An Efficient Local Search Heuristic with Row Weighting for the Unicost Set Covering Problem

Chao Gao[a], Xin Yao[a,b], Thomas Weise[a], Jinlong Li[a,*]

[a] *USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI), School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China.*
[b] *The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K.*

Abstract

The Set Covering Problem (SCP) is an $\mathcal{NP}$-hard optimization task encountered in many applications. We propose a new Row Weighting Local Search (RWLS) algorithm for solving the unicost variant of the SCP, i.e., USCPs where all costs of all sets are identical. RWLS is a hybrid algorithm that has three major components united in its local search framework: **(1)** a weighting scheme, which updates the weights of uncovered elements used to prevent convergence to local optima, **(2)** tabu strategies to avoid possible cycles during the search, and **(3)** a timestamp method to break ties when prioritizing sets. RWLS is evaluated on a large number of problem instances from the OR-Library and compared to other approaches. It is able to find all the best known solutions (BKS) and improves 14 of them, although requiring a higher computational effort on several instances. RWLS is especially effective on the combinatorial OR-Library instances and can improve the best known solution of the hardest instance CYC11 considerably. It is conceptually simple and has no instance-dependent parameters, which makes it a practical and easy-to-use USCP solver.

Keywords: Combinatorial Optimization; Unicost Set Covering Problem; Row Weighting Local Search; Hybrid Algorithms

*Corresponding author. Tel: +8618019557504
Email addresses: chao.gao.ustc@gmail.com (Chao Gao), x.yao@cs.bham.ac.uk (Xin Yao), tweise@ustc.edu.cn (Thomas Weise), jlli@ustc.edu.cn (Jinlong Li)

1. Introduction

The Set Covering Problem (SCP) is a combinatorial optimization problem with many applications, ranging from crew scheduling in railway and mass-transit companies to job assignment in manufacturing and service location [1, 2]. It can be described as follows: We are given a set of elements X, a set S of subsets s with $s \subseteq X$, $\forall s \in S$ and $\bigcup_{s \in S} s = X$, each subset in S is associated with a cost, the goal is to find a set $F \subseteq S$ whose union is X (which contains all elements from X) at the minimal total cost. If each set s has the same cost, the problem is referred to the unicost set covering problem (USCP). Although being a special case of SCP, the unicost version is generally considered to be harder to solve [3] and is the subject of this paper.

Formally, an SCP instance is usually defined as a $m \times n$ zero-one matrix $A = \{a_{ij}\}_{m \times n}$ where $a_{ij} = 1$ means column (set) j can cover row (element) i. The objective is to find a set of columns at the minimal cost to cover all the rows. If the problem is a USCP, the objective can also be viewed as to find the smallest set of columns to cover all the rows. A candidate solution C can then be represented as a subset of $N = \{1, \ldots, n\}$. Such a solution is feasible if and only if $\sum_{j \in C} a_{ij} \geq 1$, $\forall i \in M$ with $M = \{1, \ldots, m\}$ and the objective function subject to minimization is $|C|$.

In this paper, we propose the stochastic Row Weighting Local Search (RWLS) algorithm for solving USCPs. RWLS uses two search operators to perturb the candidate solution and hybridizes three major existing strategies into its local search procedure:

(1) A weighting scheme, which updates the weights of the uncovered rows, is applied in order to escape local optima.
(2) Different tabu strategies, which prevent possible cycles during the search.
(3) A timestamp method to break ties, which makes the sets that are not moved for longer time are preferred to be selected.

In our experiments, RWLS has improved 14 best known solutions in the literature for 87 USCP instances from the OR-Library [4] and shown excellent performance. It is especially effective on the problems in which the number of rows (elements) is much larger than the number of columns (sets). However, RWLS is also effective in other cases as well, for example, for the seven railway crew scheduling problems, with up to millions of columns and thousands of rows. Incorporated with problem size reduction, it outperforms CPLEX12.5 consistently and succeeds in finding good solutions to all seven instances when CPLEX12.5 failed on four larger instances. Overall, RWLS is simple, efficient, and only needs a single parameter to indicate the stopping criterion.

The rest of this paper is organized as follows. We first discuss related work in Section 2 and then give a detailed description of RWLS in Section 3. The experimental studies are presented in Section 4 and compared with several approaches from the related work. Conclusions and future work are finally given in Section 5.

2. Related Work

The SCP is NP-hard in the strong sense [5]. Many algorithms have been developed for solving the SCP in the past years. Exact approaches [6, 7, 8, 9] are mostly based on branch-and-bound or branch-and-cut. Caprara et al. [10] compared different exact algorithms and found that the best exact approach is CPLEX. However, although exact algorithms can guarantee the optimality of the found solutions, they usually require substantial computational efforts when facing large scale problems.

Therefore, large instances of SCP are typically tackled by heuristic algorithms. The simplest approximation algorithm for SCPs is the greedy algorithm [11]. It has been proven to have an approximation ratio of $\ln(k') + 1$, where k' is the size of the optimum solution. Later, several randomized greedy algorithms [12, 13] are proposed. They usually produce better results than the pure greedy one. A variety of other heuristic algorithms have also been proposed, including some general meta-heuristics, such as Genetic Algorithms [14], Simulated Annealing [15] and Lagrangian Relaxation-based heuristics [16, 17, 18, 19], among which the heuristic methods by Ceria et al [17], Caprara et al. [18] and Yagiura et al. [19] are able to achieve remarkable results on the very large-scale instances by exploiting their specific features. We highlight the 3-flip neighborhood local search (3FNLS) method by Yagirua et al. [19], for it successfully combines 3-flip local search, adaptive penalty weights control techniques and Lagrangian Relaxation, and has the best performance on the very large-scale railway crew scheduling problems. For a good survey of relaxation-based heuristics for the SCP, see [20].

Lan et al. [21] noticed that the cost information plays an important role in the Genetic algorithm [14], Simulated Annealing [15] and the Lagrangian-based heuristic [16], which makes them being not recommended for USCPs. They therefore proposed the Meta-RaPS approach that works effectively both for unicost and non-unicost SCPs. Recently, Yelbay et al. also gave a detailed explanation of the usefulness and limitations of the dual information from Lagrangian Relaxation or Linear Programming (LP) Relaxation and they further pointed out that the unicost problems may be more challenging than the non-unicost problems [3].

There are heuristics dedicated to specifically solving USCPs. Grossman and Wool [22] compared nine heuristics, including several greedy variants and a neural network algorithm designed by the authors. In their report, the randomized greedy variant R-Gr has the best performance on an extensive set of instances from the OR-Library [4]. A newer GRASP algorithm incorporating a local improvement procedure from (Satisfiability) SAT solving has shown better results than R-Gr [23].

The Electromagnetism Meta-heuristic (EM) proposed by Naji-Azimi et al. [24] creates the initial population by generating a pool of solutions, and then a fixed number of local search and movement iterations are applied based on the "electromagnetism" theory. In order to further escape from the local optima, mutation is also adopted. The computational results show that EM performs much better than GRASP, but in comparison with Meta-RaPS for the combinatorial problem set, for 3 instances the solution qualities obtained by EM are inferior.

Stochastic local search is a popular approach for solving hard combinatorial problems [25]. Musliu [26] proposed a local search algorithm for the USCP using a simple fitness function, which is the number of uncovered elements plus the cardinality of the candidate solution. New candidate solutions are created by adding and removing sets from the current one. To avoid cycles during the local improvement phase, a tabu mechanism is used. According to our investigation, Musliu's algorithm is able to find the best known solutions on 80 instances from the OR-Library [4] as unicost problems and 5 instances from Steiner triple systems [27].

