Computers & Industrial Engineering 85 (2015) 328–334

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

An improved hybrid algorithm for the set covering problem

Sameh Al-Shihabi *, Mazen Arafeh, Mahmoud Barghash

Industrial Engineering Department, The University of Jordan, Amman 11942, Jordan

ARTICLE INFO

Article history:
Received 30 November 2014
Received in revised form 2 April 2015
Accepted 7 April 2015
Available online 16 April 2015

Keywords:
Linear programming
Lagrangian relaxation
Max–min ant system
Ant colony optimization
Set covering problem

ABSTRACT

The state-of-the-art ant colony optimization (ACO) algorithm to solve large scale set covering problems (SCP) starts by solving the Lagrangian dual (LD) problem of the SCP to obtain quasi-optimal dual values. These values are then exploited by the ACO algorithm in the form of heuristic estimates. This article starts by discussing the complexity of this approach where a number of new parameters are introduced to escape local optimums and normalize the heuristic values. To avoid these complexities, we propose a new hybrid algorithm that starts by solving the linear programming (LP) relaxation of the SCP. This solution is used to eliminate unnecessary columns, and to estimate the heuristic information. To generate solutions, we use a Max–Min Ant System (MMAS) algorithm that employs a novel mechanism to update the pheromone trail limits to maintain a predetermined exploration rate. Computational experiments on different sets of benchmark instances prove that our proposed algorithm can be considered the new state-of-the-art meta-heuristic to solve the SCP.

1. Introduction

The set covering problem (SCP) is a classical NP-hard problem that has been addressed by numerous researchers. It is about choosing a subset of columns that have the least sum of costs to cover a set of rows. This problem is represented through the following Integer Program (IP) (1):

$$SCP: minZ = \sum_{j \in N} c_j x_j, \tag{1}$$

subject to,

$$\sum_{j \in N} a_{ij} x_j \geqslant 1, \quad i \in M,$$

$$x_j \in \{0,1\}, \quad j \in N.$$

where c_j is the cost of column j, while $a_{ij} = 1$, if row i is covered by column j and 0, otherwise. The number of columns is $|N|$, and the number of rows is $|M|$. J_i represents the set of columns covering row i, and I_j shows the set of rows covered by column j. Many industrial engineering problems have been modeled as SCP; examples of such problems include the facility location problem (Vasko & Wilson, 1984), steel production (Vasko & Wolf, 1987), ship scheduling (Fisher & Rosenwein, 1989), vehicle routing problem (Foster & Rayan, 1976), and most recently multi-depot train driver scheduling (Yaghini, Karimi, & Rahbar, 2015).

Solving large SCP instances to optimality using exact methods is not practical due to the significant time needed. For this reason, researchers depended on heuristic and meta-heuristic techniques to solve this problem, and these techniques are often coupled with information obtained from solving the Lagrangian relaxation (LR) problem and its associated Lagrangian dual (LD) problem (Caparara, Fischetti, & Toth, 1999).

The meta-heuristics used to solve this problem include: simulated annealing (SA)(Brusco, Jacobs, & Thompson, 1999), tabu search (TS) (Caserta, 2007), artificial bee colony (ABC) (Sundar & Singh, 2012), genetic algorithm (GA) (Beasely & Chu, 1996; Aickelin, 2002), and ant colony optimization (ACO) (Ren, Feng, Ke, & Zhang, 2010; Crawford & Castro, 2006; Lessing, Dumitrescu, & Stützle, 2004). As shown in Sundar and Singh (2012), the best meta-heuristics to solve the SCP are the ACO algorithms of Lessing et al. (2004) and Ren et al. (2010), and the ABC algorithm of Sundar and Singh (2012).

Due to their inferior results, the TS and SA approaches are not discussed here; on the other hand and despite their slightly bad performance compared to ACO algorithms, we discuss the GA solution approaches as they have some resemblance to the ACO algorithms with respect to problem representation.

To solve the SCP using GA algorithm, researchers had different ideas about what the chromosomes represent. In Beasely and Chu (1996), binary chromosomes represent the columns such that a 1 means that a column is included and a 0 means that it is excluded. On the other hand, in Aickelin (2002), chromosomes

* Corresponding author. Tel.: +962 795101756.
E-mail address: s.shihabi@ju.edu.jo (S. Al-Shihabi).

http://dx.doi.org/10.1016/j.cie.2015.04.007
0360-8352/

Table 1
Characteristics of the different SCP sets. For each set, we show the number of instances making up the set followed by the parameters of these instances: the number of rows, number of columns, and density.

