# A Weighting-Based Local Search Heuristic Algorithm for the Set Covering Problem

Chao Gao, Thomas Weise, Jinlong Li

***Abstract*—The Set Covering Problem (SCP) is NP-hard and has many applications. In this paper, we introduce a heuristic algorithm for SCPs based on weighting. In our algorithm, a local search framework is proposed to perturb the candidate solution under the best objective value found during the search, a weighting scheme and several search strategies are adopted to help escape from local optima and make the search more divergent. The effectiveness of our algorithm is evaluated on a set of instances from the OR-Library and Steiner triple systems. The experimental results show that it is very competitive, for it is able to find all the optima or best known results with very small runtimes on non-unicost instances from the OR-Library and outperforms two excellent solvers we have found in literature on the unicost instances from Steiner triple systems. Furthermore, it is conceptually simple and only needs one parameter to indicate the stopping criterion.**

**This is a preview version of the paper [1] (see page 8 for the reference). Read the full piece in the proceedings.**

## I. Introduction

The SCP is a combinatorial optimization problem: Given an universal set $X$ and a set $S$ which contains many subsets of $X$ with $\bigcup_{\forall s \in S} = X$. Each element in $S$ is associated with a cost. The goal is to find a $F \subseteq S$ of the smallest total cost but still contains all elements in $X$, i.e., ($\bigcup_{\forall s \in F} = X$). In literature, instances of this problem are generally presented by a zero-one matrix $A = \{a_{ij}\}_{m \times n}$ that contains $m = |X|$ rows and $n = |S|$ columns. Each column $j (j \in N,$ where $N$ is the set of all columns) has a *cost*, and $a_{ij} = 1$ means column $j$ can cover row $i$. The task is to find a set of columns that ensure all rows in $M (M$ represents all the rows) are covered and minimize the total cost. It can be formally written as follows:

$$\min \sum_{j \in N} j.cost \cdot x_j \tag{1}$$

subject to

$$\sum_{j \in N} a_{ij} \geq 1 \tag{2}$$

$$x_j \in \{0, 1\}, \forall i \in M, \forall j \in N \tag{3}$$

When all columns have the same cost, it is called the unicost set covering problem (USCP). Otherwise, it refers to the non-unicost SCP.

Chao Gao, Thomas Weise and Jinlong Li are with the Department of Computer Science, University of Science and Technique of China, Hefei, China (email: chao.gao.ustc@gmail.com, tweise@ustc.edu.cn and jlli@ustc.edu.cn).

The SCP is NP-hard in the strong sense [2] and has many applications, such as crew scheduling in railway and mass-transit companies to job assignment in manufacturing and service location [3, 4].

In this paper, we propose a stochastic local search heuristic algorithm [5] for SCPs based on weighting. There are three major features of our algorithm.

1) A local search framework with upper bound restriction to iteratively improve the candidate solution
2) Tabu strategies to avoid possible cycles during the search
3) A weighting scheme to help escape from local optima

Experimental results show that our algorithm is very competitive on the benchmark instances, for it is able to find all the optima or best known solutions within short runtimes. Moreover, it works very well both on unicost and non-unicost problems and outperforms the state-of-the-art methods discussed in the related work on the unicost Steiner triple system instances.

The rest of this paper is organized as follows: In Section II, we discuss the related work and state-of-the-art. Section III presents our algorithm systematically. The experimental results are given in Section IV. Finally, we conclude this paper in Section V.

## II. Related work

Many algorithms have been proposed over the years. The exact algorithms [6, 7, 8, 9] are mostly based on branch-and-bound or branch-and-cut. Caprara et al. [10] compared different exact algorithms and pointed out that CPLEX has the best performance. Though exact algorithms can guarantee the optimality of the found solutions, they always require substantial computational efforts when facing large scale problems, thus become infeasible [10]. Therefore, large instances of SCP are typically solved using heuristic algorithms.

The simplest approximation algorithm for SCP is the greedy algorithm [11]. Due to the myopic and deterministic nature, greedy algorithms can rarely produce good quality solutions, thus researchers have tried to improve the solution quality of the greedy algorithm by introducing randomness, and such randomized and probabilistic greedy methods [12, 13] usually produce better results than the pure greedy one.

Besides greedy algorithms and their variants, some modern heuristics have also been proposed, such as Genetic Algorithm [14], Simulated Annealing [15]. However, one drawback of these meta-heuristics is that the cost information plays an important role and thus they hardly work effectively both on unicost and non-unicost problems. Lan et al. [16] therefore proposed the Meta-RaPS approach. Meta-RaPS works effectively for both the unicost and non-unicost SCP.