The weighting approach has been adopted in several heuristics for different problems, such as clause weighting in SAT [28, 29, 30, 31]. Our algorithm is the first pure stochastic local search heuristic dedicated for the USCP. Because in USCP all sets have the same cost, the task can also be seen as to minimize the number of sets in a solution. RWLS utilizes this property of USCP, and uses a general search framework to iteratively reduce the size of the candidate solution.

3. Row Weighting Local Search

3.1. Notations and Definitions

Before presenting our algorithm in detail, we give some necessary notations and definitions. As mentioned, the USCP can be presented as a $m \times n$ zero-one matrix. M and N indicates the set of rows and columns, respectively. We define J_i, the set of columns which are able to cover row i, and

I_j, the set of rows covered by column j, as follows:

$$J_i = \{j \in N | a_{ij} = 1\} \quad i = 1, \ldots, m, \tag{1}$$
$$I_j = \{i \in M | a_{ij} = 1\} \quad j = 1, \ldots, n. \tag{2}$$

A candidate solution C is noted as a subset of columns: $C \subseteq N$. For all i in M, we say that row i is covered if and only if there exists a j in C that satisfies $i \in I_j$.

For all j in N, an attribute denoted $j.score$, which is later used to prioritize the columns for covering, is defined and calculated according to Equation (3).

$$j.score = \begin{cases} \sum\limits_{\substack{i \in I_j \\ \sigma(C,i)=0}} i.weight & \text{if } j \notin C \\ -\sum\limits_{\substack{i \in I_j \\ \sigma(C,i)=1}} i.weight & \text{if } j \in C \end{cases} \tag{3}$$

In Equation (3), $i.weight$ is the weight of row i and $\sigma(C, i) = C \cap J_i$ represents the number of columns in C covering row i. When a column $j \notin C$, $j.score$ is the sum of all the weights of rows that j is able to cover and are still not covered by C. If a column $j \in C$, $j.score$ is the negation of the sum of the weights of rows which are only covered by j in C. It can be seen that, if we move j into or out of C, the *score* of j is negated.

Each column has a timestamp associated with it, which gets updated whenever it is moved in or out. It is used to break ties when more than two columns have the same *score*.

We also define relations of columns. For all j_1, j_2 in N, and $j_1 \neq j_2$, if $\exists i \in M, i \in I_{j_1} \cap I_{j_2}$, we call j_1 and j_2 *neighbors*. The notation $neighbor(j)$ contains all the neighbors of j, defined as

$$neighbor(j) = \{d \in N | d \neq j \wedge I_d \cap I_j \neq \emptyset\}, \quad j = 1, \ldots, n \tag{4}$$

Each column has a Boolean attribute named *canAddToSolution*, which is used to implement one of the two tabu strategies in RWLS. A column j can only be added to C if $j.canAddToSolutin$ is true. The set of uncovered rows are maintained in a variable L in RWLS.

3.2. The RWLS Algorithm

RWLS is a USCP solver. It tries to find the smallest set of columns that covers all the rows in M. For this purpose, we adopt a two phase search procedure. In the first phase, an initial solution C is constructed in a greedy manner. Then, a local search improvement is conducted with the weighting scheme. The overall procedure of RWLS is described as Algorithm 1.

A preprocessing step is necessary when there are rows which are only covered by one column. Such columns must be selected into the solution and the rows they cover can be removed from the problem. After reading the problem instance, we examine the number of columns covering each row, and for rows that are only covered by one single column, the corresponding column is marked not to be removed from the candidate solution permanently. This can be done with a time complexity at most $\mathcal{O}(m)$. If every row is covered by two or more columns in the problem, preprocessing is unnecessary.

Algorithm 1 The RWLS algorithm

```
1: function RWLS( )
2:     read problem instance
3:     set stopping criteria
4:     preprocessing if necessary
5:     INIT( )
6:     LOCALSEARCH( )
7: end function
```

3.2.1. Initialization

Algorithm 2 describes the initialization phase, which builds two sets C and L, representing the initial solution and the set of uncovered rows, respectively. Every row is initialized to have a weight of 1 and each column j has a *canAddToSolution* of *true*, a *timestamp* of 1 and a *score* computed according to Equation (3), i.e., the number $|I_j|$ of rows it can cover. C is then constructed greedily in a loop until L becomes empty. It is then used as the candidate solution in the subsequent local improvement phase.

The $ADD(j)$ is a simple operator, in which the scores of j and $neighbor(j)$ are updated accordingly, and the *canAddSolution*s of $neighbor(j)$ are set to *true*. After calling $ADD(j)$, j is moved to C, and the newly covered rows are removed from L.

3.2.2. Local Search

Let the size of the initial solution C be $k = |C|$. If there is any better solution, it must have a size less than k. If we always maintain k as the size of the best solution we have encountered so far, 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 k, find a $k-1$ size solution which is able to cover all the rows in M.

Therefore, we take the initial solution C as the candidate solution into the local search improvement phase defined as Algorithm 3. Here, the *REMOVE* function is first called, which removes a column from C. C then becomes a

Algorithm 2 Initialization

```
1: function INIT(C candidate solution)
2:     for j ∈ N do
3:         j.score ← 0
4:         j.timestamp ← 1
5:         j.canAddToSolution ← true
6:     end for
7:     L ← ∅
8:     for i ∈ M do
9:         i.weight ← 1
10:        add i into L
11:        for d ∈ J_i do
12:            d.score ← d.score + i.weight
13:        end for
14:    end for
15:    C ← ∅
16:    while L ≠ ∅ do
17:        j ← rand({d ∈ N\C ∧ d.score = max{d1.score|d1 ∈ N\C}})
18:        remove newly covered rows from L
19:        ADD(j)
20:    end while
21: end function
```

partial solution with $k-1$ columns. However C may have redundant columns, and if one of such columns is removed, we get an even better solution and the stored best solution and the variable k will be updated. We continue to remove columns until C becomes a partial solution which cannot cover all rows in M. As a partial solution of size $k-1$ has been obtained, a pair of operations (ADD and $REMOVE$) are used to perturb C. The weighting scheme is also applied, which means that the weights of uncovered rows are increased. The weighing scheme improves the chance of uncovered rows being covered in the following iteration for the "hard-to-cover" rows.

As described in Algorithm 3, in each iteration, C becomes a partial solution of size $k-1$. The $REMOVE$ operator deletes the columns with the highest negative *score* in C. RWLS keeps track of two previously added columns in the tabu list, i.e., a FIFO queue of size two, to prevent them from being removed again immediately. We found that even with a tabu list length of one, good results can be achieved, but with length two the algorithm tends to proceed faster. After the removal, a row is randomly selected from the uncovered row set L and the column with the highest *score* and $canAddToSolution = true$ is chosen to be added to C. The $REMOVE(j)$ operator is symmetry with $ADD(j)$, in which the scores of j and $neighbor(j)$ are updated accordingly, and $j.canAddToSolution$ is set to *false*, whereas its neighbors' $canAddToSolution$ are updated to $true$.