Set	Number of instance	Number of rows	Number of columns	Density (%)
4	10	200	1000	2
5	10	200	2000	2
6	5	200	1000	5
A	5	300	3000	2
B	5	300	3000	5
C	5	400	4000	2
D	5	400	4000	5
NRE	5	500	5000	10
NRF	5	500	5000	20
NRG	5	1000	10,000	2
NRH	5	1000	10,000	5

represent rows. A decoder is then used to translate the sequence of rows into a solution of selected columns covering all rows.

Three ACO algorithms were published in the literature to solve the SCP. The quality of results presented in Crawford and Castro (2006) is inferior to other ACO algorithms; therefore, we exclude it from the current summary. Similar to the GA algorithms, ACO algorithms used two different construction graphs. In Lessing et al. (2004), the nodes of the construction graphs represent columns and two arcs are connected to each node such that these two arcs represent the inclusion and exclusion of the preceding column, node. In Ren et al. (2010), however, the nodes of the construction graph represents the sequence of rows. An ant would jump from an uncovered row to the other, and at each node it needs to select one of the arcs that represent the columns that cover the row.

In both (Lessing et al., 2004; Ren et al., 2010), the LR and its associated LD problems are solved before initiating the ACO algorithms. The LR problem of the SCP is represented through the following program (2), which shows how the constraints are added to the objective function after multiplying them by their associated dual values u_i.

$$LR: \min_{x\in\{0,1\}^n} L(u) = \sum_{j\in N} c_j(u)x_j + \sum_{i\in M} u_i. \quad (2)$$

The value of $x_j = 1$ if $c_j(u) = c_j - \sum_{k\in I_k} u_k \leqslant 0$, and $x_j = 0$, otherwise. The (LD) is about finding the best set of u that will maximize $L(u)$.

$$LD: \max_{u\in\mathbf{R}^n} L(u). \quad (3)$$

The above problem is solved using the steepest descent algorithm as described in Caparara et al. (1999). The set of dual variables $u \in \mathbf{R}^n$ that maximizes the function shown in program (3) are used to develop heuristic information by the ACO algorithms.

Different combinations of ACO frameworks, such as Max–Min Ants System (Stützle & Hoos, 2000) (MMAS) and Ant Colony System (ACS) (Lessing et al., 2004), in combination with different heuristic information were tested in Lessing et al. (2004). Among the different heuristics compared, the best ones found were based on the column's reduced cost that in turn depends on the dual information found from solving the LD problem. The successful heuristics are dynamic; meaning that when an ant constructs a solution, the values of the heuristics changes are based on the rows that have already been covered. The ACO framework used had a minor effect on the quality of results obtained, compared to the heuristics used. A number of the studied combinations succeeded in finding the best known solutions for all benchmark instances tested in the literature; however, the 3-flip (Yagiura, Kishida, & Ibaraki, 2006) local search algorithm, which is capable of finding the optimal solutions alone as shown in Yagiura et al. (2006) and Sundar and Singh (2012), was used to improve the ants solutions. Consequently, the excellent results found in Lessing et al. (2004) cannot be attributed to the ACO algorithms tested as discussed in Sundar and Singh (2012). Therefore, the ACO algorithm of Ren et al. (2010) can be considered as the state-of-the-art ACO algorithm to solve the SCP.

In Ren et al. (2010), the algorithm implements three optimization techniques: exact method, represented by solving the LD problem; a meta-heuristics, represented by the ACO algorithm; and a local search to improve solutions. The authors of Ren et al. (2010) use ACO–LS to denote their algorithm while in this paper, we denote this algorithm by LD–ACO–LS. Similar to Lessing et al. (2004), they use the dual values to assess their heuristic information. To guarantee results of high quality, the authors needed to introduce a set of new parameters to escape local optimums and normalize the heuristic information. The introduction and tuning of these parameters complicates the LD–ACO–LS algorithm which we discuss in this paper.

Like any ABC algorithm, the algorithm of Sundar and Singh (2012) divides the colony of bees into three types: employed, onlookers, and scouts. The employed bees find high quality solutions to attract onlookers to search the vicinity of these solutions. After searching the vicinity, if the employed bee solution is not improved, the employed bee becomes a scout that finds a new randomly generated solution. To generate high quality solutions in Sundar and Singh (2012), onlooker bees mutate the solutions of the chosen employed bee solution with other employed bees that generated good solutions. It also implements an improved version of the local search implemented in Ren et al. (2010). Moreover, Sundar and Singh (2012) did not allow employed bees to have the same solution. Excellent results were obtained using this algorithm; however, it had high computational time compared to the ACO algorithms as shown in Sundar and Singh (2012).

