Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 176 (2007) 1387–1403
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

An effective and simple heuristic for the set covering problem

Guanghui Lan [a,*], Gail W. DePuy [b], Gary E. Whitehouse [c]

[a] *Georgia Institute of Technology, School of Industrial and Systems Engineering, Atlanta, GA 30332, United States*
[b] *University of Louisville, Louisville, KY 40292, United States*
[c] *University of Central Florida, Orlando, FL, United States*

Received 16 August 2004; accepted 2 September 2005
Available online 6 December 2005

Abstract

This paper investigates the development of an effective heuristic to solve the set covering problem (SCP) by applying the meta-heuristic Meta-RaPS (Meta-heuristic for Randomized Priority Search). In Meta-RaPS, a feasible solution is generated by introducing random factors into a construction method. Then the feasible solutions can be improved by an improvement heuristic. In addition to applying the basic Meta-RaPS, the heuristic developed herein integrates the elements of randomizing the selection of priority rules, penalizing the worst columns when the searching space is highly condensed, and defining the core problem to speedup the algorithm. This heuristic has been tested on 80 SCP instances from the OR-Library. The sizes of the problems are up to 1000 rows × 10,000 columns for non-unicost SCP, and 28,160 rows × 11,264 columns for the unicost SCP. This heuristic is only one of two known SCP heuristics to find all optimal/best known solutions for those non-unicost instances. In addition, this heuristic is the best for unicost problems among the heuristics in terms of solution quality. Furthermore, evolving from a simple greedy heuristic, it is simple and easy to code. This heuristic enriches the options of practitioners in the optimization area.

Keywords: Combinatorial optimization; Set covering; Meta-RaPS

1. Introduction

The set covering problem (SCP) is a fundamental combinatorial problem in Operations Research. It is usually described as the problem of covering the rows of this m-row, n-column, zero-one matrix (a_{ij}) by a subset of the columns at minimal cost. It can be formally formulated as a binary integer program as follows:

* Corresponding author.
E-mail address: guanghui_lan@hotmail.com (G. Lan).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2005.09.028

$$\text{Let } x_j = \begin{cases} 1, & \text{if column } j \text{ belongs to the solution} \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J$$

$$\text{Minimize} \quad \sum_{j \in J} c_j x_j \tag{1}$$

$$\text{Subject to} \quad \sum_{j \in J} a_{ij} x_j \geqslant 1 \quad \text{for } i \in I, \tag{2}$$

$$\text{and } x_j = 0 \text{ or } 1 \quad \text{for } j \in J. \tag{3}$$

The following notations are often used for describing the set covering problem:

$J_i = \{j \in J : a_{ij} = 1\}$: the subset of columns covering row i.
$I_j = \{i \in I : a_{ij} = 1\}$: the subset of rows covered by column j.
$q = \sum_{i \in I} \sum_{j \in J} a_{ij}$: the number of non-zero entries of the matrix (a_{ij}).
$d = \frac{q}{m \times n}$: the density of the set covering problem.

If the costs c_j are equal for $j \in J$, the problem is referred to as the unicost SCP, otherwise, the problem is called the weighted or non-unicost SCP.

The SCP is important in practice, as it has been used to model a large range of problems arising from scheduling, manufacturing, service planning, information retrieval, etc. One important application problem is the delivery and routing problem. Another famous problem is the airline crew scheduling problem (Caprara et al., 1999). Two good survey papers for the application of SCP exist; Balas (1983) provided a survey for the applications in location, distribution and scheduling, and Ceria et al. (1998a,b) gave an annotated bibliography that refers to more recent studies on the SCP.

The SCP is an NP-hard problem in the strong sense (Garey and Johnson, 1979) and many algorithms have been developed for solving the SCP. The exact algorithms (Fisher and Kedia, 1990; Beasley and JØrnsten, 1992; Balas and Carrera, 1996) are mostly based on branch-and-bound and branch-and-cut. Caprara et al. (2000) compared different exact algorithms for the SCP. They show that the best exact algorithm for the SCP is CPLEX. Since exact methods require substantial computational effort to solve large-scale SCP instances, heuristic algorithms are often used to find a good or near-optimal solution in a reasonable time. Greedy algorithms may be the most natural heuristic approach for quickly solving large combinatorial problems. As for the SCP, the simplest such approach is the greedy algorithm of Chvatal (1979). Although simple, fast and easy to code, greedy algorithms could rarely generate solutions of good quality as a result of their myopic and deterministic nature. Researchers have tried to improve greedy algorithms by introducing some randomness. These randomized or probabilistic greedy algorithms (Vasko and Wilson, 1984; Feo and Resende, 1989; Haouari and Chaouachi, 2002) often generate better results than the pure greedy one. To improve the solution quality, modern heuristics, such as simulated annealing (SA), genetic algorithms (GA), and neural networks (NN), introduce randomness in a systematic manner. These heuristics are often classified as Meta-heuristics, since they are top-level general strategies that guide other heuristics to search for feasible solutions. An incomplete list of heuristics of this kind for the SCP includes genetic algorithms (Beasley and Chu, 1996; Aickelin, 2002), Simulated Annealing Algorithms (Jacobs and Brusco, 1995), Neural Network Algorithms (Ohlsson et al., 2001). Rather than applying a general Meta-heuristic, some other heuristics, such as those based on Lagrangian relaxation (Beasley, 1990; Ceria et al., 1998a,b; Caprara et al., 1999) are developed based on the problem-specific information of the SCP. In these heuristics, randomness has also been included to improve the solution.

There are two main drawbacks associated with existing SCP heuristics. First, most SCP heuristics are designed for the non-unicost problems. Beasley (1990, 1996) pointed out that their algorithms based on

the Lagrangian relaxation and genetic algorithm were not recommended for unicost problems, since the cost information plays an important role in these algorithms. Jacobs and Brusco (1995) assumed the same point of view for their heuristic based on Simulated Annealing. From the literature, very few heuristics are found to work effectively for both unicost and non-unicost problems. The second drawback of current SCP solution techniques is that most heuristics that could generate good solutions are difficult for implementation for those practitioners without a strong background in operations research. On the other hand, some simple heuristics, such as greedy heuristics usually could not find very good results.