The restriction $canAddToSolution = true$ in Line 13 is the second tabu strategy applied in RWLS. Generally, we do not want the column which has been removed from C to be added back again if none of its neighbors' states

Algorithm 3 Local search improvement

```
1: function LOCALSEARCH( )
2:     while stop criteria not satisfied do
3:         while L = ∅ do
4:             update the best solution
5:             select j ∈ C with the highest score
6:             REMOVE(j)
7:             add newly uncovered rows to L
8:         end while
9:         select j ∈ C ∧ j ∉ tabu_list with highest score and oldest on tie
10:        REMOVE(j)
11:        add newly uncovered rows to L
12:        r ← rand(L)
13:        select d ∈ {d1 ∈ J_r | d1.canAddToSolution = true} with the highest score and oldest on tie
14:        ADD(d)
15:        remove newly covered rows from L
16:        for i ∈ L do
17:            i.weight ← i.weight + 1
18:        end for
19:        put d to tabu_list
20:        j.timestamp ← step
21:        d.timestamp ← step
22:        step ← step + 1
23:    end while
24: end function
```

has changed since its removal. We set $j.canAddToSolution = false$ if j leaves C, which means j is not eligible to be added to C. If one of the states of the neighbors of j changes (due to their removal or addition), $j.canAddToSolution$ is changed to *true*. To save computing time, we implement this strategy along with the operators *ADD* and *REMOVE*.

Finally, the timestamp used in Algorithm 3 makes sure that columns that have not been selected for a longer time are preferred; i.e., when two more columns have the same score, we break ties by preferring the oldest one with the smaller timestamp.

The viability of Line 13 in Algorithm 3 is guaranteed by observation, as bellow:

Lemma 3.1. $\forall i \in L$, $|\{j \in J_i | j.canAddToSolution = true\}| \geq 1$.

Proof: Before the proof, we reassert that after necessary preprocessing, the remaining rows are covered by two or more columns. Then we consider the following two circumstances.

(a) Initially, all the columns have $canAddToSolution = true$, and then an initial solution is constructed. At this time, no columns have left C and no column has a *false* value for *canAddToSolution*. Thus, the proposition holds.

(b) In the period of local search, when a column j leaves C, then we set $j.canAddToSolution = false$, and $\forall j' \in neighbor(j)$, $j'.canAddToSolution = true$. Let's assume that the removal of j causes some row $r \in I_j$ to become

uncovered, because $J_r \cap neighbor(j) \neq \emptyset$. Then, there is at least one columns in J_r whose $canAddToSolution = true$ and, thus, the proposition holds.

3.3. The Row Weighting Scheme in RWLS

The row weighting scheme plays an important role in our algorithm. In RWLS, each row is associated with a weight, which is represented by a positive integer number. Initially, all the rows are given a weight of 1. During the local search improvement phrase, whenever the candidate solution C becomes a partial solution in each iteration, the weight scheme is applied, which means the weights of uncovered rows are increased. In RWLS, the simplest additive increasing method is adopted, which means the weights are simply increased by 1.

Whenever a partial solution with $k-1$ columns has been obtained, RWLS repeatedly perturbs the candidate solution. Since the columns in C are changing dynamically, the uncovered rows in L also change. Because of the weight increasing scheme, the "hard to cover" rows, which have bigger weights, may have good chance to be covered in the following iterations. Both the perturbation and increasing weights help RWLS to escape from potential local optima.

3.4. Analysis of the ADD and REMOVE Operators

As shown in Algorithm 3, the operators *ADD* and *REMOVE* are crucial to RWLS. Therefore, it is necessary to precisely specify them, which is done in Algorithms 4 and 5.

Algorithm 4 Add a column into C

```
1: function ADD(j)
2:     add j to C
3:     j.score ← −j.score
4:     for d ∈ neighbor(j) do
5:         d.canAddToSolution ← true
6:         for r ∈ I_d ∩ I_j do
7:             if |J_r ∩ C| = 1 then
8:                 cover r
9:                 d.score ← d.score − r.weight
10:            else if |J_r ∩ C| = 2 then
11:                if d ∈ C then
12:                    d.score ← d.score + r.weight
13:                end if
14:            end if
15:        end for
16:    end for
17: end function
```

When column j is added or removed, the scores of j and its neighbors are calculated according to Equation (3) and Equation (4), respectively, the

Algorithm 5 Remove a column from C

```
1: function REMOVE(j)
2:     remove j from C
3:     j.score ← −j.score
4:     j.canAddToSolution ← false
5:     for d ∈ neighbor(j) do
6:         d.canAddToSolution ← true
7:         for r ∈ I_d ∩ I_j do
8:             if |J_r ∩ C| = 1 then
9:                 d.score ← d.score − r.weight
10:            else if |J_r ∩ C| = 0 then
11:                uncover r
12:                d.score ← d.score + r.weight
13:            end if
14:        end for
15:    end for
16: end function
```

canAddToSolution of $neighbor(j)$ are updated to *true*. Only when j is removed, $j.canAddToSolution$ is set to *false*. The time complexity of these two operators is dependent on the size of $neighbor(j)$. To further analyze it, we define variables p, q and t. For all d in $neighbor(j)$, let $\gamma(j, d)$ be the set of rows that they both can cover.

$$\gamma(j, d) = I_d \cap I_j \tag{5}$$

$$q = \max\{|\gamma(j, d)||d \in neighbor(j)\} \tag{6}$$

Therefore, if column j is added or removed, the time complexity of the two operators is $\mathcal{O}(|neighbor(j)| \times q)$. More generally, if we define

$$t = \max\{|J_r||r \in M\} \tag{7}$$

$$p = \max\{|I_j||j \in N\} \tag{8}$$

where $\forall j \in N, |neighbor(j)| \leq t$, and $\forall d, j \in N, |\gamma(j, d)| \leq p$, we can conclude that the time complexity of these two operators will not exceed $\mathcal{O}(tp)$. Thus, the two operators used to perturb C are efficient when the product of t and p is relatively small, which is often the case.

4. Computational Results

In order to demonstrate the effectiveness of RWLS, we test it on instances from the OR-Library [4] as well as instances from the Steiner Triple Systems (STS) [27]. There are 87 SCP instances from the OR-Library, in which 70 of them are randomly generated, 7 are very large-scale instances arising from crew-scheduling at Italian railways. The remaining 10 are unicost instances from two combinatorial mathematical models. Similar to what is done in

other works on the USCP [22, 23, 24, 26], we convert the non-unicost instances into USCPs by ignoring the cost information.

For the purpose of fully investigate the performance of RWLS, different stopping criteria are used in our experiments. A maximum number of search steps is adopted at first to show the very best solutions RWLS is able to obtain. When comparing to other algorithms, we resort to the same time limits for the termination of RWLS, thus the comparison can be as fair as possible.

4.1. The Problem Instances

Table 1: Details of the random problem sets [32, 6, 16], the combinatorial problems [22] and the STS instances

Set	m	n	Density(%)	Max number of 1s per row	Num of Instances
4	200	1000	2	36	10
5	200	2000	2	60	10
6	200	1000	5	71	5
A	300	3000	2	81	5
B	300	3000	5	192	5
C	400	4000	2	105	5
D	400	4000	5	244	5
E	50	500	20	124	5
NRE	500	5000	10	561	5
NRF	500	5000	20	1086	5
NRG	1000	10000	2	258	5
NRH	1000	10000	5	580	5
CYC06	240	192	2.1	4–4	1
CYC07	672	448	0.9	4–4	1
CYC08	1792	1024	0.4	4–4	1
CYC09	4608	2304	0.2	4–4	1
CYC10	11520	5120	0.08	4–4	1
CYC11	28160	11264	0.02	4–4	1
CLR10	511	210	12.3	10–126	1
CLR11	1023	330	12.4	20–210	1
CLR12	2047	495	12.5	30–330	1
CLR13	4095	715	12.5	50–495	1
STS243	9801	243	1.2	3–3	1
STS405	27270	405	0.7	3–3	1
STS729	88452	729	0.4	3–3	1
STS1215	245835	1215	0.2	3–3	1

+ m and n refer to the number of rows and columns, respectively.
+ Density is the number of non-zero entries in the matrix.