This paper is organized as follows. In Section 2 we review the LD–ACO–LS algorithm (Ren et al., 2010) and discuss the robustness of this algorithm. In Section 3 we present a new MMAS to solve the SCP where we introduce a new mechanism to control the pheromone trails limits. Moreover, we exploit the solution of the LP relaxation to reduce the size of the problem. In Section 4, we compare the best meta-heuristics to solve the SCP in terms of solution quality and time. Conclusions and future work are then discussed.

2. Ant colony optimization algorithms for the set covering problem

A generic framework for the ACO algorithms can be represented by Algorithm 1 where m represent the size of the ant colony and k is the number of solutions improved by the local search.

Algorithm 1. A generic pseudo-code for the ant colony optimization algorithms

```
Develop a representative construction graph of the problem
while termination criteria not met do
    Generate m solutions
    Improve k ⩽ m solutions using local search
    Update the pheromone trails
end
Report the best solution found
```

As discussed earlier, in the LD–ACO–LS algorithm, the nodes of the construction graph represent the rows and an ant moves from an uncovered row to the other and at each row; it selects a column that covers this row. The local search algorithm is used to improve every generated solution $m = k$. To improve solution S, this local search can be described by Algorithm 2.

Table 2
The performance of the LP–MMAS–LS, ACO–LS, and ABC algorithms with respect to the benchmark instances. For each algorithm and instance, the table shows the best and average solutions, in addition to the average computational time in seconds.

Instance	Best Known	LP–MMAS–LS			ACO–LS			ABC		
		Best	Avg.	Time	Best	Avg.	Time	Best	Avg.	Time
4.1	**429**	429	429.0	1.0	429	429.0	1.74	429	429.0	5.03
4.2	**512**	512	512.0	1.0	512	512.0	2.26	512	512.0	4.82
4.3	**516**	516	516.0	1.0	516	516.0	2.31	516	516.0	4.87
4.4	**494**	494	494.0	1.0	494	494.0	2.12	494	494.0	5.40
4.5	**512**	512	512.0	1.0	512	512.0	2.17	512	512.0	5.57
4.6	**560**	560	560.0	1.0	560	560.0	2.41	560	560.0	5.07
4.7	**430**	430	430.0	1.0	430	430.0	1.87	430	430.0	4.97
4.8	**492**	492	492.0	1.0	492	492.0	2.17	492	492.0	5.40
4.9	**641**	641	641.0	1.0	641	641.0	2.38	641	641.0	4.30
4.10	**514**	514	514.0	1.0	514	514.0	1.99	514	514.0	5.55
5.1	**253**	253	253.0	1.0	253	253.0	2.29	253	253.0	6.99
5.2	**302**	302	302.0	1.6	302	302.0	2.42	302	302.0	6.06
5.3	**226**	226	226.0	1.0	226	226.0	2.17	226	226.0	7.41
5.4	**242**	242	242.0	1.0	242	242.0	2.28	242	242.0	6.16
5.5	**211**	211	211.0	1.0	211	211.0	1.70	211	211.0	6.56
5.6	**213**	213	213.0	1.0	213	213.0	1.97	213	213.0	6.67
5.7	**293**	293	293.0	1.0	293	293.0	2.22	293	293.0	7.29
5.8	**288**	288	288.0	1.0	288	288.0	2.43	288	288.0	6.12
5.9	**279**	279	279.0	1.0	279	279.0	2.30	279	279.0	6.12
5.10	**265**	265	265.0	1.0	265	265.0	2.14	265	265.0	6.13
6.1	**138**	138	138.0	<1.0	138	138.0	2.75	138	138.0	11.60
6.2	**146**	146	146.0	<1.0	146	146.0	3.06	146	146.0	10.69
6.3	**145**	145	145.0	<1.0	145	145.0	3.00	145	145.0	10.26
6.4	**131**	131	131.0	<1.0	131	131.0	2.70	131	1310	9.10
6.5	**161**	161	161.0	<1.0	161	161.0	3.02	161	161.0	13.11
A.1	**253**	253	253.4	2.8	253	253.0	4.04	253	253.0	11.85
A.2	**252**	252	252.0	2.0	252	252.0	4.23	252	252.0	11.14
A.3	**232**	232	232.8	2.5	232	232.8	4.14	232	232.0	10.73
A.4	**234**	234	234.0	1.8	234	234.0	4.13	234	234.0	11.61
A.5	**236**	236	236.8	1.6	236	236.0	4.13	236	236.0	10.72
B.1	**69**	69	69.0	1.0	69	69.0	6.11	69	69.0	39.41
B.2	**76**	76	76.0	1.0	76	76.0	6.42	76	76.0	37.63
B.3	**80**	80	80.0	1.0	80	80.0	6.72	80	80.0	34.80
B.4	**79**	79	79.0	1.0	79	79.0	6.63	79	79.0	41.69
B.5	**72**	72	72.0	1.0	72	72.0	6.20	72	72.0	35.94
C.1	**227**	227	227.0	3.8	227	227.0	6.10	227	227.0	17.18
C.2	**219**	219	219.2	3.1	219	219.0	6.35	219	219.0	17.27
C.3	**243**	243	243.1	3.8	243	2430	6.36	243	243.0	18.77
C.4	**219**	219	219.0	2.5	219	219.0	6.11	219	219.0	17.54
C.5	**215**	215	215.0	3.3	215	215.0	6.23	215	215.0	17.63
D.1	**60**	60	60.0	1.0	60	60.0	9.85	60	60.0	78.57
D.2	**66**	66	66.0	1.0	66	66.0	10.32	66	66.0	74.27
D.3	**72**	72	72.0	1.0	72	72.0	10.47	72	72.0	74.56
D.4	**62**	62	62.0	1.6	62	60.2	10.35	62	62.0	66.14
D.5	**61**	61	61.0	1.0	60	60.0	9.72	61	61.0	67.42
NRE.1	**29**	29	29.0	1.0	29	29.0	21.12	29	29	89.05
NRE.2	**30**	30	30.0	1.8	30	30.0	23.49	30	30.0	98.82
NRE.3	**27**	27	27.0	1.2	27	27.0	21.50	27	27.0	104.09
NRE.4	**28**	28	28.0	1.5	28	28.0	23.27	28	28.8	93.18
NRE.5	**28**	28	28.0	1.0	28	28.0	23.72	28	28.0	97.71
NRF.1	**14**	14	14.0	1.0	14	14.0	30.13	14	14.0	330.41
NRF.2	**15**	15	15.0	1.0	15	15.0	28.23	15	15.0	282.54
NRF.3	**14**	14	14.0	1.4	14	14.0	30.70	14	14.0	308.46
NRF.4	**14**	14	14.0	1.2	14	14.0	29.94	14	14.0	325.11
NRF.5	**13**	13	13.3	1.7	13	13.5	27.13	13	13.7	324.23
NRG.1	**176**	176	176.0	11.9	176	176.0	31.16	176	176.0	97.54
NRG.2	**154**	154	155.0	11.9	154	155.1	29.03	155	155.0	96.89
NRG.3	**166**	166	167.5	11.7	166	167.3	30.24	166	166.0	94.41
NRG.4	**168**	168	168.9	10.1	168	168.9	29.73	168	168.0	92.51
NRG.5	**168**	168	168.7	9.7	168	168.1	30.85	168	168.0	92.80
NRH.1	**63**	63	63.8	7.4	64	64.0	71.47	63	63.9	532.36
NRH.2	**63**	63	64.0	7.7	63	63.9	71.04	63	63.9	533.91
NRH.3	**59**	59	60.2	7.2	59	59.4	69.66	59	59.0	557.13
NRH.4	**58**	58	58.8	6.2	58	58.7	70.38	58	58.0	565.78
NRH.5	**55**	55	55.0	5.3	55	55.0	68.23	55	55.0	539.07