In an attempt to resolve the previously mentioned problems associated with current SCP heuristics, the goal of this work is to design a robust, simple and fast heuristic that generates good results for both unicost and non-unicost set covering problems. Based on a new meta-heuristic, Meta-RaPS (Meta-heuristic for Randomized Priority Search), an effective heuristic that could achieve the above-mentioned goals was developed. Experimentation conducted using SCP instances from the OR-library (Beasley, 1990) shows the developed algorithm generates all the best-known solutions for non-unicost problems, and for unicost problems it not only finds all the best known solution, but also updates the best known solutions in two instances.

2. General Meta-RaPS approach

Meta-RaPS is a meta-heuristic developed by DePuy et al. (2002). It evolved from a computer heuristic designed by Arcus (1966) to solve the assembly line balancing problem—COMSOAL (Computer Method of Sequencing Operations for Assembly Lines). The theme of Meta-RaPS is the use of randomness as a mechanism to avoid local optima, which is similar to other above-mentioned meta-heuristics. Meta-RaPS is an iterative searching procedure. At each iteration, it constructs a feasible solution through the utilization of a construction heuristic in a randomized fashion, and then applies an improvement heuristic to improve the feasible solution if desirable. The best solution is then reported after a number of iterations.

The construction heuristic generates a feasible solution by adding basic elements step by step. The basic elements in a solution for a given combinatorial problem, such as, the cities in a tour for TSP, or the columns in a cover for SCP, are described by a finite set $\mathrm{BE} = \{1, 2, \ldots, n\}$, and the feasible basic elements ($\mathrm{FE} \subseteq \mathrm{BE}$) are those elements eligible for selection at each construction step. Each feasible basic element is characterized by a greedy score (P_i for $i \in \mathrm{FE}$) according to some priority rule. Dependent on the definition of the priority rule, the best feasible element might assume either the lowest score or the highest score. Instead of always choosing the best feasible element, as is done in greedy heuristics, Meta-RaPS introduces randomness via two parameters: %priority and %restriction. The %priority parameter determines the percentage of time that the best feasible basic element will be chosen. The remaining time, the element added to the solution will be randomly chosen from a candidate list that includes all elements considered 'acceptable'. An acceptable element is one whose greedy score is close to that of the best feasible element. The second parameter, %restriction, is used to determine whether a feasible basic element is acceptable and therefore should appear on the candidate list (CL). The CL is formed in the following manner.

If the best feasible element (k) is defined as the one with the lowest greedy score, i.e.

$$P_k = \min_{j \in \mathrm{FE}}\{P_j\}, \quad \text{then } \mathrm{CL} = \{j \,:\, j \in \mathrm{FE} \quad \text{and} \quad P_j \leqslant P_k \times (1 + \%\text{restriction}/100)\},$$

Otherwise, if the best feasible element (k) is the element with the highest greedy score, i.e.

$$P_k = \max_{j \in \mathrm{FE}}\{P_j\}, \quad \text{then } \mathrm{CL} = \{j : j \in \mathrm{FE} \quad \text{and} \quad P_j \geqslant P_k \times (1 - \%\text{restriction}/100)\}.$$

The smaller the %priority, the more randomness will be introduced. For a given %priority, the larger the %restriction, the more randomness will be introduced. Elements are added to the solution until a feasible solution is generated.

After a feasible solution is constructed, an improvement algorithm, usually a neighborhood search method, can be applied to improve the solution. The parameter %improvement is used to define how many feasible solutions will be improved. Suppose $Z^*_{\text{before improvement}}$ is the best objective function value found before applying the improvement procedure, a constructed solution will be improved if its objective function value Z satisfies the following requirements.

$Z \leqslant (1 + \%\text{improvement}/100) \times Z^*_{\text{before improvement}}$ for a minimization problem, or
$Z \geqslant (1 - \%\text{improvement}/100) \times Z^*_{\text{before improvement}}$ for a maximization problem.

The idea underlying this strategy is the expectation that good unimproved solutions lead to better neighboring solutions.

The differences between Meta-RaPS and other similar algorithms, such as Greedy Algorithms, COMSOAL and GRASP (Greedy Randomized Adaptive Search Procedure) have been investigated by DePuy et al. (2005). Meta-RaPS could be viewed as a general form of these three heuristics and it is more flexible and more efficient. Meta-RaPS has been applied successfully for solving the traveling salesman problem (DePuy et al., 2005), multi-dimensional knapsack problem (Moraga et al., 2005), and resource constrained project-scheduling problem (Whitehouse et al., 2002), etc.

3. Meta-RaPS SCP

The essential steps to successfully apply Meta-RaPS are to design effective construction and improvement heuristics. For the SCP, a simple and well-known construction heuristic will be used initially and a demonstration of how this simple heuristic evolves into an effective one with the collaboration of Meta-RaPS will be provided.

3.1. SCP construction heuristic

The first effort of this research is to modify Chvatal's (1979) greedy heuristic. This greedy heuristic evaluates each column j by the function $f(c_j, k_j) = c_j/k_j$, where c_j is the cost of column j, and k_j is the number of currently uncovered rows that could be covered by column j, i.e. $k_j = |\{i : i \in I_j \setminus \cup_{n \in X} I_n\}|$, and always adds to a solution set (X) the column with the minimum value of c_j/k_j. In Meta-RaPS, we still choose $f(c_j, k_j) = c_j/k_j$ as the priority rule, so the greedy score for each column j (P_j) is c_j/k_j. Note that here the lower the greedy score, the more eligible a column is. Only during the %priority of time, the column with the minimum value of P_j will be selected, while the remaining time, a column from the CL will be randomly selected. After a feasible solution is constructed, all redundant columns (column j is redundant if $X \setminus \{j\}$ is still a cover) will be removed from the solution such that the redundant column with largest cost will be removed at first. Fig. 1 describes how this modified greedy heuristic works.