Table 1 contains the details of the 70 random instances, divided into 12 problem sets (4 to NRH) with the number of rows ranging from 50 to 1000 and the number of columns spanning from 1000 to 10000. Each set of instances is generated according to a specific density, i.e., a percentage of non-zero entries in the (sparse) matrix. Sets 4 to 6 are from [32], A to E are from [6], and NRE to NRH are from [16].

As non-unicost SCPs, the 45 random instances from 4 to 6 and A to D are relatively easy to solve and their optima are known. The mixed integer linear

programming tool CPLEX can solve them in reasonable time [10]. However, no optima are known for the instances that are converted to USCPs. The instances from set E are randomly generated USCPs and their optima can be easily obtained by a greedy procedure [22].

Table 1 also contains the ten combinatorial problem instances (CYC and CLR). The only instance whose optimum is known is CYC06. One obvious feature of the CYC instances is that each row is exactly covered by 4 columns. Different from the random instances, m is always larger than n. For a detailed explanation of the CLR and CYC problems, see [22].

The STS instances are unicost problems have regular structures, such as $|J_i| = 3, \forall i \in M$ and $|I_{j1} \cap I_{j2}| = 1, \forall j1 \neq j2 \in N$. They are generally regarded very difficult for the previous algorithms [19]. The perl script used to generate larger size instance STS1215 can be found in the website [1]. Similar with the combinatorial problems, the STS problems also have much larger m than n.

For the 7 railway crew scheduling instances, because of their very large sizes, we take them seperately in Section 4.4.

4.2. Experimental Results

Our algorithm is programmed in `C`, compiled with `gcc` with `-O2` optimization, running on a machine with Intel(R) Core(TM) i5 650 3.20GHz CPU and 4 GB RAM under a 64-bit Linux system. The maximum number of search steps for the random instances is set to 3×10^7, and to 1×10^8 at first to show the best solutions that RWLS is able to achieve, and then we give direct comparisons with the most effective heuristics found in the literature.

4.2.1. Comparison of Best Solutions Found by Different Algorithms

Table 2 contains the best solutions found by GRASP [23], EM [24], the local search algorithm by Musliu [26] and RWLS on the random instances. For convenience, in the rest of this paper, we will refer to the local search algorithm by Musliu [26] as Musliu's algorithm. The best known solution (BKS) for each instance is also included in the table and we highlight those that are updated by RWLS in boldface and a trailing asterisk. We can see that the best solutions of EM are generally better than those of GRASP and the EM has found the BKS of NRE1, which is 16, whereas Musliu's algorithm has achieved the remaining BKSs on these instances. However, when comparing with RWLS, it is easy to see that the RWLS has surpassed Musliu, since it has discovered all the BKSs and updated 12 of them.

[1] `http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/scp/stcp`

Table 3 contains the best solutions found by GRASP [23], EM [24], Meta-RaPS [21], Musliu [26], 3FNLS and RWLS on combinatorial and STS instances. Because 3FNLS were not tested on the combinatorial problems, the best results obtained by 3FNLS on the CLR and CYC problems are unknown in literature. Similarly, the best solutions of GRASP, EM and Meta-RaPS on the STS problems are also not reported previously.

From Table 3, we can see that the best solutions obtained by RWLS are still better than those of the other four approaches, since it has updated 2 BKSs on these 10 combinatorial problems. Especially, the best solution value of CYC11 is improved from 4088 to 3968. The only instance on which RWLS does not achieve the BKS is CYC10, whose BKS is 1792 [33]. However, the best solution value of RWLS on instance CYC10 is still much better than those of GRASP, EM, Meta-RaPS, and Musliu.

Combining the results from Tables 2 and 3, we can see that RWLS has improved 14 BKSs in total. Since Musliu's algorithm has the best overall performance among all other algorithms, we choose Musliu's algorithm for further comparisons.

Table 2: The best solutions comparison of GRASP [23], EM [24], Musliu [26] and RWLS on the random USCP instances

Inst.	BKS	GRASP	EM	Musliu	RWLS	Inst.	BKS	GRASP	EM	Musliu	RWLS
4.1	38	38	38	38	38	C.1	43	43	43	43	43
4.2	37	37	37	37	37	C.2	43	44	43	43	43
4.3	38	38	38	38	38	C.3	43	44	43	43	43
4.4	38	39	38	38	38	C.4	43	44	43	43	43
4.5	38	38	38	38	38	C.5	43	44	43	43	43
4.6	37	38	38	37	37	D.1	24	25	25	24	24
4.7	38	38	38	38	38	D.2	25	25	25	25	**24***
4.8	37	38	38	37	37	D.3	24	25	25	24	24
4.9	38	38	38	38	38	D.4	25	25	25	25	**24***
4.10	38	38	38	38	38	D.5	25	25	25	25	**24***
5.1	34	35	34	34	34	E.1	5	5	5	5	5
5.2	34	34	34	34	34	E.2	5	5	5	5	5
5.3	34	35	34	34	34	E.3	5	5	5	5	5
5.4	34	34	34	34	34	E.4	5	5	5	5	5
5.5	34	34	34	34	34	E.5	5	5	5	5	5
5.6	34	34	34	34	34	NRE1	16	17	16	17	16
5.7	34	34	34	34	34	NRE2	17	17	17	17	**16***
5.8	34	35	34	34	34	NRE3	17	17	17	17	**16***
5.9	35	36	35	35	35	NRE4	16	17	17	16	16
5.10	34	35	34	34	34	NRE5	17	17	17	17	**16***
6.1	21	21	21	21	21	NRF1	10	10	10	10	10
6.2	20	20	20	20	20	NRF2	10	10	10	10	10
6.3	21	21	21	21	21	NRF3	10	10	10	10	10
6.4	20	21	21	20	20	NRF4	10	10	10	10	10
6.5	21	21	21	21	21	NRF5	10	10	10	10	10
A.1	39	39	39	39	**38***	NRG1	61	-	63	61	61
A.2	38	39	39	38	38	NRG2	62	-	63	62	**61***
A.3	39	39	39	39	**38***	NRG3	62	-	63	62	**61***
A.4	37	38	38	37	37	NRG4	62	-	63	62	**61***
A.5	38	39	38	38	38	NRG5	62	-	63	62	**61***
B.1	22	22	22	22	22	NRH1	34	-	34	34	34
B.2	22	22	22	22	22	NRH2	34	-	34	34	34
B.3	22	22	22	22	22	NRH3	34	-	34	34	34
B.4	22	22	22	22	22	NRH4	34	-	34	34	34
B.5	22	22	22	22	22	NRH5	34	-	34	34	34

+ The best solutions of RWLS are obtained by setting max search step to 3×10^7 as stopping criterion.

+ We emphaize the updated BKSs by RWLS with boldface and trailing asterisk.

Table 3: The best solution comparison of GRASP [23], EM [24], Meta-RaPS [21], Musliu [26], 3FNLS and RWLS on the combinatorial and STS instances

Instance	BKS	GRASP	EM	Meta-RaPS	Musliu	3FNLS	RWLS
CLR10	25	25	25	25	25	-	25
CLR11	23	23	23	23	23	-	23
CLR12	23	23	23	23	23	-	23
CLR13	23	23	23	23	23	-	23
CYC06	60	60	60	60	60	-	60
CYC07	144	144	144	144	144	-	144
CYC08	342	348	344	344	342	-	342
CYC09	774	813	812	793	774	-	**772***
CYC10	1792	1916	1915	1826	1820	-	1798
CYC11	4088	4268	4272	4140	4088	-	**3968***
STS243	198	-	-	-	198	198	198
STS405	335	-	-	-	-	337	335
STS729	617	-	-	-	-	617	617
STS1215	-	-	-	-	-	-	1063

+ The best solutions of RWLS are obtained using max search step 1×10^8 as stopping criterion.
+ We emphaize the updated BKSs by RWLS with boldface and trailing asterisk.
+ The BKS 198 of STS243 has been proven to optimality [34].
+ The BKS 335 of STS405 is recently found by Resende et al. [35] using a biased random-key genetic algorithm dedicated for the STS problems.