The bold numbers show the best-known solutions while the underline values show the instances that each algorithm missed to obtain the best-known solution.

Algorithm 2. A pseudo-code for the local search algorithm used in Sundar and Singh (2012)

```
for each column j ∈ S, find the set of rows o_j that are solely
    covered by j
for each column j ∈ S in a non-increasing order of cost do
    if |o_j = 0| then
        remove column j
    end
    if o_j = {i_1}, c_k < c_j, k ∈ {N/S} and k ∈ J_{i_1} then
        replace column j by column k
    end
      if o_j = {i_1, i_2}, c_k < c_j, k ∈ {N/S}, k ∈ J_{i_1} and k ∈ J_{i_2} then
        replace column j by column k
      end
      if o_j = {i_1, i_2}, c_{k1} + c_{k2} < c_j,
      k1 ∈ {N/S}, k2 ∈ {N/S}, k1 ∈ J_{i_1}, k1 ∈ J_{i_1} then
          then replace column j by columns k1 and k2
      end
end
```

The local search tries to remove redundant columns first, and then it tries to replace columns covering one or two rows only by less expensive ones. In their ABC algorithm, Sundar and Singh (2012) extends this local search such that columns covering three rows, $|o_j| = 3$, are also checked. It was argued by Sundar and Singh (2012) that this step improves the local search.