Each column in the solution has an associated variable σ_j to know if it is redundant. We define $\sigma_j = \min_{i \in I_j}\{\eta_i - 1\}$, where η_i is the number of selected columns that can cover row i. Therefore, column j is redundant if and only if $\sigma_j > 0$. Observe that the variables k_j and η_i can easily be updated during the step to add a column ω to the solution as shown in Fig. 2. The initial value of $k_j = |I_j|$ for $j \in J$ and the initial value of $\eta_i = 0$ for $i \in I$. In this way, the overall time complexity for the addition of a column and the updating of k_j and η_i is $O(q)$, whereas the time required for each execution of line 4 is $O(n)$. Therefore the construction procedure requires $O(rn + q)$ time, where $r \leqslant m$ is the cardinality of the solution

procedure Meta-RaPS-SCP-Construction (I, J, %priority, %restriction)
1 Set the solution set to be empty: $X = \phi$
2 Set I^* be the set of the currently uncovered rows: $I^* = I$
3 **while** $I^* \neq \phi$
4 Select $f(c_j, k_j) = c_j / k_j$ as the priority rule, and find the best candidate ω such that
$$f(c_\omega, k_\omega) = \min_{j \in J \setminus X} (f(c_j, k_j))$$
5 P = RND(1,100)
6 **if** P > %priority **then**
7 Construct the candidate list CL:
8 $$CL = \{j : j \in J \setminus X \text{ and } c_j / k_j \leq c_\omega / k_\omega \times (1 + \%restriction/100)\}$$
9 Randomly select an element ω_1 from CL and set $\omega = \omega_1$
10 **end if**
11 Add element ω to the solution X: $X = X \cup \{\omega\}$; $I^* = I^* \setminus I_\omega$
12 **end while**
13 Remove redundant columns from X
14 **return** X
end Meta-RaPS-SCP-Construction

Fig. 1. Pseudo-code of Meta-RaPS SCP construction.

for each $i \in I_\omega$
Update the number of selected columns for each row *i*: $\eta_i = \eta_i + 1$
for each $j \in J_i$
$k_j = k_j - 1$
end for
end for

Fig. 2. Pseudo-code for solution updating procedure.

found. Since the average cardinality of the solution is $m \times d = q/n$, the time complexity for the average case is O(q).

3.2. Improvement heuristic

To improve the solution quality, a neighborhood search procedure can be applied after construction. This research defines the neighboring solutions as follows: If two solutions share at least one column, these two solutions are called neighboring solutions, that is, given two solutions X_1 and X_2, if $X_1 \cap X_2 \neq \Phi$, then X_1 and X_2 are neighboring solutions, otherwise, they are disjoint solutions.

A neighboring solution is obtained via a two-step procedure. First, a number of columns, as determined by a user-defined parameter, are randomly removed from the given feasible solution, so the solution will become infeasible because there are some uncovered rows. Then the partial solution is made feasible by solving a reduced size SCP that is made up of the uncovered rows and the columns that could cover these rows. The pseudo-code of the neighbor search procedure is shown in Fig. 3. The parameter search_magnitude is used to control how many columns will be removed from the solution. The number of removed columns is equal to $|X| \times$ search_magnitude. After the removing of columns, the reduced-size SCP is defined by Fig. 4.

$$\min \left\{ \sum_{j \in J'} c_j x'_j : \sum_{j \in J'} a_{ij} x'_j \geqslant 1, i \in I';\ x'_j = 0 \text{ or } 1, j \in J' \right\}, \tag{4}$$

procedure NeighborSearch (I, J, X, %priority, %restriction. search_magnitude, imp_iteration)
for iter = 1, …, imp_iteration
  Randomly remove columns from X, the maximum number of columns to remove equals to: $|X| \times search\_magnitude$
  Formulate a reduced-size SCP:
    $I' = I \setminus \bigcup_{m \in X} I_m \qquad J' = \bigcup_{i \in I'} J_i$
  Solve this reduced-size SCP:
    X' = Meta-RaPS-SCP-Construction (I', J', %priority, %restriction)
  Construct the neighboring solution: $X' = X' \cup X$
  Remove redundant columns from X'
  if the objective function value of X' is less than that of X **then**
    X = X'
  end if
end for
return X
end NeighborSearch

Fig. 3. Pseudo-code of neighbor search procedure.

procedure Meta-RaPS-SCP(I_0, J_0, %priority, %restriction, %improvement, max_iteration, search_magnitude, imp_iteration)
Perform preprocessing and get the reduced row set $I \subseteq I_0$ and column set $J \subseteq J_0$
Set the initial solutions to empty: $X^* = X = \phi$
$Z^* = Z^*_{before\ improvement}$ = LARGE_NUMBER {set to a large number}
Set the initial core problem to empty: $J_c = \phi$
for it = 1, …, max_iteration
  {the construction phase}
  Call Meta-RaPS-SCP_Construction (I, J, %priority, %restriction)
  and update the core problem J_c if desirable
  $Z = \sum_{j \in X} c_j$, where X is the solution from construction phase.
  if $Z < Z^*_{before\ improvement}$ **then** $Z^*_{before\ improvement} = Z$
  {the improvement phase}
  if $Z \leq (1 + \%improvement) \times Z^*_{before\ improvement}$ **then**
    NeighborSearch (I, J_c, X, %priority, %restriction, search_magnitude, imp_iteration)
  Penalize the worst elements in X
  {Update of the best solution}
  if ($Z < Z^*$) **then**
    $Z^* = Z$
    $X^* = X$
  end if
end for
return X^*
end Meta-RaPS-SCP

Fig. 4. Pseudo-code of final Meta-RaPS SCP algorithm.

where

$$I' = I \Big\backslash \bigcup_{m \in X} I_m \quad \text{and} \quad J' = \bigcup_{i \in I'} J_i. \tag{5}$$

A simple heuristic, such as a greedy heuristic, might be used to solve this reduced-size SCP. However, from our computational experience, applying the randomized greedy construction heuristic, Meta-RaPS SCP Construction, to the reduce-size SCP usually generates much better neighboring solutions. After solving this reduced-size SCP, a neighboring solution is obtained by combining the solution of the small SCP (X') and the partial solution (X) and removing the redundant columns. The number of neighboring solutions to explore is controlled by the parameter imp_iteration.

Notice in Fig. 3 that once the improvement procedure finds a better solution, the next neighbor search will be executed around this better solution. A different neighbor search strategy, in which the neighbor search will always be executed around the initial solution, has also been investigated, but it usually generates inferior results.

The time complexity of the neighbor search procedure is $O(r'n' + q')$, where $r' \leqslant m'$ is the cardinality of the solution for the reduced-size SCP, $m' = |I'|$, $n' = |J'|$, and $q' = \sum_{i \in I'} \sum_{j \in J'} a_{ij}$. The time complexity for the average case is $O(q')$. Since the neighbor search methods are embedded in Meta-RaPS, the time complexity for the overall procedure is $O[(rn + q) + \Lambda(r'n' + q')]$, and that of the average case is $O(q + \Lambda q')$, where $\Lambda \equiv imp\_iteration$.