Several very effective heuristics [17, 18, 19] for SCPs are based on the linear programming relaxation (LP) or Lagrangian relaxation. By using LP or Lagrangian relaxation, which can provide reliable information to evaluate the goodness between columns, they usually adopt techniques to reduce the problem size. For example, Caprara et al. [18] defined a *core problem* by fixing some variables $x_j$ to 0 and then updating it dynamically. Later, Yagiura et al. [20] proposed a 3-flip local search algorithm (3-FNLS) based on Lagrangian relaxation. Incorporating with the subgradient method in [18] to solve the Lagrangian dual relaxation, 3-FNLS conducts the local search many times and each time the fixing variables and penalty weights are updated, thereby realizing a strategic oscillation. With a sophisticated implementation, 3-FNLS achieves remarkable results on instances from the OR-Library [21].

However, the effectiveness of the LP relaxation or Lagrangian relaxation based heuristics also decreases when solving unicost problems or when facing problems with more rows than columns, which would make the problem size reduction techniques become almost useless. Recently, Yelbay et al. [22] have revisited the value of the dual information from LP or Lagrangian relaxation both in exact or heuristic algorithms, they express similar opinion in their article, see [22].

The weighting approach has been adopted in many heuristic algorithms for different problems, such as clause weighting in SAT [23, 24], and edge weighting in the minimum vertex covering problems (MVC) [25, 26]. However, there are no weighting related heuristics for the SCP so far. To our best knowledge, our algorithm is the first heuristic of this kind. The novelties of our algorithm are the propose of a local search framework that iteratively improves the candidate solution, which is realized by using two search operators to perturb the candidate solution with the upper bound restriction, and the successful hybridization of a weighting scheme, two tabu strategies and a timestamp method into its local search procedure.

## III. Our Algorithm

### A. Preliminary Terminologies

Before describing our algorithm, we fist introduce some necessary preliminaries related to our algorithm.

As the SCP can be presented as a $m \times n$ zero-one matrix $A$, where $m$ is the number of rows and n is the number of columns. Usually, $M$ is used to note as the set of rows, and $N$ is set of columns, thus, for $i \in M, j \in N$, we define

$$\theta(i) = \{j \in N | a_{ij} = 1\} \tag{4}$$

$$\delta(j) = \{i \in M | a_{ij} = 1\} \tag{5}$$

Apparently, the definition of $\theta(i)$ indicates the set of columns which are able to cover $i$, and $\delta(j)$ is the set of rows covered by $j$.

A candidate solution is denoted by $C$, $C \subseteq N$. During the search process, we say row $i$ is covered $\iff \exists j \in C$ and $i \in \delta(j)$.

For a column $j \in N$, an attribute *score* is defined and calculated as Equation 6.

$$j.score = \begin{cases} \sum\limits_{\substack{i \in \delta(j) \\ \sigma(C,i)=0}} i.weight & \text{if } j \notin C \\ -\sum\limits_{\substack{i \in \delta(j) \\ \sigma(C,i)=1}} i.weight & \text{if } j \in C \end{cases} \tag{6}$$

In Equation 6, $i.weight$ is the weight of row $i$, and $\sigma(C, i) = C \cap \theta(i)$, which represents the number of columns in $C$ covering row $i$.

The meaning of equation 6 is that when a column $j \notin C$, its $score$ is the sum of all the weights of rows it is able to cover and are still not covered by $C$. If a column $j \in C$, it means the negation of the sum of weights of rows $j$ covers and are only covered by this only column in $C$. It can be seen from Equation 6, if we move $j$ in or out from $C$, the $score$ of $j$ should be negated.

We also define neighborhood relations of columns, for $j1, j2 \in N$, and $j1 \neq j2$, if there is at least one row $j1, j2$ both can cover, we call $j1$ and $j2$ are mutually neighbors. The set $neighbor(j)$ holds all the neighbors of $j$, defined as

$$neighbor(j) = \{d \in N | d \neq j \wedge \delta(d) \cap \delta(j) \neq \emptyset\} \ j = 1...n \tag{7}$$

Each column has an additional Boolean attribute named $eligible$, which is used to prevent a column from adding back to $C$ when its $eligible$ value is false.

During its process, our algorithm maintains several variables. The best solution discovered so far is always stored as *bestSol* and its total cost is kept as *UB*= $\sum_{j \in bestSol}$*j.cost*. Additionally, the set of uncovered rows is maintained in a variable $L$.

### B. Algorithm Description

Our algorithm follows the general local search procedure. At first, an initial candidate solution $C$ is constructed greedily, and then a local search improvement is conducted by a perturbing method to improve the initial solution $C$.

