters

parameters	description	values
s	number of elite solutions in E	10
λ	penalty parameter in Equation (1)	defined in (5)
α	parameter in Equation (4)	0.7
T_0	initial temperature	1.0
γ	temperature updating parameter	0.9
G_{no}	number of non-improving temperature decreases	10
M	number of iterations in SA procedure	10
$maxgeneration$	maximum generation	5000
μ	swap magnitude	10

Table 2. Computational results

instance	optimum	GA	TGA	CA	BFA		SAMSH	
					f_{best}	f_{avg}	f_{best}	f_{avg}
SCP4.1	429	432	429	448	429	433.5	429	431.4
SCP4.2	512	521	512	603	517	541.9	512	520.1
SCP4.3	516	526	520	540	519	532.36	516	523.6
SCP4.4	494	500	504	512	495	519.66	494	499.6
SCP4.5	512	518	512	520	514	524.16	512	526.8
SCP4.6	560	569	560	605	563	583.8	560	563.5
SCP4.7	430	443	432	447	430	435.56	430	432.3
SCP4.8	492	502	497	548	497	506.6	492	497.2
SCP4.9	641	662	641	671	655	675.36	641	654.6
SCP4.10	514	543	517	533	519	533.76	514	518.3
SCP5.1	253	274	255	309	257	268.53	253	258.0
SCP5.2	302	313	308	330	309	315	302	307.3
SCP5.3	226	229	228	232	229	239.66	226	226.0
SCP5.4	242	247	243	250	242	247.73	243	247.5
SCP5.5	211	212	212	218	211	218.83	211	214.9
SCP5.6	213	219	213	227	213	233.96	213	214.2
SCP5.7	293	308	293	310	298	308.66	293	299.6
SCP5.8	288	314	288	311	291	303.13	292	300.7
SCP5.9	279	283	280	292	284	298.6	279	297.0
SCP5.10	265	275	269	278	268	274.46	265	268.6
SCP6.1	138	144	144	155	138	148.46	143	143.0
SCP6.2	146	154	146	171	147	154.36	146	147.0
SCP6.3	145	148	148	176	147	151.53	145	145.0
SCP6.4	131	133	131	141	131	136.43	131	132.1
SCP6.5	161	177	163	186	164	175.53	161	162.0
SCPa.1	253	260	253	303	255	259.63	255	261.5
SCPa.2	252	270	265	272	259	268.6	252	265.3
SCPa.3	232	240	234	245	238	246.36	242	250.9
SCPa.4	234	255	235	251	235	246.2	235	251.5
SCPa.5	236	242	237	248	236	240.1	236	242.3
SCPb.1	69	70	69	87	71	78.93	69	69.0
SCPb.2	76	84	80	78	78	85.23	76	76.0
SCPb.3	80	82	80	85	80	84.43	80	80.0
SCPb.4	79	84	83	83	80	83.36	79	79.0
SCPb.5	72	73	72	75	72	75.7	72	72.0
SCPc.1	227	234	232	254	230	234.03	236	246.7
SCPc.2	219	227	223	225	223	231.23	227	239.2
SCPc.3	243	263	248	259	253	265.2	250	256.5
SCPc.4	219	224	224	240	225	238.36	224	229.2
SCPc.5	215	217	219	219	217	220.93	217	220.4
SCPd.1	60	63	62	68	60	31.33	60	60.0
SCPd.2	66	69	68	71	68	71.33	66	66.0
SCPd.3	72	77	73	80	75	78.1	72	72.1
SCPd.4	62	65	63	67	62	65.33	62	62.3
SCPd.5	61	69	64	66	63	65.26	61	61.3

We use the exact penalty function to evaluate the quality of a solution. The penalty parameter λ is problem independent. In our experiments, we set

$$\lambda = \begin{cases} 30, & SCP4, \\ 12, & SCP5,\ SCPa,\ SCPc, \\ 8, & SCP6, \\ 3, & SCPb,\ SCPd. \end{cases} \tag{5}$$

The parameter settings of SAMSH used in our experiments are listed in Table 1. These parameter values were determined by a preliminary experiment.

We ran our proposed algorithm 10 times. Table 2 lists the best cost (f_{best}) and the average cost (f_{avg}) of our algorithm on the 45 benchmark instances. The column 'optimum' lists the optimal solution for each benchmark instance. In order to make comparison with genetic algorithm (GA) [3], two stage genetic algorithm (TGA) [3], cultural algorithm (CA) [4], and binary firefly algorithm (BFA) [5], we also list the computational results of these algorithms in Table 2. Note that the GA and TGA stop when 50 generations are reached. The data of GA and TGA is from [3]. The results of CA and BFA are taken from [4] and [5], respectively.