After this quick review of the LD–ACO–LS algorithm, we review in details how the dual information obtained from solving the LD problem is used to develop heuristics to be used by ants when generating solutions. We, however, shed some lights into a number of problems associated with this approach.

2.1. Dynamic reduced cost-based heuristics

In Ren et al. (2010), the set of duals $u = \{u_1, u_2, \ldots u_m\}^{R^m}$ are obtained from solving the LD problem using the subgradient method, before starting the ACO algorithm.

As discussed earlier, a row-based construction graph is used such that when an ant generates a solution, it randomly selects an uncovered row i, then it calculates the reduced costs of column $j \in J_i$ according to Eq. (4):

$$\pi_j(u, i, R) = c_j - \sum_{i \in I_j \cap R} u_i \tag{4}$$

The set R represents the set of uncovered rows. Before generating a solution, $\pi_j(u, i, R) = \pi_j(u)$, then the values of $\pi_j(u, i, R)$ change based on the row selected and the rows that have already been covered by the solution. To avoid the problem of having positive and negative $\pi_j(u, i, R)$, a new term σ is defined as shown in Eq. (5). This terms is added to $\pi_j(u, i, R)$ when calculating the heuristic information as shown in Eq. (7).

$$\sigma = 2.|min\{c_j(u)\}| \tag{5}$$

In addition to the reduced cost $\pi_j(u, i, R)$ of column j, the heuristic information η_j of the column depends on the number of rows that are still uncovered and can be covered by column j, which is found based on Eq. (6).

$$\phi_j = |R \cap I_j| \tag{6}$$

The heuristic information η_j is calculated according to Eq. (7), and the probability of selecting column $j \in J_i$ is given by Eq. (8)

$$\eta_j = \frac{\phi_j}{\pi_j(u) + \sigma}. \tag{7}$$

$$P_j = \frac{\tau_j^{\alpha} \cdot \eta_j^{\beta}}{\sum_{q \in I_q} \tau_q^{\alpha} \cdot \eta_q^{\beta}}, \tag{8}$$

where τ_j represents the amount of the pheromone trails left by the ants along arc j- column j for the SCP. After selecting a column to cover the randomly chosen row i, an ant randomly chooses another uncovered row. It selects a new column according to Eq. (8) and keeps randomly moving to new uncovered rows, until all the rows are covered.

2.2. Problems in using the dynamic reduced cost-based heuristics

To guarantee high quality solutions, Ren et al. (2010) complicate their algorithm by adding still new parameters and procedures to their algorithm. One such procedure is that if the solution of the problem does not improve for 50 consecutive runs, each component of the best u found, before starting the ACO algorithm, is perturbed by a random number δ, which is uniformly distributed between -0.2 and 0.2. The new components of u become

$$u_i = (1 + \delta)u_i, \tag{9}$$

and the subgradient optimization technique is applied again to reach a new set of quasi-optimal u.

In Ren et al. (2010), it was not shown how the different parameters used to calculate π_j are chosen; these parameters are the 2 factor in Eq. (5), the 50 runs after which u needs to be perturbed, and the amount of perturbation that is controlled by $\delta = uniform(-0.2, 0.2)$. Thus, a valid question that can be asked here: are these parameters valid for any SCP or are they specific for the set of benchmark problems used in Ren et al. (2010)? We, therefore, try to find a simpler ACO algorithm that has less parameters and can still produce high quality results.

3. A new hybrid max–min ant system for the set covering problem

We present here a hybrid of LP-relaxation and MMAS to solve large scale SCP. Hereafter, we will refer to this new algorithm as LP–MMAS–LS. We use the same construction graph and local search used by Ren et al. (2010); however, we exploit the LP-relaxation solution to reduce the size of the problem. This idea is not new for ACO algorithms. For example, to solve large scale traveling salesman problems (TSP), candidate lists are used to minimize the selection choices for ants (Stützle & Hoos, 2000). Moreover, using candidate lists to minimize the search space is used in a number of meta-heuristics to solve the SCP. Both Sundar and Singh (2012) and Beasely and Chu (1996) used such candidate lists in their ABC and GA algorithms, respectively. Lastly, in the pioneering work of Caparara et al. (1999), a heuristic is suggested to find feasible solutions and this heuristic is applied over a set of core columns after reducing the problem size. In this work, we adopt the idea of reducing the problem size instead of using candidate lists.

In the LP–MMAS–LS algorithm, we use a dynamic heuristic similar to Ren et al. (2010) and Lessing et al. (2004); however, we suggest a new methodology to normalize the heuristics that does not depend on any external factor like the σ factor in Eq. (5). We also propose a new pheromone updating mechanism such that the lower limits of the pheromone trails are dynamically calculated when choosing columns.