Before constructing the initial solution, a preprocessing step is necessary when there are rows which are only covered by one column, and in this situation, such columns must be selected into the solution and the rows covered by them can be eliminated from the problem. This preprocessing has a time complexity of $\mathcal{O}(m)$. In other words, even in cases where it cannot reduce the problem size, its required runtime is negligible compared to the rest of the algorithm. If each row is certainly covered by two or more columns in the problem, preprocessing is unnecessary.

Let *cost(C)* denotes the cost of the candidate solution, which is *cost(C)*= $\sum_{\forall j \in C}$*j.cost*. The *UB* is initially set as *UB = cost(C)*. Therefore, if there are any better solutions, they must have costs less than *UB*. As we always maintain *UB* as the cost of the best solution we have found, then the local search improvement can also be regarded as to solve a series of new problems: given the original problem and an integer number *UB*, find a feasible solution whose cost is smaller than *UB* but still be able to cover all the rows in $M$.

The candidate solution becomes infeasible when it cannot cover all rows in $M$. Our algorithm repeatedly perturbs infeasible solutions with smaller cost than *UB*. Thus, once the initial candidate solution has constructed, at first, one more columns are removed from $C$ until $C$ becomes an infeasible solution under *UB*. In this process, if better solutions are found, the *UB* and stored *bestSol* should be updated.

The weighting scheme is applied each iteration when the candidate solution becomes infeasible. With this scheme, the weights of uncovered rows are increased by 1, thus making those "hard to cover" rows have a better chance to be covered by the new $C$ in the following iterations.

Based on the explanations above, we outline our heuristic algorithm as Algorithm 1.

**Algorithm 1:** Our Local Search Heuristic Algorithm

**Input:** A $m \times n$ zero-one matrix to represent a SCP instance, each column is associated with a cost
**Output:** A set of columns that ensure every row is covered and with the minimal total cost

1 preprocessing to eliminate redundant rows and columns;
2 initiate all rows have weight of 1;
3 initiate all columns have $timestamp$ of 1, $eligible$ of $true$;
4 calculate $score$ of each column accordingly;
5 initiate the set of uncovered rows $L \leftarrow M$;
6 construct a $C$ as an initial solution greedily until $L$ becomes empty;
7 *UB*$\leftarrow cost(C)$;
8 *bestSol*$\leftarrow C$;
9 *iteration*$\leftarrow 1$;
10 **while** *stop criterion not satisfied* **do**
11     **while** $L = \emptyset$ **do**
12         *UB*$\leftarrow cost(C)$;
13         *bestSol*$\leftarrow C$;
14         select $j \in C$ with the highest $score/cost$;
15         $remove(j)$;
16     **end**
17     select $j \in C \wedge j \notin tabu\_list$ with highest $j.score/j.cost$ and oldest on tie;
18     $remove(j)$;
19     *j.timestamp* $\leftarrow iteration$;
20     clear $tabu\_list$;
21     **while** $L$ *is not empty* **do**
22         $r \leftarrow rand(L)$;
23         select $d \in \{d1 \in \theta(r) | d1.eligible = true\}$ with the highest $j.score/j.cost$ and oldest on tie;
24         **if** $cost(C) + d.cost \geq$ *UB* **then** *break*;
25         ;
26         $add(d)$;
27         **forall** $i$ *of* $L$ **do**
28             *i.weight* $\leftarrow$ *i.weight* $+ 1$;
29         **end**
30         put $d$ to $tabu\_list$;
31         *d.timestamp* $\leftarrow iteration$;
32     **end**
33     $iteration \leftarrow iteration + 1$;
34 **end**
35 **return** *bestSol*;

As the description of Algorithm 1, after necessary preprocessing and initialization, a candidate solution is constructed by a greedy procedure until $L$ becomes empty. Then, from Line 10 to Line 33, in each iteration, $C$ becomes an infeasible solution by using the $remove$ operator consecutively delete the columns with the highest negative *score/cost*.

Right after $C$ becomes infeasible, one column which is not in the $tabu\_list$ is chosen to be removed again, and then a row is randomly selected from the uncovered row set $L$, and the best column with highest *score/cost* and *eligible* $= true$ is chosen to be added to $C$. While enforcing the upper bound restriction in Line 24, more than one columns are permitted to be added to $C$ each iteration until L becomes empty. The tabu list keeps track the columns which are added in the last iteration and is cleared before adding columns into C.

The *eligible* $= true$ restriction in Line 23 is a another kind of tabu. We do not want a column which has been removed from $C$ to be added back again if none of its neighbors' state have changed, so we set $j$.*eligible* $= false$ if $j$ leaves from $C$, which means $j$ is not eligible to be added to $C$. When the state of one of $j$'s neighbors changes (due to removal or addition), *j.eligible* changes back to $true$ again.