4.2.2. Further Comparison with Musliu's Algorithm and 3FNLS

The experimental results in the previous section have demonstrated the ability of RWLS in finding high quality solutions. It has found 14 new best known solutions among 80 benchmark problem instances in the OR-library. This section will examine the reason of RWLS's ability in finding high quality solution. Is it because it used much longer computation time than other algorithms or is it because of the novel hybridization of different search operators and the weighting scheme? To answer such questions in detail, we will compare RWLS against Musliu's algorithm as well as the well-known 3-flip neighborhood local search (3FNLS) by Yagiura et al. [19]. We have seen from Table 2 and Table 3 that Musliu's algorithm has the best solution qualities among existing algorithms on the USCP instances from the OR-Library. The 3FNLS algorithm, on the other hand, represents one of the most effective algorithm that combines Lagrangian Relaxation and local search. It has been known to be effective on a variety of instances, achieved state-of-the-art results on the non-unicost very large-scale railway crew scheduling instances. However, the effectiveness of 3FNLS is not previously tested on the OR-Library instances as unicost problems.

In order to make the comparisons to be as fair as possible as well as carry out direct comparisons with 3FNLS, we asked the Musliu for the executable of his solver on Linux. For 3FNLS, we asked the author Yagiura to provid us their source code of 3FNLS, which is written in `C`. The same as RWLS, we compile 3FNLS on our machine using `gcc`, with O2 option. Both Musliu,

3FNLS and RWLS are running on the same Intel Core i5 3.2Ghz CPU, 4GB RAM machine under 64bit Linux system. Duo to the randomness of the algorithms, for each instance, 10 independent trials are performed with different seeds, and the computational results are presented as the best solution (*best*), average solution (*avg*) among the 10 trials, the number of trials that the best is found as well as the average time (*time*) on these runs detecting the best. We follow the suggestion of Musliu, set the tabu factor for the random problems (4 to NRH) as 0.05, and 0.15 for the combinatoral and STS problems. The stopping criterion of the three algorithms are set to the same time limits, which are roughly set according to the report in [26].

Table 4 and Table 5 contain the computational results of Musliu, 3FNLS and RWLS. As shown in Table 4, for random instances from sets 4 to 6 and sets A to E, when RWLS and 3FNLS both achieve the same best solutions, RWLS is more efficient computationally, because it consumes less average runtimes to achieve the same best and smaller averages. Moreover, RWLS has obtained 9 better solutions than Musliu and 3FNLS among these 50 instances. As for the larger instances from sets NRE to NRH, the overall solution qualities of RWLS are still better than Musliu and 3FNLS, both of the best and the average solutions, while the average runtimes reported by RWLS are generally much larger than those of Musliu's algorithm. Observed that the computational times of Musliu are always tend to be very small even given longer runtimes, we suspect that Musliu's algorithm is quickly stuck in local optima after certain iterations of process.

Table 5 also contains the comparison between Musliu's algorithm, 3FNLS and RWLS on the ten combinatorial and 4 STS instances. We can see that RWLS outperforms Musliu's algorithm in terms of solution quality by finding the same or better solutions in all cases, although tends to consume a little more computation time in some small size problems. RWLS also outperforms 3FNLS on all the combinatorial and STS problems both in solution qualities and computational times. It is quite interesting that 3FNLS's solution qualities are generally better than Musliu, although 3FNLS's performance on USCPs are not previously investigated by their authors. But when comparing to RWLS, it is easy to see that our algorithm outperforms 3FNLS, for it can always obtain better solution qualities within shorter runtimes, especially on the larger ones which have many more rows than columns, such as CYC11 and STS1215.

Combining the results of Table 4 and Table 5, we can see that RWLS has obtained 19 better solutions than Musliu and 3FNLS given the same amount of running time on the same machine.

Table 4: Computational results comparison between Musliu, 3FNLS and RWLS on the random USCP instances

Inst	Musliu				3FNLS				RWLS			
	best	avg	#best	time	best	avg	#best	time	best	avg	#best	time
4.1	38	38.1	9	0.0	38	38.2	8	2.79	38	38.0	10	0.02
4.2	37	37.0	10	0.0	37	37.0	10	0.29	37	37.0	10	0.01
4.3	38	38.0	10	0.0	38	38.0	10	0.22	38	38.0	10	0.01
4.4	38	38.9	3	0.0	38	38.9	1	5.19	38	38.0	10	0.17
4.5	38	38.1	9	0.0	38	38.0	10	1.96	38	38.0	10	0.02
4.6	37	37.4	6	0.0	37	37.6	4	5.58	37	37.0	10	0.13
4.7	38	38.5	5	0.1	38	38.2	8	3.88	38	38.0	10	0.07
4.8	37	37.9	1	0.0	37	37.7	3	5.68	37	37.0	10	0.08
4.9	38	38.2	8	0.0	38	38.0	10	2.91	38	38.0	10	0.02
4.10	38	38.4	7	0.0	38	38.9	1	1.14	38	38.0	10	0.15
5.1	35	35.1	9	0.1	35	35.0	10	0.25	**34**	34.0	10	0.40
5.2	35	35.1	9	0.0	34	34.5	5	4.52	34	34.0	10	0.10
5.3	35	35.3	7	0.0	34	34.0	10	1.20	34	34.0	10	0.04
5.4	35	35.1	9	0.0	34	34.4	6	3.86	34	34.0	10	0.07
5.5	35	35.1	9	0.0	34	34.0	10	2.57	34	34.0	10	0.06
5.6	34	34.3	6	0.5	34	34.3	7	5.62	34	34.0	10	0.09
5.7	34	34.9	3	0.0	34	34.0	10	1.64	34	34.0	10	0.04
5.8	35	35.4	7	0.2	34	34.7	3	1.77	34	34.0	10	0.17
5.9	35	36.7	1	0.0	35	35.0	10	1.18	35	35.0	10	0.03
5.10	35	35.6	5	0.0	34	34.9	1	6.24	34	34.0	10	0.16
6.1	21	21.2	8	0.0	21	21.0	10	0.32	21	21.0	10	0.02
6.2	20	20.7	3	0.1	20	20.7	3	7.35	20	20.0	10	0.17
6.3	21	21.1	9	0.1	21	21.0	10	0.59	21	21.0	10	0.02
6.4	21	21.1	9	0.0	21	21.0	10	1.34	**20**	20.0	10	0.48
6.5	21	21.0	10	0.0	21	21.0	10	1.36	21	21.0	10	0.03
A.1	39	39.0	10	0.0	39	39.0	10	5.56	**38**	38.9	1	10.03
A.2	39	39.1	9	0.4	39	39.0	10	3.40	**38**	38.0	10	3.46
A.3	39	39.0	10	0.0	39	39.0	10	3.01	**38**	38.7	3	14.10
A.4	37	37.9	1	1.0	38	38.0	10	4.42	37	37.1	9	4.39
A.5	38	38.9	1	0.4	38	38.9	1	17.84	38	38.0	10	0.42
B.1	22	22.9	2	0.2	22	22.4	6	10.49	22	22.0	10	0.35
B.2	22	22.0	10	0.0	22	22.0	10	6.09	22	22.0	10	0.31
B.3	22	22.0	10	0.0	22	22.3	7	12.33	22	22.0	10	0.68
B.4	23	23.0	10	0.0	23	23.0	10	1.73	22	22.0	10	1.07
B.5	22	22.5	5	0.6	22	22.1	9	6.37	22	22.0	10	0.68
C.1	43	43.8	2	1.1	43	43.4	6	8.66	43	43.0	10	0.81
C.2	43	43.9	2	0.5	43	43.9	1	17.63	43	43.0	10	1.14
C.3	43	43.5	5	1.0	43	43.8	2	13.39	43	43.0	10	0.73
C.4	44	43.9	1	0.7	43	43.8	2	9.65	43	43.0	10	0.82
C.5	43	43.7	3	0.6	43	43.9	1	5.75	43	43.0	10	4.25
D.1	25	25.8	2	0.0	25	25.1	9	12.37	**24**	24.1	9	7.27
D.2	25	25.3	7	0.9	25	25.0	10	4.86	**24**	24.9	1	8.54
D.3	25	25.3	7	0.3	25	25.4	6	14.32	**24**	24.9	1	5.60
D.4	25	25.6	4	0.5	26	26.0	10	0.00	25	25.0	10	1.69
D.5	25	25.4	6	1.0	26	26.0	4	14.42	**24**	24.9	1	18.27
E.1	5	5.0	10	0.0	5	5.0	10	0.0	5	5.0	10	0.0
E.2	5	5.0	10	0.0	5	5.0	10	0.0	5	5.0	10	0.0
E.3	5	5.0	10	0.0	5	5.0	10	0.0	5	5.0	10	0.0
E.4	5	5.0	10	0.0	5	5.0	10	0.0	5	5.0	10	0.0
E.5	5	5.0	10	0.0	5	5.0	10	0.0	5	5.0	10	0.0