3.3. Some measures to improve solution quality

Based on initial experimentation using SCP test problems from the OR-Library, the algorithm that includes the randomized greedy heuristic and the neighbor search procedure (i.e. basic Meta-RaPS SCP) generates very promising results. However, not all the optimal solutions are obtained. Simply adjusting the parameters, such as the %priority or %restriction, may improve the solution quality slightly, but still cannot lead to all optimal solutions. Furthermore, it is very difficult to find a single set of parameters that would be optimal for all problems. These initial results led to the development of two methods to improve the solution quality; randomizing the selection of priority rules and penalizing the worst columns.

3.3.1. Randomizing the selection of priority rules

One reason the solutions converge to a local optima for some problems is the bias introduced by the priority rule in the greedy heuristic. Altering the priority rule in a way such that the search will be guided to the global optimal region is desired. Since the solutions have been very close to the optimal solutions, it is believed that some slight modifications to this priority rule c_j/k_j would lead to the global optima. So the effect of changing the ratio of the influence of c_j and k_j is investigated by applying the following functions: c_j/k_j^2, $c_j/k_j \log(1 + k_j)$, $c_j^{1/2}/k_j$, $c_j/\log(1 + k_j)$ and $c_j/k_j^{1/2}$. From computational experience, it is difficult to find a single priority rule that leads to the optimal solution for all test problems. Inspired by the previous research of Vasko and Wilson (1984), it is discovered that randomizing the selection of priority rules in Meta-RaPS often results in better solutions for the SCP. Now before evaluating all feasible columns and adding a column to the solution, a priority rule will be randomly selected from a pool of these priority rules. So, the only necessary change to the pseudo-code in Fig. 1 is in the line 4. We now need to randomly select a priority rule $f(c_j, k_j)$ before evaluating the greedy scores. From our computational experience, it is discovered that the priority rules with logarithm are relatively time consuming, and incorporating these two priority rules cannot gain much in the solution quality. Using other four priority rules can find all the optimal solutions, however, the results deteriorate if applying fewer than these four priority rules. So finally the four priority rules c_j/k_j, c_j/k_j^2, $c^{1/2}/k_j$ and $c_j/k_j^{1/2}$ are used.

It should be noted this improved Meta-RaPS SCP construction does little to improve the solution quality of the unicost problems. Since $c_j = 1$ for all columns in the unicost problems, this method cannot change the column's ranking, it only affects the size of the candidate list. So, applying this method for unicost SCP is actually the same as adjusting the parameter %restriction dynamically, which does not improve the solution quality too much from the tests. Therefore this method is not recommended for unicost SCP.

3.3.2. Penalizing the worst columns

Another method that works in some cases is the penalizing method. Given a feasible solution at each iteration, the worst columns are penalized so that the probability these columns will be selected in the future is decreased. In this research, the columns with the maximum value of $c_j/(u_j \times |I_j|)$ are penalized, and the penalty added to the cost of a column is calculated by: $\varsigma \times c_j \times (1 - \exp(-u_j))$, where ς is a parameter to control the value of penalty, and u_j is the penalized times for column j. u_j is increased by 1 whenever the column j is penalized. In this way, if a column seems inevitable in the optimal solution, the possibility that this column would be penalized, and the value of the penalty, if any, would be decreased. It is discovered that penalizing a larger number of columns often results in better solutions than only penalizing the worst columns, therefore, the parameter %penalty is used to control the number of columns to be penalized, such that the columns with the value of $c_j/(u_j \times |I_j|) \geqslant \max(c_n/(u_n \times |I_n|: n \in X) \times (1 - \%\text{penalty}/100)$ will be penalized. We found the following parameters are good for the penalizing method: $\varsigma = 2.0$ and $\%\text{penalty} = 2$.

This penalizing method works when a very high %priority (>98%) or a very low %restriction (<2%) is used. Since the solutions are highly similar in these situations, penalizing the worst columns in a solution help to guide the search into an unexplored region. Grossman and Wool (1997) found that for some extremely sparse set covering problems, the greedy algorithm always works better than other approximation algorithms. It is found in this research that using a very high %priority is better than a lower %priority for these problems. In these cases, the penalizing method does improve the solution quality.

3.4. Some measures to improve computational speed

In addition to improving solution quality, we developed two methods to reduce the run time of Meta-RaPS SCP.

3.4.1. Preprocessing

Preprocessing is a popular method to speedup the algorithm. A number of preprocessing methods for the SCP have been proposed in the literature (Beasley, 1987). In this research, two of those found to be most effective are used.

- *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. As expected, this reduction is ineffective for unicost problems.
- *Column inclusion*: If a row is covered by only one column after the above domination, this column must be included in the optimal solution.

3.4.2. Defining core problem

Since the neighbor search procedure is iterated *imp_iteration* times for each constructed solution, the improvement phase is more computationally challenging than the construction phase. Therefore, reducing the run time of the neighbor search procedure is more significant to reduce total run time than speeding up the construction heuristic. In order to reduce the computation time, this algorithm integrates a procedure to define the core problem, which includes only a small subset of interesting columns, for the improvement

heuristic to work on, such that the reduced-size SCP (4) and (5) will become even smaller. The core problem is defined as those columns selected at least once in the candidate list during the construction phase. Initially the core problem J_c is set to be empty. During each construction step, the core problem is updated as $J_c = J_c \cup CL$. So only a slight modification in the construction is needed to define the core problem. After construction the new reduced-size SCP for neighbor search procedure is as follows:

$$\min\left\{\sum_{j\in J'\cap J_c} c_j x_{j'} : \sum_{j\in J'\cap J_c} a_{ij}x'_j \geqslant 1, i\in I'; \quad x'_j = 0 \text{ or } 1, \ j\in J'\cap J_c\right\}, \tag{6}$$

where

$$I' = I\Big\backslash \bigcup_{m\in X} I_m, \quad J' = \bigcup_{i\in I'} J_i, \quad \text{and } J_c \text{ is the core problem.} \tag{7}$$

Apparently this SCP has at least one feasible solution, i.e. the feasible solution from the construction heuristic.

Notice the size of the core problem will increase as more iterations have been run for this algorithm. Hopefully, the columns in the core problem will enter a steady state after a certain number of iterations. Although defining core problem can greatly decrease the total execution time, it also has some disadvantages. Since it dismisses some columns from consideration in the improvement phase, the possibility to find an optimal solution might be decreased. As a compensation to this negative effect, it is found that the parameter %restriction and %improvement should be set a little larger to find more optimal solutions if the core problem is defined.

3.5. Overall algorithm

The final version of Meta-RaPS SCP includes the following elements: preprocessing, Meta-RaPS SCP Construction, neighbor search improvement procedure, penalizing method and core problem definition.