The timestamp method used in Algorithm 1 is used to break ties. It makes sure that those columns, which have not been selected for a longer time are preferred, for they have smaller timestamps.

The $add$ and $remove$ operators in our algorithm are quite simple, whenever a column $j$ is added into or removed from $C$, $j.score$ is negated and its neighbors' *score* value are updated as of Equation 6. Then its neighbors' $eligible$ are set to $true$. Only when a column is just removed from $C$, then its *eligible* is set to *false*.

Our algorithm only needs a parameter for termination. It may be computational budget limit such as a maximum search iteration number or an indicated maximum runtime.

## IV. Experimental Results

To show the effectiveness and efficiency of the proposed heuristic algorithm, we test it both on unicost SCP instances and non-unicost SCP instances. Then compare our algorithm with two excellent solvers in literature, which are 3-FNLS and Meta-RaPS.

Our algorithm is programmed with C++, compiled with g++ with -O2 option, run on a Intel i3-3220 3.3GHz CPU with 4GB RAM machine under Linux system. Time is measured in CPU seconds in all experiments. Because of the randomness of our algorithm, usually 10 trials are performed for each instances with different seeds. More exactly, we use 10 consecutive integer numbers as random seeds.

### A. Test on non-unicost instances from the OR-Library

The OR-Library is a collection of test data sets for a variety of Operations Research (OR) problems, which was originally described by J.E.Beasley [21].

There are totally 70 randomly generated non-unicost instances in the OR-Library, which are divided into 12 problem sets. One such set of instance is known to be very simple and a greedy procedure can easily obtain its optima. Thus we only test our algorithm on the remaining 65 harder instances.

Table I contains the details of the 65 hard random problems, in which *Density* is the percentage of non-zero entries in the matrix. The *Range of cost* column gives the cost range

TABLE I

THE PROBLEM DETAILS

| Problem set | m | n | Range of cost | Density(%) | Number of problems |
|---|---|---|---|---|---|
| 4 | 200 | 1000 | 1–100 | 2 | 10 |
| 5 | 200 | 2000 | 1–100 | 2 | 10 |
| 6 | 200 | 1000 | 1–100 | 5 | 5 |
| A | 300 | 3000 | 1–100 | 2 | 5 |
| B | 300 | 3000 | 1–100 | 5 | 5 |
| C | 400 | 4000 | 1–100 | 2 | 5 |
| D | 400 | 4000 | 1–100 | 5 | 5 |
| NRE | 500 | 5000 | 1–100 | 10 | 5 |
| NRF | 500 | 5000 | 1–100 | 20 | 5 |
| NRG | 1000 | 10000 | 1–100 | 2 | 5 |
| NRH | 1000 | 10000 | 1–100 | 5 | 5 |

TABLE II

SOLUTIONS BY OUR ALGORITHM ON INSTANCES FROM 4-6, A-D

| Inst | opt | best | #best | Avg Time | Inst | opt | best | #best | Avg Time |
|---|---|---|---|---|---|---|---|---|---|
| 4.1 | 429 | 429 | 10 | 0.01 | 6.4 | 131 | 131 | 10 | 0.01 |
| 4.2 | 512 | 512 | 10 | 0.02 | 6.5 | 161 | 161 | 10 | 0.01 |
| 4.3 | 516 | 516 | 10 | 0.00 | A1 | 253 | 253 | 10 | 0.23 |
| 4.4 | 494 | 494 | 10 | 0.07 | A2 | 252 | 252 | 10 | 0.07 |
| 4.5 | 512 | 512 | 10 | 0.01 | A3 | 232 | 232 | 10 | 0.03 |
| 4.6 | 560 | 560 | 10 | 0.01 | A4 | 234 | 234 | 10 | 0.03 |
| 4.7 | 430 | 430 | 10 | 0.06 | A5 | 236 | 236 | 10 | 0.04 |
| 4.8 | 492 | 492 | 10 | 0.01 | B1 | 69 | 69 | 10 | 0.01 |
| 4.9 | 641 | 641 | 10 | 0.01 | B2 | 76 | 76 | 10 | 0.01 |
| 4.10 | 514 | 514 | 10 | 0.02 | B3 | 80 | 80 | 10 | 0.01 |
| 5.1 | 253 | 253 | 10 | 0.01 | B4 | 79 | 79 | 10 | 0.02 |
| 5.2 | 302 | 302 | 10 | 0.01 | B5 | 72 | 72 | 10 | 0.01 |
| 5.3 | 226 | 226 | 10 | 0.01 | C1 | 227 | 227 | 10 | 0.09 |
| 5.4 | 242 | 242 | 10 | 0.01 | C2 | 219 | 219 | 10 | 0.07 |
| 5.5 | 211 | 211 | 10 | 0.01 | C3 | 243 | 243 | 10 | 0.08 |
| 5.6 | 213 | 213 | 10 | 0.01 | C4 | 219 | 219 | 10 | 0.02 |
| 5.7 | 293 | 293 | 10 | 0.01 | C5 | 215 | 215 | 10 | 0.06 |
| 5.8 | 288 | 288 | 10 | 0.01 | D1 | 60 | 60 | 10 | 0.04 |
| 5.9 | 279 | 279 | 10 | 0.01 | D2 | 66 | 66 | 10 | 0.03 |
| 5.10 | 265 | 265 | 10 | 0.01 | D3 | 72 | 72 | 10 | 0.05 |
| 6.1 | 138 | 138 | 10 | 0.01 | D4 | 62 | 62 | 10 | 0.03 |
| 6.2 | 146 | 146 | 10 | 0.01 | D5 | 61 | 61 | 10 | 0.02 |
| 6.3 | 145 | 145 | 10 | 0.01 | | | | | |