From Table 2, one can see that the proposed algorithm is able to find the current best known solutions on all tested instances. The average success rate of these instances is 68%, and the average CPU time for reaching the best results is about 231.525 seconds. These results provide evidence of the efficacy of our proposed algorithm.

4. **Conclusions.** In this paper, we presented a simulated annealing based multistart heuristic (SAMSH) for the set covering problem. The proposed algorithm uses an adaptive memory mechanism to generate good quality solutions. The newly generated solutions is enhanced by using the SA procedure. It achieves a good compromise between intensification and diversification in the search process. Computational experiments on 45 benchmark instances have demonstrated that our proposed algorithm is efficient. In future work, we look forward to extending the proposed algorithm to other related NP-hard problems.

Acknowledgment. This research was supported by the Science and Technology Project of the Education Bureau of Fujian, China, under Grant JB13063.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[2] J. E. Beasley and P. C. Chu, A genetic algorithm for the set covering problem, *European Journal of Operational Research*, vol.94, no.2, pp.392-404, 1996.

[3] Z. Wu, T. Chen, H. Wang et al., Two stage genetic algorithm for set covering problem, *Journal of Chinese Computer System*, vol.32, no.4, pp.732-737, 2011 (in Chinese).

[4] B. Crawford, R. Soto and E. Monfroy, Cultural algorithms for the set covering problem, *Advances in Swarm Intelligence*, 2013.

[5] B. Crawford, R. Soto, M. Riquelme-Leiva et al., Modified binary firefly algorithms with different transfer functions for solving set covering problems, in *Software Engineering in Intelligent Systems*, Springer International Publishing, 2015.

[6] S. Ceria, P. Nobili and A. Sassano, A Lagrangian-based heuristics for large-scale set covering problems, *Mathematical Programming*, vol.81, no.2, pp.215-228, 1998.

[7] B. M. Smith, IMPACS – A bus crew scheduling system using integer programming, *Mathematical Programming*, vol.42, no.1, pp.181-187, 1988.

[8] M. L. Fisher and P. Kedia, Optimal solutions of set covering/partitioning problems using dual heuristics, *Management Science*, vol.36, no.6, pp.674-688, 1990.

[9] M. Ashouri, Z. Zali, S. R. Mousavi and M. R. Hashemi, New optimal solution to disjoint set K-coverage for lifetime extension in wireless sensor networks, *Wireless Sensor Systems, IET*, vol.2, no.1, pp.31-39, 2012.

[10] M. Yagiura, M. Kishida and T. Ibaraki, A 3-flip neighborhood local search for the set covering problem, *European Journal of Operational Research*, vol.172, no.2, pp.472-499, 2006.

[11] Z. Naji-Azimi, P. Toth and L. Galli, An electromagnetism metaheuristic for the unicost set covering problem, *European Journal of Operational Research*, vol.205, no.2, pp.290-300, 2010.

[12] C. Gao, X. Yao, T. Weise and J. Li, An efficient local search heuristic with row weighting for the unicost set covering problem, *European Journal of Operational Research*, vol.246, no.3, pp.750-761, 2015.

[13] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing, *Science*, vol.220, no.19, pp.671-680, 1983.

[14] V. F. Yu, S. W. Lin, W. Lee and C. J. Ting, A simulated annealing heuristic for the capacitated location routing problem, *Computers & Industrial Engineering*, vol.58, no.2, pp.288-299, 2010.

[15] N. Jawahar, A. Gunasekaran and N. Balaji, A simulated annealing algorithm to the multi-period fixed charge distribution problem associated with backorder and inventory, *International Journal of Production Research*, vol.50, no.9, pp.2533-2554, 2012.

[16] A. R. Hedar and R. Ismail, Simulated annealing with stochastic local search for minimum dominating set problem, *International Journal of Machine Learning and Cybernetics*, vol.3, no.2, pp.97-109, 2012.

[17] M. K. Tiwari, S. Kumar and R. Shankar, Solving part-type selection and operation allocation problems in an FMS: An approach using constraints-based fast simulated annealing algorithm, *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol.36, no.6, pp.1170-1184, 2006.

[18] V. Chvatal, A greedy heuristic for the set-covering problem, *Mathematics of Operations Research*, vol.4, no.3, pp.233-235, 1979.