4. Experimental results and analysis

4.1. Experimental study on non-unicost SCP

The effectiveness of the Meta-RaPS SCP heuristic was evaluated using 65 non-unicost SCP test instances from Beasley's OR Library (1990). Because nearly all the SCP heuristics that have been developed during the last fifteen years were tested using these problems, comparisons can be made among solution techniques. These instances are divided into 11 sets as in Table 1, in which Density is the percentage of non-zero entries in the SCP matrix.

First, different configurations of Meta-RaPS SCP are tested, namely, basic Meta-RaPS, Meta-RaPS w/randomized priority rules, and Meta-RaPS w/randomized priority rule and core problem. These variations are described in Table 2.

When applying the Meta-RaPS method, several parameters must be set. Coy et al. (2001) note that it is often difficult to find appropriate parameter settings for meta-heuristics and common procedures have ranged from simple trial-and-error to complicated sensitivity analysis. A trial and error procedure was used in this research. The first instances in problem set 4, 5, 6, A, B, C and D are selected as representative and different combinations of parameters are applied for these instances. More specifically, we take %priority from $\{5, 25, 55, 100\}$, %restriction from $\{5, 15, 25, 35, 45\}$, %improvement from $\{10, 15, 20\}$, imp_magnitude from $\{0.3, 0.4\}$. The parameter max_iteration is chosen from $\{100, 500, 1000\}$ and imp_iteration is chosen

Table 1
Non-unicost instances

Set	No. of instances	No. of rows (m)	No. of columns (n)	Range of cost	Density (d)	Optimal solution
4	10	200	1000	1–100	2%	Known
5	10	200	2000	1–100	2%	Known
6	5	200	1000	1–100	5%	Known
A	5	300	3000	1–100	2%	Known
B	5	300	3000	1–100	5%	Known
C	5	400	4000	1–100	2%	Known
D	5	400	4000	1–100	5%	Known
NRE	5	500	5000	1–100	10%	Unknown
NRF	5	500	5000	1–100	20%	Unknown
NRG	5	1000	10,000	1–100	2%	Unknown
NRH	5	1000	10,000	1–100	5%	Unknown

Table 2
Different variations of Meta-RaPS SCP

	Construction	Core problem
Basic Meta-RaPS	Single priority rule (c_j/k_j)	No
Meta-RaPS w/randomized priority rules	Randomized priority rules	No
Meta-RaPS w/randomized priority rules and core problem	Randomized priority rules	Yes

from $\{300, 400, 500\}$, so 1080 different combinations of parameter settings are tested. The final parameters used for all test instances are listed in Table 3. The same %priority, %restriction, %improvement, and imp_magnitude are used for basic Meta-RaPS and Meta-RaPS w/randomized priority rule. If the core problem is defined, larger values for %restriction and %improvement are applied. A larger number of iterations for Meta-RaPS and the improvement heuristic are applied for larger instances, NRG and NRH.

The experiments were carried out on an Intel Pentium IV 1.7 GHz PC. While more detailed results can be found in Lan (2004), the summarized results are compared in Table 4 (all times are in CPU seconds), in which GAP is the percentage of the deviation from the optima (Opt.) or best known solution (BKS), i.e. GAP = (solution-BKS)/BKS × 100, the total time is the total execution time for the program, and the solution time reports the time when the best solution is found for the first time. Some conclusions can be drawn from these results. Firstly, it is clear that Meta-RaPS with randomized priority rules performs the best in terms of the solution quality, as it has a zero deviation from all the best-known solutions. Secondly, with the definition of core problem, the execution time has been greatly decreased. The average solution time is calculated over all test problems and over optima hits (those problems for which optimal/best-known solu-

Table 3
Parameter setting for non-unicost instances

	%priority	%restriction	%improvement	search_magnitude	max_iterations	imp_iterations
Basic Meta-RaPS	5	35	15	0.3	100 for set 4-set NRF 1000 for set NRG and set NRH	400 for set 4-set NRF 500 for set NRG and set NRH
Meta-RaPS w/randomized priority rules	5	35	15	0.3		
Meta-RaPS w/randomized priority rules and core problem	5	40 for set 4-set NRF 45 for set NRG and set NRH	20 for set 4-set NRF 25 for set NRG and set NRH	0.3		

Table 4
Comparison of different variations of Meta-RaPS SCP for non-unicost instances

	Optimal solutions found	Avg. GAP	Avg. total time	Avg. solution time	
				Over all problems	Over optima hits
Basic Meta-RaPS	55/65	0.10	982.78	206.13	199.09
Meta-RaPS w/randomized priority rules	65/65	0	878.37	133.79	133.79
Meta-RaPS w/randomized priority rules and core problem	63/65	0.06	549.84	97.98	108.41

tions are found), and the time reduction is about 26.77% and 18. 97%, respectively. The presumption that the reduction in solution time is caused by the inferior solution quality might be safely precluded.

Furthermore, Meta-RaPS with randomized priority rules can be compared to the greedy heuristics (G) as well as the following existing SCP heuristics.

- Be: the Lagrangian heuristic by Beasley (1990).
- BeCh: the genetic algorithm by Beasley and Chu (1996).
- CFT: the Lagrangian heuristic by Caprara et al. (1999).
- PROGRES: a probabilistic greedy search heuristic by Haouari and Chaouachi (2002).
- IGA: an indirect genetic algorithm by Aickelin (2002).

Table 5 report the detailed results for Meta-RaPS SCP and the greedy heuristics, which is simulated in Meta-RaPS by setting 100% priority, and 0% improvement. In Table 5, "Time" is the solution time when the best solution is found for the first time.

Tables 6 and 7 provide a summary/comparison on the solutions quality for all these different heuristics. The number of optimal or best-known solutions found by all these heuristic is listed in Table 6. Meta-RaPS SCP found all the optimal or best-known solutions for the 65 non-unicost instances.

Table 7 is the summarized results for the solution quality that reports the average GAP. From this table, Meta-RaPS is only one of the two heuristics that have a zero deviation from the best-known or optimal solutions for these test problems.