in corresponding problem set. Problem sets 4 and 5 have ten instances, the rest all have five instances. Problems from set 4 to 6 and A to D are those whose optima are known. For the large-sized SCPs in sets NRE to NRH are unknown.

Table II contains our experimental results of these problems whose optima are known. The second column *opt* gives the optimum of the corresponding instances. For each instance, we present the best solution (*best*) obtained during the ten trials, as well as the number *#best* and average runtime (*Avg Time*) of the trials discovering that best solution.

From Table II we can see that for instances from 4 to 6 and A to D, our algorithm has found the optima for every instance on every trial using almost negligible time.

Table III contains the comparison of our algorithm and 3-FNLS. The source code of 3-FNLS are provided by the author Yagiura [20]. It is written in C++, too, so we can compare it with our algorithm directly. 3-FNLS is also compiled with the same options and run on the same machine as our method under Linux system. As the explanation in [20], the parameter settings of 3-FNLS are $\alpha = 3, minitr\_ls = 100, max\_rcost = 0.1$ for all the instances tested in this paper.

Table III contains the comparison on random instances from NRE to NRH. Time limits for both algorithms are set to 10 seconds. The first column of Table III is the problem instance and the second column 'BKS' is the best known solution for the corresponding instance. From Table III, we can see that our algorithm is comparable with 3-FNLS on most instances. For one instance, NRE2, our algorithm performs better than 3-FNLS, because it can find the BKS all runs with less runtime. But for instance NRH1, our algorithm is slightly inferior to 3-FNLS. From the summarization at the bottom of Table III, we can conclude that these two algorithms are comparable on these instances.

TABLE III

COMPARISON WITH 3-FNLS ON INSTANCES FROM NRE - NRH

| Inst. | BKS | 3-FNLS | | | Our algorithm | | |
|---|---|---|---|---|---|---|---|
| | | best | #best | Avg Time | best | #best | Avg Time |
| NRE1 | 29 | 29 | 10 | 0.20 | 29 | 10 | 0.03 |
| NRE2 | 30 | 30 | 3 | 5.46 | 30 | 10 | 0.62 |
| NRE3 | 27 | 27 | 10 | 0.22 | 27 | 10 | 0.10 |
| NRE4 | 28 | 28 | 10 | 0.20 | 28 | 10 | 0.05 |
| NRE5 | 28 | 28 | 10 | 0.20 | 28 | 10 | 0.04 |
| NRF1 | 14 | 14 | 10 | 0.26 | 14 | 10 | 0.10 |
| NRF2 | 15 | 15 | 10 | 0.20 | 15 | 10 | 0.06 |
| NRF3 | 14 | 14 | 10 | 0.21 | 14 | 10 | 0.11 |
| NRF4 | 14 | 14 | 10 | 0.28 | 14 | 10 | 0.14 |
| NRF5 | 13 | 13 | 10 | 0.23 | 13 | 10 | 1.49 |
| NRG1 | 176 | 176 | 10 | 0.56 | 176 | 10 | 0.14 |
| NRG2 | 154 | 154 | 10 | 0.70 | 154 | 10 | 0.96 |
| NRG3 | 166 | 166 | 8 | 2.56 | 166 | 7 | 2.98 |
| NRG4 | 168 | 168 | 10 | 1.24 | 168 | 10 | 3.46 |
| NRG5 | 168 | 168 | 10 | 0.88 | 168 | 10 | 1.22 |
| NRH1 | 63 | 63 | 10 | 1.13 | 63 | 4 | 3.51 |
| NRH2 | 63 | 63 | 10 | 1.26 | 63 | 10 | 0.89 |
| NRH3 | 59 | 59 | 10 | 0.55 | 59 | 10 | 1.18 |
| NRH4 | 58 | 58 | 10 | 0.62 | 58 | 10 | 0.72 |
| NRH5 | 55 | 55 | 10 | 0.45 | 58 | 10 | 0.32 |
| SUM | 1342 | 1342 | 191 | 17.41 | 1342 | 191 | 18.12 |