+ Musliu, 3FNLS and RWLS are all run on the same Intel(R) Core(TM) i5 650 3.20GHz CPU, 4GB RAM machine, under 64bit Linux system.

+ Time limits for 4 – 6 and A – E are set to 10 seconds and 20 seconds, respectively.

+ For each instance, the results are reported as the best solution (*best*), the average solution (*avg*) from the 10 runs, the number of runs (#*best*) that the *best* is found as well as the average time (*time*) over the these runs decting the *best*.

+ We emphasize our better solutions than Musliu and 3FNLS with boldface.

Table 5: Computational results comparison between Musliu, 3FNLS and RWLS on the random NRE to NRH, combinatorial and STS problems

Inst	Musliu				3FNLS				RWLS			
	best	avg	#best	time	best	avg	#best	time	best	avg	#best	time
NRE1	17	17.2	8	2.4	17	17.3	7	62.16	17	17.0	10	8.82
NRE2	17	17.2	8	0.8	17	17.2	8	38.80	17	17.0	10	3.64
NRE3	17	17.2	8	0.0	17	17.0	10	51.22	17	17.0	10	3.05
NRE4	17	17.1	9	0.1	17	17.4	6	54.98	17	17.0	10	3.64
NRE5	17	17.4	6	0.7	17	17.2	8	67.59	17	17.0	10	10.76
NRF1	10	10.8	2	1.2	11	11.0	10	0.00	10	10.1	9	25.31
NRF2	10	10.8	2	3.2	10	10.9	1	89.45	10	10.2	8	55.24
NRF3	10	10.6	4	3.2	10	10.9	1	24.44	10	10.1	9	28.34
NRF4	10	10.9	1	3.8	11	11.0	10	0.00	10	10.0	10	36.29
NRF5	10	10.8	2	2.9	11	11.0	10	0.00	10	10.1	9	34.91
NRG1	62	62.6	4	3.2	63	63.3	7	63.81	**61**	61.3	7	54.05
NRG2	62	62.2	8	3.4	62	63.0	1	88.96	**61**	61.5	5	69.76
NRG3	63	63.0	10	2.7	63	63.4	6	55.69	**61**	61.7	3	82.09
NRG4	63	63.2	8	2.6	63	63.3	7	50.41	**61**	61.9	1	86.72
NRG5	62	63.1	1	3.0	63	63.4	6	30.09	**61**	61.9	1	88.44
NRH1	34	34.8	2	7.4	35	35.3	7	53.36	34	34.9	1	53.22
NRH2	35	35.0	10	1.5	35	35.2	8	62.59	35	35.0	10	15.39
NRH3	35	35.0	10	0.5	34	35.2	1	59.35	34	34.9	1	97.02
NRH4	34	34.9	1	6.9	35	35.3	7	75.01	34	34.8	2	97.46
NRH5	34	34.9	1	4.3	35	35.1	9	50.73	34	34.9	1	25.56
CLR10	25	25.1	9	0.0	25	25.0	10	2.75	25	25.0	10	0.01
CLR11	23	23.0	10	0.0	23	23.1	9	11.69	23	23.0	10	0.08
CLR12	23	23.0	10	0.5	23	25.1	1	13.87	23	23.0	10	0.38
CLR13	23	24.1	5	5.6	29	29.7	5	16.71	23	23.0	10	3.89
CYC06	60	60.0	10	0.0	60	60.0	10	0.00	60	60.0	10	0.00
CYC07	144	144	10	0.0	144	144.0	10	0.00	144	144.0	10	0.02
CYC08	349	349.9	6	1.8	342	343.8	1	46.60	342	342.0	10	0.30
CYC09	809	813.0	3	15.6	780	780.1	9	104.39	**772**	773.6	2	266.70
CYC10	1894	1909.9	1	42.7	1801	1807.2	1	819.80	**1798**	1798.6	7	663.73
CYC11	4270	4271.1	1	0.9	4103	4144.9	3	392.36	**3968**	4021.1	1	520.69
STS243	198	201.7	2	1.6	198	198.0	10	170.50	198	198.0	10	0.09
STS405	343	345.0	4	7.6	336	336.0	10	151.07	**335**	335.7	3	117.81
STS729	649	649.8	4	87.9	617	630.3	3	831.78	617	617.0	10	23.36
STS1215	1119	1119.0	10	0.1	1071	1076.3	1	1659.63	**1063**	1065.9	1	886.25

+ Musliu, 3FNLS and RWLS are all run on the same Intel(R) Core(TM) i5 650 3.20GHz CPU, 4GB RAM machine, under 64bit Linux system.
+ Time limit for NRE – NRH is set to 100 seconds.
+ Time limit for CLR10 – CLR13, CYC06 – CYC08 and STS243 is set to 20 seconds.
+ Time limit for CYC09 – CYC11 and STS243 – STS729 instances is set to 1000 seconds.
+ Time limit for STS1215 is set to 2000 seconds due to its larger size.
+ For each instance, the results are reported as the best solution (*best*), the average solution (*avg*) from the 10 runs, the number of runs (*#best*) that the *best* is found as well as the average time (*time*) over these runs decting the *best*.
+ We emphasize our better solutions than Musliu and 3FNLS with boldface.

It is by now clear that RWLS can achieve better or the same solution quality within the same time limits as Musliu and 3FNLS on the 70 random USCP instances, 10 combinatorial and 4 STS instances, which shows the inherent advantages of RWLS in solving USCP, primarily derived from its unique hybridization of different search operators and the weighting scheme. In order to gain a deeper understanding of what caused the good performance of RWLS on USCP instances, several main differences and similarities between Musliu, 3FNLS and RWLS are worth noting.