3.1. Reducing the problem size

We exploit the information obtained from solving the LP-relaxation of the problem such that we keep the columns having one of the following two characteristics:

1. Have $\pi_j \leqslant 1.0$.
2. Be one of the five least expensive columns covering a row.

The percent of columns kept out of the original instances ranged from 55.7% to 2.2% for the set of benchmark instances solved in this work. The average percent of columns kept is 13.6%. We use the notation N^{core} to represent the set of columns found after the reduction step; moreover, J_i^{core} represents the set of core columns that covers row i.

3.2. Pheromones updating

Ren et al. (2010) pointed out that many local optimum solutions might exist for this problem but in their algorithm, only one of these solutions is selected to update the pheromone trails. We believe that by using one solution, information about good columns that are not part of the updating solution will be lost. In our work, we randomly choose one of these solutions to update the pheromone trails. Columns that are part of many incumbent solutions will have higher probabilities to be selected to update the pheromone trails. Moreover, a column that is included in one incumbent solution needs to have a higher pheromone trail value than columns that are not selected by any solution. The set of incumbent solutions will be denoted by IS. Denoting the updating solution by $S^{updating} \in IS$, the pheromone trails are updated according to Eq. (10)

$$\tau_j(t) = \rho\tau_j(t-1) + (1-\rho).1\{j \in S^{updating}\} \tag{10}$$

In a typical MMAS, the pheromone values are checked against an upper and a lower τ limits, τ_{max} and τ_{min}, respectively. We only check the τ values against their upper limits, such that if $\tau_j(t) \geqslant \tau_{max}$ then $\tau_j(t) = \tau_{max}$. For τ_{min}, we make it a function of the row selected, $\tau_{min}(i)$; thus, it will be calculated dynamically when generating solutions. The reason behind this is that the different rows have different number of columns to cover them.

To calculate $\tau_{min}(i)$, we define a new parameter Q to represent the probability of regenerating the $S^{updating}$ solution. As a selection problem, Q depends on selecting the $\left|S^{updating}\right|$ columns included in $S^{updating}$. Therefore, the probability of selecting one of these columns is

$$P_{correct} = Q^{|S^u|}. \tag{11}$$

The value of $\tau_{min}(i)$ will depend on $P_{correct}$ and $|J_i^{core}|$ as shown in Eq. (12)

$$\tau_{min}(i) = \frac{\tau_{max}(1-P_{correct})}{|J_i^{core}| - 1}. \tag{12}$$

We check the values of τ against $\tau_{min}(i)$ when constructing solutions. If $\tau_j(t) \leqslant \tau_{min}(i)$ then $\tau_j(t) = \tau_{min}(i)$.

3.3. Column selection probabilities

Similar to Ren et al. (2010), the heuristic used is based on the dynamic reduced costs and the number of rows that can be covered by selecting this column. We use Eq. (4) above to calculate the reduced costs; however, we select $j \in J_i^{core}$ instead of $j \in J_i$. To normalize the values of the heuristics, we use a max–min normalization procedure rather than adding the σ term as shown in Eq. (6). After finding the minimum and maximum values of the reduced costs, $\pi_j^{min}(i)$ and $\pi_j^{max}(i)$, we normalize the values of $\pi_j(u,i,R)$ using Eq. (13)

$$\pi_j(i) = 0.1 + \frac{\pi_j(u,i,R) - \pi_j^{min}(i)}{\pi_j^{max}(i) - \pi_j^{min}(i)}. \tag{13}$$

The 0.1 is needed to have finite values of η_j. Having these values, we calculate the initial values of the heuristic information η_j

$$\eta_j = \frac{\phi_j}{\pi_j(u,i,R)}. \tag{14}$$

We, however, normalize the values of η_j again by finding their minimum and maximum values $\eta_j^{min}(i)$ and $\eta_j^{max}(i)$ respectively, and similar to π_j, we use a max–min normalization procedure as shown in Eq. (15)

$$\eta_j(i) = 0.1 + \frac{\eta_j(i) - \eta_j^{min}(i)}{\eta_j^{max}(i) - \eta_j^{min}(i)}. \tag{15}$$

Selecting a column to cover row i is found using Eq. (8).

3.4. Algorithm summary

Trying to put all the ideas discussed earlier together, we present here the pseudo-code that summarizes the proposed algorithm in addition to the solution generation procedure. Algorithm 3 represents the whole LP–MMAS–LS algorithm. It starts by relaxing the constraints and solving the resulting LP-relaxation problem. The reduced costs are then used to find N^{core}. The MMAS is executed as long as a termination criteria is not met.