TABLE IV

RESULTS OF META-RAPS ON INSTANCES FROM NRE–NRH

| Instance | best | #best | mean | best time | total time |
|---|---|---|---|---|---|
| NRE1 | 29 | 80 | 29.58 | 0.01 | 141.83 |
| NRE2 | 30 | 9 | 33.40 | 6.09 | 95.61 |
| NRE3 | 27 | 27 | 28.82 | 1.61 | 120.01 |
| NRE4 | 28 | 35 | 29.89 | 2.50 | 125.09 |
| NRE5 | 28 | 71 | 29.63 | 1.12 | 108.33 |
| NRF1 | 14 | 50 | 15.17 | 0.93 | 194.70 |
| NRF2 | 15 | 74 | 15.87 | 0.09 | 213.30 |
| NRF3 | 14 | 24 | 15.50 | 0.54 | 21.53 |
| NRF4 | 14 | 39 | 15.46 | 3.46 | 165.32 |
| NRF5 | 13 | 2 | 14.49 | 16.18 | 222.34 |
| NRG1 | 176 | 4 | 180.02 | 68.12 | 232.95 |
| NRG2 | **155** | 11 | 157.93 | 10.31 | 212.96 |
| NRG3 | 166 | 1 | 171.41 | 95.99 | 224.41 |
| NRG4 | 168 | 1 | 173.38 | 144.83 | 223.38 |
| NRG5 | 168 | 10 | 172.09 | 1.56 | 221.57 |
| NRH1 | 63 | 1 | 66.17 | 251.96 | 387.68 |
| NRH2 | 63 | 1 | 67.38 | 32.14 | 354.68 |
| NRH3 | 59 | 1 | 63.35 | 58.73 | 356.84 |
| NRH4 | 58 | 3 | 60.87 | 181.61 | 370.53 |
| NRH5 | 55 | 14 | 57.64 | 86.60 | 358.38 |

We also report the results of Meta-RaPS on instances from NRE to NRH. The source code of Meta-RaPS is provided by the author Lan [16]. It is written in Borland C++, thus we can only run Meta-RaPS on Windows system. For this purpose, we installed Windows on the same machine from above and then run Meta-RaPS with the default parameter settings, as below:

TABLE V
COMPARISON WITH 3-FNLS ON STS INSTANCES

| Inst. | BKS | 3-FNLS | | | Our algorithm | | |
|---|---|---|---|---|---|---|---|
| | | best | #best | Avg Time | best | #best | Avg Time |
| $STS_{135}$ | 103 | 103 | 10 | 134.64 | 103 | 10 | 3.97 |
| $STS_{243}$ | 198 | 198 | 10 | 48.66 | 198 | 10 | 0.07 |
| $STS_{405}$ | 335 | 336 | 10 | 137.54 | **335** | 3 | 436.55 |
| $STS_{729}$ | 617 | 617 | 2 | 182.25 | 617 | 10 | 20.03 |

TABLE VI
RESULTS OF META-RAPS ON STS INSTANCES

| Inst. | BKS | best | #best | mean | best time | total time |
|---|---|---|---|---|---|---|
| $STS_{135}$ | 103 | 104 | 4 | 105.23 | 3.98 | 15.63 |
| $STS_{243}$ | 198 | 203 | 6 | 204.64 | 6.04 | 49.91 |
| $STS_{405}$ | 335 | 343 | 1 | 345.88 | 112.74 | 139.38 |
| $STS_{729}$ | 617 | 650 | 4 | 651.76 | 61.59 | 505.95 |

*max iterations: 100*,
*priority: 5%*,
*restriction: 20%*,
*improvement: 15%*,
*penalty:2%*.

Table IV contains the results of Meta-RaPS. Column *best* holds the best solution obtained within the 100 iterations, *#best* is the number iterations until that the best is hit, *mean* is the mean value of solutions among the 100 iterations, *best time* is the shortest time cost to find the best among the 100 iterations, and *total time* is the total time cost of the 100 iterations.

As the stopping criteria are different, we do not give direct comparison between our algorithm and Meta-RaPS, but from Table IV, we can see that the best times of Meta-RaPS are generally larger than the average times of our algorithm. For one instance NRG2, Meta-RaPS fails to obtain the BKS 154.