First, we would like to discuss the fitness functions of three algorithms. For Musliu and 3FNLS, they all define a penalty function as their fitness functions, specifically, Musliu's fitness function is defined as the number of the number of uncovered rows plus the cardinality of the candidate solution, while when solving USCPs, the penalty function of 3FNLS can be seen as

the sum of the penalty weights of rows plus the cardinality of the current candidate solution. The main difference turns out to be that 3FNLS assign penalty weights to rows, and during its local search, it adaptively adjust the weights whenever the search gets stuck. But the fitness function of Musliu can be seen as each row is assign a constant weight 1, and never changes during its local search. According to our exprimental results, although Musliu can almost always find a good solution quickly, as runtime increases, 3FNLS usually obtains the same or even better solutions than Musliu. Whereas for RWLS, the fitness function is not clearly defined, but in each iteration, after randomly choose a uncovered row, than the best column with the greatest score is selected into the candidate solution, and when removing, the column with the smallest absolute score value is always selelcted. Therefore, RWLS always prefer candidate solution with larger total score value of the columns.

Second, the tabu mechanisms are different. Musliu defines a tabu list which is decided by the production of a tabu factor parameter and the cardinality of the initial solution. It stores the information for the columns which are removed or added in the past certain iterations, and such columns are not permitted to be selected in the following iterations. For RWLS, a variety of tabu strategies are adopted, including a timestamp method, the $canAddToSolution$ restriction, as well as not allowing the last two removed or added columns to be selected immediately in the next iteration. The main advantage of our tabu mechanism is that it is universal for different instances, whereas the performance of Musliu heavily depends on the tabu factor factors for different kinds of problems. In fact, the best solutions previously reported by Musliu are obtained by exploiting different tabu factors for every individual instances, which is beyond our work, thus we only use the suggested parameters in our experiments. We do not find obvious tabu mechanisms in the 3FNLS algorithm.

Third, the strategies used to escape from local optima are different, which may be the most significant difference between Musliu, 3FNLS and RWLS. RWLS uses a weighting scheme to update the weights of uncovered rows when stuck in local optima. The adaptive adjustment of weights in each iteration leads to a good chance to escape from a local optimum. As mentioned above, Musliu can be seen as assign all rows a constant weight of 1, and thus may get stuck in a local optimum after certain iterations. Different with Musliu, 3FNLS also uses a mechanism to adaptively adjust the weights of rows during its local search, but unlike our weighting increase scheme, 3FNLS uses complex weighting adjust techniques that tends to consume more computing.

Fourth, we want to further note that although 3FNLS has a sophisticated local search procedure, it also uses the information from solving Lagrangian Relaxation for prioritizing columns during the search. But in some situations,

information from Lagrangian Relaxation would become useless in some cases, such as for the STS problems [19].

4.3. The Effectiveness of the Row Weighting Scheme

In RWLS, the weighting scheme is used to help the search from escaping from local optima. To investigate the effectiveness of this method, we execute our algorithm without it, which means that the weights of the rows remain 1 all the time. We then compare the results with the original RWLS on the hardest combinatorial instance CYC11. To distinguish between these two algorithms, we name the one without row weighting scheme as RWLS-1.

Table 6: The effectiveness of the row weighting scheme on CYC11

Algorithm \ Step	1	10^2	10^3	10^4	10^5	10^6	10^7	10^8
RWLS	4799	4742	4628	4598	4516	4289	4199	3968
RWLS-1	4799	4742	4626	4421	4371	4317	4317	4317

The first row of Table 6 contains the results obtained by RWLS, and the second row contains the results of RWLS-1, at different search steps, respectively. We can see that, initially at Step 1, RWLS and RWLS-1 have the same solution because they share the same initial solution construction method. During the first 10^5 search steps, RWLS-1 is able to obtain better solutions than RWLS, but after that, the solutions found by RWLS continue to improve, whereas those of RWLS-1 remain the same. It is clear that the row weighting scheme helps RWLS to avoid being trapped in a local optimum and to continue exploring the solution space. Similar observations can be made on other instances.

4.4. Evaluation RWLS on the Railway Instances

To show the effectiveness of RWLS, we evaluate it further on the challenging railway instances. Table **??** gives the details of the railway crew scheduling instances from the OR-Library [4]. These instances are very large, rising up to thousands of rows and millions of columns. Hence, directly tackling them can seldom produce high quality solutions. Noting that the railway instances have many more columns than rows, one type of approaches to deal with such instances is to use Lagrangian relaxation and its dual information to reduce the number of columns. This is indispensable in several heuristics and has been shown to be very effective for the non-unicost instances [17, 18, 19]. We will incorporate such a technique into RWLS.

Table 7: Details of the railway instances

Instance	m	n	Density	Instance	m	n	Density
RAIL507	507	63009	1.2	RAIL2586	2586	920683	0.4
RAIL516	516	47311	1.3	RAIL4284	4284	1092610	0.2
RAIL582	582	55515	1.2	RAIL4872	4872	968672	0.2
RAIL2536	2536	1081841	0.4				

[+] We turn this set of instances into unicost problems by ignoring the cost information.

More precisely, we use the problem size reduction technique from 3FNLS [19], which is based on Lagrangian relaxation and uses the subgradient method to solve the *core problem* defined by [18]. By incorporating this technique, we adapt our RWLS to Algorithm 6, which is noted as RWLS-R. The problem size reduction in 3FNLS, which is also named variable fixing, has two phases, i.e., the initial fixing stage and the modification stage. Initially, the subgradient method is called to solve the Lagrangian dual relaxation to obtain the Lagrangian cost for columns (variables) and only columns good enough are selected into the local search (the other variables are set to 0). Then, whenever the local search stops, the fixed variables are heuristically adjusted by freeing some variables whose value are zero previously. In RWLS-R, the initial column selection in Line 6 and column addition in Line 9 are based on the first fixing phase and the modify fixing phase in 3FNLS, respectively. The interested reader is referred to [19] for more details.

From Algorithm 6, we can see that the local search is conducted many times, i.e., each time on different sets of selected columns. In Line 5, the local search is on the original problem, and in Line 8, the local search is only on the small set of columns which have been selected. In our experiments, we set the maximum number of search steps for the local search in Line 5 to 1000, and that in Line 8 to $10 * selected\_n$, where $selected\_n$ is the number of selected columns.

Algorithm 6 RWLS-R: Incorporating RWLS with problem size reduction

```
1: function RWLS-R( )
2:     read problem instance
3:     preprocessing to add columns that able to cover some rows alone to the candidation solution
   permanently
4:     init a solution
5:     local search
6:     column selection
7:     while stopping time not reached do
8:         local search on the selected columns
9:         add some new columns to the selected columns
10:        restart the local search by initiating the candidate solution as the best found solution
11:    end while
12: end function
```

In order to compare the results with CPLEX, we run our algorithms and 3FNLS on an Intel Duo Core 2.4GHz CPU with 2 GB RAM machine which has CPLEX12.5 [2] installed. We set the maximum runtime for RWLS-R and 3FNLS to 100 seconds for instances RAIL507, 516 and 586, 1000 seconds for $RAIL2536$, 2586, 4284 and 4872 because of their larger sizes. For each instance, the results of ten independent runs of RWLS-R and 3FNLS are shown in 8. To the best of our knowledge, no results have yet been reported by other methods that treat these railway instances as USCPs.

From Table 8, we can see that for the first three instances (RAIL507, 516, 582), CPLEX is able to solve them to optimality in 1375.57 seconds, 6.49 seconds, 158.67 seconds, respectively. However, because the last four instances (RAIL2536, 2586, 4284, 4872) are very large, CPLEX failed to produce solutions on these instances.

According to Table 8, we can find good solutions to all railway instances, including the last four large instances where CPLEX fails to produce a solution. On the first three instances, both RWLS-R and CPLEX find good solutions, whereas the best solution by 3FNLS on RAIL582 is inferior than CPLEX and RWLS-R. However, on the four larger ones (RAIL2536, 2586, 4284, 4872), 3FNLS is able to obtain better solutions than RWLS-R. Here, we also report the lower bounds found by 3FNLS on instances RAIL2536, 2586, 4284, 4872 are 363, 505, 579, 857, respectively.