To compare the computation time for different heuristics is difficult since the computational efficiency is affected by many factors such as the implementation language, the complier, the executing environment including CPU, RAM and Operating system, and it is even affected by the programmer's skills. For an approximate comparison, people often transfer the actual computation time from different computers to the same computer according to the performance report for various kinds of computers. For example, Caprara et al. (1999) adjusted the computation time of many algorithms and heuristics for SCP to DEC-station 5000/240 seconds according to the performance report by Dongarra (2004). There are at least two problems associated with the application of this performance report. Firstly, since it is not possible to include all kinds of computers in this report, similar computers must be used for an approximation. Secondly, as this report is based on the performance for solving standard linear equations, using it for evaluating heuristics for SCP often leads to unavoidable errors. From our experiments for our algorithm on an Intel PIV 1.7 GHz PC and an Intel PII 400MHz PC, the average error would be as high as 52.43%. To provide more reliable information, we report both the actual computation time in different computers and the transformed time to DECstation 5000/240 seconds. Table 8 reports the summarized data for computation time. Meta-RaPS performs very well for smaller problem sets, but is slow for problem sets NRG and NRH, since a much larger number of iterations are applied for these instances. It is slower than CFT for all the problem sets, but faster than BeCh for seven problem sets except set A, C, D and NRH.

Table 5
Detailed results for non-unicost instances

Name	BKS	Greedy		Meta-RaPS		Name	BKS	Greedy		Meta-RaPS	
		Sol.	Time	Sol.	Time			Sol.	Time	Sol.	Time
4.1	429	439	0	429	1.36	5.1	253	271	0	253	1.55
4.2	512	547	0	512	0.24	5.2	302	329	0	302	0.59
4.3	516	546	0	516	0.29	5.3	226	232	0.01	226	1.14
4.4	494	510	0	494	0.39	5.4	242	253	0	242	0.32
4.5	512	519	0	512	0.9	5.5	211	220	0	211	0.33
4.6	560	594	0	560	0.1	5.6	213	234	0	213	0.14
4.7	430	447	0	430	0.04	5.7	293	302	0	293	1.03
4.8	492	502	0	492	1.46	5.8	288	308	0.01	288	0.08
4.9	641	672	0	641	3.47	5.9	279	290	0	279	0.04
4.10	514	521	0	514	0.08	5.10	265	275	0	265	0.03
6.1	138	147	0	138	0.25	A.1	253	271	0	253	6.22
6.2	146	160	0	146	0.02	A.2	252	267	0	252	0.28
6.3	145	152	0	145	0.02	A.3	232	244	0	232	16.94
6.4	131	137	0	131	0.34	A.4	234	246	0	234	0.04
6.5	161	178	0	161	1.02	A.5	236	247	0.01	236	9.37
B.1	69	73	0	69	0.14	C.1	227	246	0	227	0.43
B.2	76	78	0	76	0.53	C.2	219	231	0.02	219	12.89
B.3	80	85	0	80	0.62	C.3	243	256	0	243	26.24
B.4	79	85	0	79	2.25	C.4	219	239	0	219	24.29
B.5	72	76	0	72	0	C.5	215	228	0.01	215	1.79
D.1	60	68	0.01	60	3.13	NRE.1	29	31	0.02	29	0.73
D.2	66	70	0	66	13.59	NRE.2	30	34	0	30	46.17
D.3	72	78	0.02	72	1.31	NRE.3	27	32	0	27	5.95
D.4	62	65	0	62	0.2	NRE.4	28	32	0.01	28	39.64
D.5	61	71	0.01	61	0.29	NRE.5	28	31	0	28	0.81
NRF.1	14	17	0.02	14	4.29	NRG.1	176	194	0.03	176	298.97
NRF.2	15	16	0.02	15	3.8	NRG.2	154	165	0.03	154	222.34
NRF.3	14	16	0.01	14	1.84	NRG.3	166	179	0.01	166	21.56
NRF.4	14	15	0.01	14	5.44	NRG.4	168	184	0.03	168	194.21
NRF.5	13	15	0.01	13	33.27	NRG.5	168	181	0.02	168	47.57
NRH.1	63	71	0.04	63	3917.08						
NRH.2	63	69	0.03	63	238.45						
NRH.3	59	65	0.05	59	783.2						
NRH.4	58	66	0.03	58	1358.28						
NRH.5	55	62	0.04	55	5.62						

Table 6
Number of optimal/best-known solutions

CFT	Meta-RaPS	BeCh	IGA	Be	PROGRES	Greedy
65/65	65/65	61/65	61/65	22/65	20/65	0/65

4.2. Experimental study on unicost SCP

While most heuristics discussed in the previous section have not reported results for unicost SCP, Meta-RaPS SCP performs well on unicost SCP. The test instances used are the set E, CLR and CYC from the OR-Library, as shown in Table 9.

Table 7
Summarized results for the solution quality (average GAP)

Problem set	CFT	Meta-RaPS	BeCh	IGA	PROGRES	Be	Greedy
4	0.00	0.00	0.00	0.00	0.57	0.06	3.78
5	0.00	0.00	0.09	0.00	0.88	0.18	5.51
6	0.00	0.00	0.00	0.00	0.69	0.56	7.22
A	0.00	0.00	0.00	0.00	0.75	0.82	5.61
B	0.00	0.00	0.00	0.00	0.00	0.81	5.57
C	0.00	0.00	0.00	0.00	0.87	1.93	6.88
D	0.00	0.00	0.00	0.32	0.00	2.75	9.79
NRE	0.00	0.00	0.00	0.00	0.00	3.5	12.75
NRF	0.00	0.00	0.00	0.00	1.43	7.16	12.98
NRG	0.00	0.00	0.13	0.13	1.18	4.83	8.49
NRH	0.00	0.00	0.63	1.30	1.68	8.12	11.78
Overall	0.00	0.00	0.08	0.16	0.72	2.36	8.21

Table 8
Comparison of computation time for non-unicost instances

Problem set	CFT actual time on DECstation 5000/240 (seconds)	Meta-RaPS		BeCh	
		Actual time on Intel PIV 1.7 G (seconds)	Expected time on DECstation 5000/240 (seconds)	Actual time on Graphics Indigo (R4000, 100) (seconds)	Expected time on DECstation 5000/240 (seconds)
4	6.5	0.83	56.85	57.59	163
5	3.2	0.58	39.72	190.87	540.2
6	9.4	0.33	22.60	20.21	57.2
A	106.6	6.57	449.98	52.79	149.4
B	7.4	0.72	49.31	54.9	155.4
C	66	13.13	899.28	70.38	199.2
D	17.2	3.70	253.42	81.41	230.4
NRE	118.2	18.66	1278.03	3082.55	8724.2
NRF	109	9.73	666.41	976.90	2764.8
NRG	504.8	156.93	10,748.22	4540.83	12,851.4
NRH	858.2	1260.53	86,334.41	2240.70	6341.6
Overall	164	133.79	9163.35	1033.5	2925

For unicost SCP, only the single priority rule c_j/k_j was used, as noted in Section 3.3.1, randomizing the selection of priority rules cannot help to improve the solution quality. We also applied the above-mentioned trial-and-error method to find the parameters of the Meta-RaPS algorithm for the unicost instances. As shown in the Table 10, two groups of parameters, which differ only in the value of %priority, are found to be effective for different instances. %priority = 100 works the best for sparser instances, such as the set CYC, while %priority = 5 works better for other instances. As mentioned in Section 3.3.2, this is consistent with the findings of Grossman and Wool (1997).