### B. Test on the unicost STS instances

In order to show the effectiveness of our algorithm for unicost problems, we also test our algorithm and 3-FNLS on the STS instances, which are from Steiner triple systems [27]. The STS instances are unicost problems with special structure and are considered difficult to solve. Another obvious feature of the STS problem is that the number of rows is much bigger than the number of columns, as below:

$STS_{135}$ n: 135, m: 3015, Density: 2.2%
$STS_{243}$ n: 243, m: 9801, Density: 1.2%
$STS_{405}$ n: 405, m: 27270, Density: 0.7%
$STS_{729}$ n: 729, m: 88452, Density: 0.4%

We can see that different with the OR-Library instances, the number of rows (m) in STS instances are always more than one magnitude than the number of columns (n). Generally, the cost of all columns in unicost problems are regarded as 1.

Table V contains the BKS of the STS instances. Currently, the BKS of $STS_{135}$ and $STS_{243}$ has been proven optimality, see [28]. Recently, the BKS of $STS_{405}$ is improved by [29] from 336 to 335 using a biased random-key genetic algorithm. However, the biased random-key genetic algorithm [29] is dedicated for the STS problems, thus we do not compare it with our algorithm here.

Time limits are set to 1000 seconds for each run on these STS instances for both the two algorithms. Table V contains the results of our algorithm and 3-FNLS. From Table V, it is easy to notice that our algorithm is better than 3-FNLS, for it is able to achieve the same best solution with less runtimes. For one instance $STS_{405}$, our algorithm has achieved the state-of-the-art result, whereas 3-FNLS can only find a solution of 336 within 1000 seconds. Moreover, we emphasize the solution of 336 for $STS_{405}$ can be easily achieved by our algorithm in all the 10 trials with an average time of 7.27s.

We further compare our algorithm with Meta-RaPS, which also work effectively both on the unicost and non-unicost problems. The parameters of Meta-RaPS are still set as default, which is indicated by the program.

Table VI contains the results obtained by Meta-RaPS. The results of Meta-RaPS are presented by the best objective value, the number of iterations until that best is hit, the mean result of the 100 iterations, the best solution time, and the total time of the 100 iteration. We can see that Meta-RaPS did not achieve the best for all these four STS instances. Combined with the results in Table IV, we can conclude that our algorithm is more effective than Meta-RaPS on these STS instances.

## V. CONCLUSIONS

In this paper, we have introduced a local search heuristic algorithm based on weighting. The effectiveness of our algorithm has been tested on the instances from OR-Library and the STS. The experimental results show that our new algorithm is comparable with 3-FNLS on instances from OR-Library, for it is able to find all the optima or BKS in very short times. Besides, our algorithm is also very effective on the unicost STS instances, which are generally regarded hard. Therefore, we believe our algorithm is very competitive. Furthermore, our algorithm only needs a parameter for termination, thus we believe our algorithm is very practical and worth existing for further study.

As the experiment has shown, most of the random instances from the OR-Library are not challenging any more. In the future, we will try collect more SCP instances, both non-unicost and unicost, then test them with our algorithm. Thus, we can have further study on the relationship between the effectiveness of our algorithm and the instance features.

## VI. ACKNOWLEDGMENTS

The authors are very grateful to Mutsunori Yagiura and Guanghui Lan for providing the source code of 3FNLS and Meta-RaPS, respectively.

We also want to note that for Meta-RaPS, if with parameter tuning, may produce better results than that in this paper, but this is not the scope of our work, thus here we only use the default parameter settings in our experiments.

This research work was partially supported by an EPSRC grant (No. EP/I010297/1), the Fundamental Research Funds for the Central Universities (WK0110000023), the

National Natural Science Foundation of China under Grants 61150110488, the Special Financial Grant 201104329 from the China Postdoctoral Science Foundation, and the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011Y1GB01.

## References


[1] C. Gao, T. Weise, and J. Li, "A weighting-based local search heuristic algorithm for the set covering problem," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'14), Proceedings of the 2014 World Congress on Computational Intelligence (WCCI'14)*. Beijing, China: Beijing International Convention Center (BICC): Los Alamitos, CA, USA: IEEE Computer Society Press, July 6–11, 2014, pp. 826–831, doi:10.1109/CEC.2014.6900355.

[2] M. R. Gary and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness," 1979.

[3] A. Caprara, M. Fischetti, P. Toth, D. Vigo, and P. L. Guida, "Algorithms for railway crew management," *Mathematical Programming*, vol. 79, no. 1-3, pp. 125–141, 1997.

[4] J. Bautista and J. Pereira, "Modeling the problem of locating collection areas for urban waste management. an application to the metropolitan area of barcelona," *Omega*, vol. 34, no. 6, pp. 617–629, 2006.