At each iteration, we generate m solutions as shown in Algorithm 3. These solutions are then improved using the local search algorithm. After improving the solutions, if one of these solutions is found to be better than the incumbent one, then this solutions becomes the new incumbent; otherwise, it is added to the set of incumbent solutions if it is not already included. One of the incumbent solutions is then chosen randomly to update the pheromone trails.

We present the function used to generate solutions in Algorithm 4. As stated earlier, row i that is not covered yet is randomly selected. The $\tau_{min}(i)$ value of this row is then calculated and if for any $j \in J_i$, the value of τ_j is less than $\tau_{min}(i)$ then $\tau_j = \tau_{min}(i)$. To calculate the heuristic information, we conduct two normalizing procedures, one for the reduced costs and the other for the heuristic information itself. A column is then chosen based on the values of τ_j and η_j.

Algorithm 3. A pseudo code for the LP–MMAX–LS algorithm

```
Formulate the LP-relaxation of the SCP;
π ⟵ solve the LP-relaxation using CPLEX;
N^core ⟵ ReduceSize(π);
while termination criteria not met do
    For i ← 1 to m;
    (S^sol, g^sol) ⟵ generateSolution();
    (S^improved, g^improved) ⟵ localSearch(S^sol, g^sol);
    if g^improved < g^best;
    then
        g^best = g^improved ;
        GS = S^improved;
    end
    if g^improved = g^best and S^improved ∉ GS;
    then
        GS = GS ⋃ S^improved;
    end
    Update the pheromone trails;
end
Report the best solution found
```

Algorithm 4. A pseudo-code representing the solution generation function, generateSolution()

```
Function generateSolution()
Initialize: S^sol = φ; I^uncovered = I; π_max = −1000; π_min = 1000;
  η_max = −1000; η_min = 1000;
while I^uncovered ≠ φ do
    Randomly select row i ∈ I^uncovered;
    Calculate the value of τ_min(i) based on Eqs. (11) and (12);
    if τ_j < τ_min(i), ∀j ∈ J_i^core then
        τ_j = τ_min(i)
    end
    Calculate π_j(u, i, R) based on Eq. (4), ∀j ∈ J_i^core;
    Find π_max and π_min
    Normalize π_j(u, i, R) based on Eq. (13), ∀j ∈ J_i^core;
    Calculate η_j based on Eq. (14), ∀j ∈ J_i^core;
    Find η_max and η_min
    Normalize η_j based on Eq. (15), ∀j ∈ J_i^core;
    Calculate P_j based on Eq. (8), ∀j ∈ J_i^core;
    Randomly select column j ∈ I_j^core based on P_j S^sol = S^sol ⋃ j
end
return S^sol
```

4. Benchmarking

In this section, we test our proposed algorithm using a set of benchmark instances that can be downloaded from http://people.brunel.ac.uk/mastjjb/jeb/info.html. The instances can be divided into 11 sets as shown in Table 1. Instances belonging to the same set have the same number of rows and columns, and the same probability of having $a_{ij} \neq 0$: density. As seen from Table 1, some sets are easier to solve than others: Set 4 only has 200 rows to cover using 1000 columns, which is easier than set NRH, which has 1000 rows to cover using 10,000 columns. For all of these sets, the costs of the columns ranged between 1 and 100.

We compare our results to the LD–ACO–LS algorithm (Ren et al., 2010) in addition to the ABC algorithm of Sundar and Singh (2012); we do not include other meta-heuristics in our comparison since their performances are inferior to the LD–ACO–LS and ABC algorithms (Sundar & Singh, 2012). Readers interested in a full comparative study, excluding our proposed algorithm, can refer to Sundar and Singh (2012).

The parameters used in our algorithm are as follow: $Q = 0.9$, $m = 20$, $\alpha = 1.0$, $\beta = 0.5$, $\rho = 0.99$ and $\tau_{max} = 0.95$. As a termination criteria, our algorithm stops if the value of $S^{incumbent}$ does not change for 500 consecutive runs. We use CPLEX12.6 to solve the LP-relaxation of the problem.

Table 2 shows that our proposed algorithm is the first meta-heuristic that finds the optimum solutions for all the instances studied. The LD–ACO–LS missed instance NRH.1 while the ABC algorithm missed instance NRG.2. In terms of the averages obtained, the ABC algorithm is the best algorithm in terms of the average solutions. The averages of 55 instances out the 65 instances were equal to the optimum solutions compared to 53 instances for the ACO–LS algorithm and 51 instances for our proposed algorithm. The better averages obtained by the ABC algorithm can be justified by the higher computation time needed by this algorithm.