The test results for the 15 unicost SCP instances from the OR-library are reported in Table 11, in which the solutions times are given in Intel PIV 1.7 GHz seconds. Notice that the best solutions for CLR and CYC are the explicit solutions by Goldberg and Russell (1995) and Harborth and Nienborg (1994), respectively. From this table, applying the penalty method can improve the solution quality if a very high %priority is used, while it does not improve the solutions for a lower value of %priority. The algorithm updates

Table 9
Unicost instances

Set or instance	No. of instances	No. of rows (m)	No. of columns (n)	Density (d)	Optimal solution
E	5	50	500	20%	Known
CLR.10	1	511	210	12.3	Unknown
CLR.11	1	1023	330	12.4	Unknown
CLR.12	1	2047	495	12.5	Unknown
CLR.13	1	4095	715	12.5	Unknown
CYC.6	1	240	192	2.1	Known
CYC.7	1	672	448	0.9	UnKnown
CYC.8	1	1792	1024	0.4	UnKnown
CYC.9	1	4608	2304	0.2	UnKnown
CYC.10	1	11,520	5120	0.08	UnKnown
CYC.11	1	28,160	11,264	0.04	Unknown

Table 10
Parameter setting for unicost instances

	%priority	%restriction	%improvement	search_magnitude	max_iterations	imp_iterations
Group1	5	15	15	0.3	100	200
Group2	100					

Table 11
Test results for unicost instances

Name	BKS	Meta-RaPS w/o penalty				Meta-RaPS w/penalty			
		%priority = 5		%priority = 100		%priority = 5		%priority = 100	
		Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
E.1	5	5	0.16	5	0	5	0	5	0
E.2	5	5	0	6	0	5	0	5	0.03
E.3	5	5	0	5	0	5	0	5	0
E.4	5	5	0	6	0	5	0.07	5	0.12
E.5	5	5	0.17	5	0	5	0	5	0
CLR.10	25	25	0.18	25	0.2	25	0.2	25	0.05
CLR.11	23	23	17.43	23	5.62	23	1.39	23	3.03
CLR.12	26	**23**	37.84	**23**	1.2	**23**	20.72	**23**	4.13
CLR.13	26	26	164.18	**23**	47.13	29	44.3	**23**	48.74
CYC.6	60	60	0.14	60	0	60	0.55	60	0
CYC.7	144	149	4.47	144	0	151	4.28	144	0
CYC.8	344	361	47.98	345	16.7	367	13.59	344	38.91
CYC.9	780	870	285.14	798	14.04	876	352.27	793	88.36
CYC.10	1792	2002	423.06	1892	455.08	2016	982.92	1826	80.56
CYC.11	4103	4539	8564.17	4268	441.2	4532	5471.91	4140	12,656.75
Avg. GAP		2.05		1.94		3.12		−1.24	
Avg. solution time		636.33		65.41		459.48		861.38	

the best-known solutions for two instances: CLR.12 and CLR.13 whose objective values are highlighted in Table 11.

Table 12
Comparison of solution quality for unicost instances

Heuristics	RR	Gr	Alt-Gr	NN	R-Gr	MF	Meta-RaPS
Avg. Gap	19.10	9.26	6.05	8.83	8.61	5.27	−1.24

Table 13
Comparison of computation time for unicost instances

Name	Actual time on IBM RS6000/370 (seconds)					MF		Meta-RaPS	
	RR	Gr	Alt-Gr	NN	R-Gr	Actual time on Intel PII 400M (seconds)	Expected time on IBM RS6000/370 (seconds)	Actual time on Intel PIV 1.7 G (seconds)	Expected time on IBM RS6000/370 (seconds)
E.1	0.6	0.0	0.0	1.0	0.0	0.15	0.05	0	0
E.2	0.6	0.0	0.0	1.0	0.0	0.14	0.04	0.03	0.035
E.3	0.6	0.0	0.0	1.0	0.0	0.15	0.05	0	0
E.4	0.6	0.0	0.0	1.0	0.0	0.14	0.04	0.12	0.14
E.5	0.6	0.0	0.0	1.0	0.0	0.16	0.05	0	0
CLR.10	3.0	1.0	1.0	3.0	2.0	0.36	0.11	0.05	0.06
CLR.11	7.0	4.0	4.0	9.0	7.0	1	0.31	3.03	3.49
CLR.12	20.0	15.0	13.0	25.0	21.0	3.2	0.1	4.13	4.76
CLR.13	52.0	42.0	38.0	68.0	58.0	10	3.11	48.74	56.17
CYC.6	0.0	0.0	0.0	1.0	0.0	0.08	0.02	0	0
CYC.7	1.0	0.0	0.0	7.0	2.0	0.2	0.06	0	0
CYC.8	6.0	0.0	0.0	39.0	12.0	0.62	0.19	38.91	44.84
CYC.9	13.0	2.0	2.0	192.0	58.0	1.6	0.5	88.36	101.82
CYC.10	137.0	7.0	7.0	941.0	288.0	3.9	1.21	80.56	92.84
CYC.11	852.0	26.0	28.0	4907.0	1455.0	9.6	2.99	12,656.75	14,585.40

Meta-RaPS SCP is compared with other heuristic algorithms from the literature. The heuristic RR (Peleg et al., 1996), Gr (Chvatal, 1979), Alt-Gr, NN and R-Gr (Grossman and Wool, 1997) are the five best heuristics out of nine algorithms tested by Grossman and Wool (1997). MF is the mean field approach developed by Ohlsson et al. (2001). Table 12 reports the solution quality for these heuristics. From Table 12, the Meta-RaPS SCP algorithm performs the best among all the available algorithms that have been applied for these instances. The average GAP for all the unicost instances is −1.24.