[5] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. San Francisco, USA: Morgan Kaufmann, 2005.

[6] J. E. Beasley, "An algorithm for set covering problem," *European Journal of Operational Research*, vol. 31, no. 1, pp. 85–93, 1987.

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

[8] E. Balas and M. C. Carrera, "A dynamic subgradient-based branch-and-bound procedure for set covering," *Operations Research*, vol. 44, no. 6, pp. 875–890, 1996.

[9] J. E. Beasley and K. Jörnsten, "Enhancing an algorithm for set covering problems," *European Journal of Operational Research*, vol. 58, no. 2, pp. 293–300, 1992.

[10] A. Caprara, P. Toth, and M. Fischetti, "Algorithms for the set covering problem," *Annals of Operations Research*, vol. 98, no. 1-4, pp. 353–371, 2000.

[11] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[12] F. J. Vasko, "An efficient heuristic for large set covering problems," *Naval Research Logistics Quarterly*, vol. 31, no. 1, pp. 163–171, 1984.

[13] T. A. Feo and M. G. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations research letters*, vol. 8, no. 2, pp. 67–71, 1989.

[14] 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.

[15] L. W. Jacobs and M. J. Brusco, "Note: A local-search heuristic for large set-covering problems," *Naval Research Logistics*, vol. 42, no. 7, pp. 1129–1140, 1995.

[16] G. Lan, G. W. DePuy, and G. E. Whitehouse, "An effective and simple heuristic for the set covering problem," *European journal of operational research*, vol. 176, no. 3, pp. 1387–1403, 2007.

[17] J. Beasley, "A lagrangian heuristic for set-covering problems," *Naval Research Logistics (NRL)*, vol. 37, no. 1, pp. 151–164, 1990.

[18] A. Caprara, M. Fischetti, and P. Toth, "A heuristic method for the set covering problem," *Operations research*, vol. 47, no. 5, pp. 730–743, 1999.

[19] S. Ceria, P. Nobili, and A. Sassano, "A lagrangian-based heuristic for large-scale set covering problems," *Mathematical Programming*, vol. 81, no. 2, pp. 215–228, 1998.

[20] 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.

[21] J. E. Beasley, "Or-library: distributing test problems by electronic mail," *Journal of the Operational Research Society*, pp. 1069–1072, 1990.

[22] B. Yelbay, S. Birbil, and K. Bulbul, "The set covering problem revisited: an empirical study of the value of dual information," Optimization Online, 2012.

[23] J. Thornton, "Clause weighting local search for sat," *Journal of Automated Reasoning*, vol. 35, no. 1-3, pp. 97–142, 2005.

[24] J. Thornton, D. N. Pham, S. Bain, and V. Ferreira Jr, "Additive versus multiplicative clause weighting for sat," in *AAAI*, vol. 4, 2004, pp. 191–196.

[25] S. Richter, M. Helmert, and C. Gretton, "A stochastic local search approach to vertex cover," in *KI 2007: Advances in Artificial Intelligence*. Springer, 2007, pp. 412–426.

[26] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artificial Intelligence*, vol. 175, no. 9, pp. 1672–1696, 2011.

[27] D. Fulkerson, G. Nemhauser, and L. Trotter, "Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of steiner triple systems," in *Approaches to Integer Programming*, ser. Mathematical Programming Studies, M. Balinski, Ed. Springer Berlin Heidelberg, 1974, vol. 2, pp. 72–81. [Online]. Available: http://dx.doi.org/10.1007/BFb0120689

[28] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio, "Solving large steiner triple covering problems," *Operations Research Letters*, vol. 39, no. 2, pp. 127–131, 2011.

[29] M. G. Resende, R. F. Toso, J. F. Gonçalves, and R. M.

Silva, "A biased random-key genetic algorithm for the steiner triple covering problem," *Optimization Letters*, vol. 6, no. 4, pp. 605–619, 2012.

This is a preview version of the paper [1] (see below for the reference). Read the full piece in the proceedings.


```
@inproceedings{GWL2014AWBLSHAFTSCP,
  author    = {Chao Gao and Thomas Weise and Jinlong Li},
  title     = {A Weighting-Based Local Search Heuristic Algorithm
               for the Set Covering Problem},
  booktitle = {Proceedings of the IEEE Congress on Evolutionary
               Computation (CEC'14), Proceedings of the 2014 World
               Congress on Computational Intelligence (WCCI'14)},
  publisher = {Los Alamitos, CA, USA: IEEE Computer Society Press},
  address   = {Beijing, China: Beijing International Convention
               Center (BICC)},
  pages     = {826--831},
  year      = {2014},
  month     = jul # {~6--11, },
  doi       = {10.1109/CEC.2014.6900355},
},
```