The processor used to conduct this experiment is 2.5 GHz Core(TM) i5-3210 M with 8 GB RAM. Other results are obtained by running the experiment using 2.4 GHz Xeon Processor with 2 GB RAM for the LD–ACO–LS algorithm and 3.0 GHz Core 2 Duo processor with 2 GB ram for the ABC algorithm. As stated in Sundar and Singh (2012), the LD–ACO–LS algorithm is better than the ABC algorithm with respect to computation time. Given the capabilities of our processor and the one used to obtain the LD–ACO–LS results (Ren et al., 2010), we believe that the differences in the computation times are in our favor regardless of any possible factor needed to adjust our times. This difference is very obvious in the big sets: NRG and NRF. Except for the NRG and NRF instances, all other instances were solved in less than five seconds, most of them in less than two seconds. The average computation time did not exceed one second for 37 instances out of the 65 instance. It took less that one second to find the optimal answer for the scp6 set. Therefore, the computation time of our algorithm is better than the LD–ACO–LS algorithm and much better than the ABC algorithm. The vivid superiority of our computational time is intuitive due to the reduction in the number of columns.

5. Conclusion

In this work, we propose a new hybrid algorithm of a MMAS, LP-relaxation and local search to solve large scale SCP. We exploit the LP-relaxation solution of the problem in two ways: reducing the size of the problem, and developing a heuristic estimate to be used by ants. Minimizing the size of the problem had a great impact on the solution quality and computation time. Exploiting the dual information to develop a dynamic heuristic estimates is neither new nor unique to ACO algorithms; however, a new normalization scheme is used that is more intuitive and avoids using new parameters. The MMAS used in this algorithm employs a new mechanism to control the pheromone trails levels such that a predefined level of exploration is maintained.

The comparison between our proposed algorithm and other ACO algorithm shows that new measures to assess the complexities of algorithms are needed by the OR society. Extra parameters were used in the previous state-of-the-art algorithm to solve the SCP compared to our algorithm. Obviously, the introduction of these parameters complicates the algorithm.

References

Aickelin, U. (2002). An indirect genetic algorithm for the set covering problems. *Journal of the Operational Research Society, 53*, 1118–1126.

Beasely, J., & Chu, P. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research, 94*, 392–404.

Brusco, M., Jacobs, L., & Thompson, G. (1999). A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set-covering problems. *Annals of Operations Research, 86*, 611–627.

Caparara, A., Fischetti, M., & Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research, 47*, 730–743.

Caserta, M. (2007). Tabu search-based metaheuristic algorithm for the large-scale set covering problem. In K. F. Doerner et al. (Eds.), *Metaheuristics: Progress in complex systems optimization* (pp. 43–63). New York: Springer.

Crawford, B., & Castro, C. (2006). Integrating lookahead and post processing procedures with ACO for solving set partitioning and covering problems. In L. Rutkowski et al. (Eds.), *Proceedings of the 8th international conference on artificial intelligence and soft computing ICAISC* (pp. 1082–1090). Berlin, Heidelberg: Springer-Verlag.

Fisher, M., & Rosenwein, M. (1989). An interactive optimization system for bulk-cargo ship scheduling. *Naval Research Logistics, 36*, 27–42.

Foster, B., & Rayan, D. (1976). An integer programming approach to the vehicle scheduling problem. *Operations Research Quarterly, 27*, 367–384.

Lessing, L., Dumitrescu, I., & Stützle, T. (2004). A comparison between ACO algorithms for the set covering problem. In M. Dorigo et al. (Eds.), *Proceedings of ANTS* (pp. 1–12). Berlin, Heidelberg: Springer-Verlag.

Ren, Z., Feng, Z., Ke, L., & Zhang, Z. (2010). New ideas for applying ant colony optimization to the set covering problem. *Computers and Industrial Engineering, 58*, 774–784.

Stützle, T., & Hoos, H. (2000). Max–min ant system. *Future Generation Computer Systems, 16*, 889–914.

Sundar, S., & Singh, A. (2012). A hybrid heuristic for the set covering problem. *Operational Research, an International Journal, 12*, 345–365.

Vasko, F., & Wilson, G. (1984). Using a facility location algorithm to solve large set covering problems. *Operations Research Letters, 3*(2), 85–90.

Vasko, F., & Wolf, F. (1987). Optimal selection of ingot sizes via set covering. *Operations Research, 35*(3), 346–353.

Yaghini, M., Karimi, M., & Rahbar, M. (2015). A set covering approach for multi-depot train driver scheduling. *Journal of Combinatorial Optimization, 29*, 636–654.

Yagiura, M., Kishida, M., & Ibaraki, T. (2006). A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research, 172*, 472–499.