The computation times for these heuristics are reported in Table 13. The solution times for RR, Gr, Alt-Gr, NN and R-Gr are given in IBM RS6000/370 seconds. The solution times for MF are given in Intel PII 400 MHz seconds, and the solutions times for Meta-RaPS are given in Intel PIV 1.7 GHz seconds. We also give the solution times transformed to IBM RS6000/370 seconds for MF and Meta-RaPS according to Dongarra (2004). For small instances such as set E, Meta-RaPS works very fast, and this is comparable to the greedy algorithms. It is very slow for the largest instance CYC.11. However, if we reduce to the number of iterations from 100 to 10, the solution for CYC.11 will be 4184, which still gives the better result than other compared heuristics, and the computation time will be 117.26 seconds.

5. Conclusions and future research

In this research, an effective heuristic for solving the set covering problems was developed. This algorithm systematically introduces randomness into a simple greedy heuristic to construct a feasible solution.

The two novel features of this algorithm are to use the construction heuristics once again to generate neighboring solutions in the improvement phase and to randomize the selection of priority rules. In addition, penalizing the worst columns if the solution searching space is highly condensed was employed to enhance the performance of the basic Meta-RaPS. A simple approach to define the core problem is proposed and it can considerably improve the computation speed at the cost of slightly inferior solution quality.

From the computational experiments, Meta-RaPS SCP ranks in the top 3 among all known SCP heuristics for solving non-unicost problems in terms of the solution quality. It finds 65/65 best-known solutions for the test instances in the OR-Library. These results are quite significant since Meta-RaPS SCP is based on the modification of a simple greedy heuristic. Furthermore, another important attractiveness of this heuristic is its outstanding performance for solving unicost problems. It currently is the best in solution quality among all the heuristic algorithms available in the literature for solving the test instances in the OR-Library.

Future research will be directed to improve the performance for larger SCP problems, such as set NRG and NRH. In addition, the introduction of memory between Meta-RaPS iterations is being investigated to further improve the computational speed. Some techniques developed in this research will also be used in other applications of Meta-RaPS such as the multi-dimensional knapsack problem and the bin packing problem.

References

Arcus, A.L., 1966. COMSOAL: A computer method of sequencing operations for assembly lines. International Journal of Production Research 4, 259–277.

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

Balas, E., 1983. A class of location, distribution and scheduling problems: Modeling and solution methods. In: Proceedings of the Chinese–US Symposium on System Analysis. J. Wiley and Sons, New Rouk.

Balas, E., Carrera, M., 1996. A dynamic subgradient-based branch-and-bound procedure for set covering. Operations Research 44, 875–890.

Beasley, J.E., 1987. An algorithm for set covering problem. European Journal of Operational Research 31, 85–93.

Beasley, J.E., 1990. A Lagrangian heuristic for set covering problems. Naval Research Logistics 37, 151–164.

Beasley, J.E., Chu, P.C., 1996. A genetic algorithm for the set covering problem. European Journal of Operational Research 94, 392–404.

Beasley, J.E., JØrnsten, K., 1992. Enhancing an algorithm for set covering problems. European Journal of Operational Research 58, 293–300.

Caprara, A., Fischetti, M., Toth, P., 1999. A heuristic method for the set covering problem. Operations Research 47 (5), 730–743.

Caprara, A., Fischetti, M., Toth, P., 2000. Algorithms for the set covering problem. Annals of Operations Research 98, 353–371.

Ceria, S., Nobili, P., Sassano, A., 1998a. Set Covering Problem. Annotated Bibliographies in Combinatorial Optimization. John Wiley and Sons, New York, pp. 415–428.

Ceria, S., Nobili, P., Sassano, A., 1998b. A Lagrangian-based heuristic for large-scale set covering problems. Mathematical Programming 81, 215–228.

Chvatal, V., 1979. A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4, 233–235.

Coy, S., Golden, B., Runger, G., Wasil, E., 2001. Using experimental design to find effective parameter setting for heuristics. Journal of Heuristics 7, 77–97.

DePuy, G.W., Moraga, R.J., Whitehouse, G., 2005. Meta-RaPS: A simple and effective approach for solving the traveling salesman problem. Transportation Research Part E: Logistics and Transportation Review 41 (2), 115–130.

DePuy, G.W., Whitehouse, G.E., Moraga, R.J., 2002. Using the Meta-RaPS approach to solve combinatorial problems. In: Proceedings of the 2002 Industrial Engineering Research Conference, May 19–21, Orlando, Florida. Computers & Industrial Engineering, submitted for publication.

Dongarra, J.J., 2004. Performance of various computers using standard linear equations software. Technical Report No. CS-89-85, Computer Science Department, University of Tennesse.

Feo, T., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Fisher, M., Kedia, P., 1990. Optimal solution of set covering/partitioning problems using dual heuristics. management Science 36, 674–688.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA.

Goldberg, M.K., Russell, H.C., 1995. Toward computing m(4). Ars Combinatoria 3 (9), 139–148.

Grossman, T., Wool, A., 1997. Computational experience with approximation algorithms for the set covering problem. European Journal of Operational Research 101, 81–92.

Haouari, M., Chaouachi, J.S., 2002. A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem. Journal of the Operational Research Society 53, 792–799.

Harborth, H., Nienborg, H., 1994. Maximum number of edges in a six-cube without four-cycles. Bulletin of the Institute of Combinatorics and Applications 12, 55–60.

Jacobs, L., Brusco, M., 1995. Note: A local-search heuristic for large set-covering problems. Naval Research Logistics 42, 1129–1140.

Lan, G., 2004. An effective and simple heuristic based on Meta-RaPS to solve large-scale set covering problems, M.S. Thesis, University of Louisville.

Moraga, R.J., DePuy, G.W., Whitehouse, G.E., 2005. Meta-RaPS approach for the 0–1 multidimensional knapsack problem. Computers & Industrial Engineering 48 (1), 83–96.

Ohlsson, M., Peterson, C., Söderberg, B., 2001. An efficient mean field approach to the set covering problem. European Journal of Operational Research 133, 583–595.

Peleg, D., Schechtman, G., Wool, A., 1996. Randomized approximation of bounded multicovering problems, Algorithmica.

Vasko, F.J., Wilson, G.R., 1984. An efficient heuristic for large set covering problems. Naval Research Logistics Quarterly 31, 163–171.

Whitehouse, G.E., DePuy, G.W., Moraga, R.J., 2002. Meta-RaPS approach for solving the resource allocation problem. In: Proceedings of the 2002 World Automation Congress, June 9–13, Orlando, Florida.