Table 8: Test railway instances as unicost problems

Instance	RWLS-R				3FNLS				CPLEX12.5	
	best	avg	#best	time	best	avg	#best	time	sol	time
RAIL507	96	96.9	1	35.24	96	96.7	3	67.88	96	1375.57
RAIL516	134	134.1	9	65.45	135	135.6	4	40.93	134	6.49
RAIL582	126	126.3	7	27.60	126	126.0	10	31.37	126	158.67
RAIL2536	*381*	381.6	4	373.12	378	379.2	2	733.34	-	-
RAIL2586	*520*	521.6	2	300.62	518	518.9	2	391.10	-	-
RAIL4284	*597*	599.4	3	550.40	594	595.0	3	834.31	-	-
RAIL4872	*882*	884.6	2	778.28	879	880.5	1	817.12	-	-

[+] For 3FNLS, we convert these problems to unicost by replacing the cost information to 1 for all columns.

4.5. How Good Is CPLEX in Solving USCP?

For non-unicost SCPs, it has been shown the random instances from sets 4 to 6 and A to D can be solved to optimality by CPLEX in reasonable time [10]. However, the USCP is generally considered to be harder to solve than non-unicost SCPs [3]. In order to find out the difficulties of the random

[2] According to IBM, the newly released CPLEX12.5 is generally 50% faster than its earlier releases in the last ten years.

USCP instances from 4 to 6 and A to D, we apply CPLEX to these 45 instances.

In Table 8, we report the best solutions found by CPLEX within 100 seconds for groups 4 to 6, since they are quite small and generally regarded as easy, and 1000 seconds for groups A to D because of their larger sizes. The table includes the BKS (not those updated by RWLS) for comparison. It can be seen that for these 45 instances, CPLEX can only achieve seven BKSs. In fact, according to our experience, as time increases, solutions found by CPLEX improve very slowly. For instance 4.1, CPLEX can find a solution of 39 in 100 seconds, but it takes about 1000 seconds to achieve the BKS of 38. Similarly on the NRE1 instance, CPLEX needs about 15000 seconds to achieve a solution of 17. It keeps running for about 50000 seconds before terminating due to an out of memory error and the solution still remains 17.

Table 9: Results of CPLEX12.5 on instances from 4 – 6 and A – D as USCPs

Instance	BKS	CPLEX12.5	Instance	BKS	CPLEX12.5
4.1	38	39	A.1	39	40
4.2	37	37	A.2	38	40
4.3	38	38	A.3	39	40
4.4	38	40	A.4	37	38
4.5	38	38	A.5	38	39
4.6	37	38	B.1	22	22
4.7	38	39	B.2	22	23
4.8	37	38	B.3	22	23
4.9	38	39	B.4	22	23
4.10	38	39	B.5	22	23
5.1	34	35	C.1	43	44
5.2	34	35	C.2	43	44
5.3	34	34	C.3	43	45
5.4	34	35	C.4	43	44
5.5	34	36	C.5	43	45
5.6	34	35	D.1	24	25
5.7	34	35	D.2	25	26
5.8	34	34	D.3	24	26
5.9	35	36	D.4	25	26
5.10	34	36	D.5	25	26
6.1	21	22			
6.2	20	21			
6.3	21	22			
6.4	20	21			
6.5	21	21			

+ Time limits are set to 100 seconds for instances from 4 to 6.
+ Time limits are set to 1000 seconds for instances from A to D.
+ We place the best know solutions along side to show the solution qualities of CPLEX.
+ The results are obtained by an Intel Core Due 2.4GHz CPU with 2 GB RAM machine.

The results in Table 9 indicate that the 45 USCP instances are not easy to solve, although their non-unicost versions are. Combining the results of

RWLS from Table 4, we can conclude that RWLS is much better than CPLEX on USCP instances, because it almost always achieves or even updates the BKSs.

5. Conclusion and Future Work

In this paper, we have introduced a new local search heuristic, dubbed RWLS, for USCPs. We proposed a local improvement framework to iteratively reduce the size of the currently best solution, which is realized by using two efficient operators to perturb the currently best solution when it becomes infeasible. In addition, several widely used strategies are adopted by RWLS, including a weighting scheme that adaptively updates the weights of rows (elements) to help RWLS to escape from local optima, two tabu strategies to avoid cycles and a timestamp method to break ties whenever adding or removing a set. RWLS successfully hybridized these general strategies into its local search framework.

The effectiveness and efficiency of RWLS have been evaluated on a large number of instances from the OR-Library [4] and Steiner triple systems [27], which vary from hundreds of rows (elements) and thousands of columns (sets) to tens of thousands of rows and columns. The experimental results show that RWLS has an excellent performance and outperforms existing state-of-the-art algorithms. It has updated 14 best known solutions in the literature. For the combinatorial instance CYC11, the best know solution is updated from 4088 to 3968.

RWLS is especially effective on the ten combinatorial instances from the OR-Library as well as instances from the Steiner triple systems, which contain many more rows than columns. However, for instances containing a significantly larger number of columns but a few rows, the problem size reduction techniques should be adopted. In Section 4.4, we have shown the effectiveness of RWLS in dealing with such USCP instances by incorporating with the problem size reduction technique from [19]. This is the first time that the seven railway crew scheduling instances were solved as USCPs, outperforming CPLEX 12.5.

In spite of excellent performance of RWLS on 91 USCP benchmark instances, more work is needed in the future. First, the extended algorithms RLWS-R was was outperformed by 3FNLS on the four larger RAIL problems. The reason for this needs to be studied. Second, the study in this paper is experimental in nature. It will be necessary to analyze the algorithm as well as the characteristics of the benchmark instances (especially the hardest ones) theoretically, so that we can understand more which algorithmic features are most important in solving what kinds of USCP instances. Third,

we have not analyzed in depth the impact of different tabu strategies on RWLS's performance, which should be done in the future. Fourth, it would be useful to investigate automatic stopping techniques for RWLS, instead of setting a time limit in advance. Fifth, it would be interesting to adapt some of RWLS's ideas to solve other hard combinatorial optimization problems.

Acknowledgments

We are grateful for the authors N. Musliu and M. Yagiura for providing their executable and source code of their algorithms, respectively. We are appreciate for the valuable comments of the anonymous reviewers, which have been very helpful for improving the quality of this paper.

This research work was partially supported by an EPSRC grant (No. EP/I010297/1) and the Fundamental Research Funds for the Central Universities (WK0110000023). Xin Yao was supported by a Royal Society Wolfson Research Merit Award. Thomas Weise was supported by 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] A. Caprara, M. Fischetti, P. Toth, D. Vigo, P. L. Guida, Algorithms for railway crew management, Mathematical Programming 79 (1-3) (1997) 125–141.

[2] J. Bautista, J. Pereira, Modeling the problem of locating collection areas for urban waste management. an application to the metropolitan area of barcelona, Omega 34 (6) (2006) 617–629.

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

[4] J. E. Beasley, OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society (1990) 1069–1072.

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

[6] J. E. Beasley, An algorithm for set covering problem, European Journal of Operational Research 31 (1) (1987) 85–93.

[7] M. L. Fisher, P. Kedia, Optimal solution of set covering/partitioning problems using dual heuristics, Management Science 36 (6) (1990) 674–688.

[8] E. Balas, M. C. Carrera, A dynamic subgradient-based branch-and-bound procedure for set covering, Operations Research 44 (6) (1996) 875–890.

[9] J. E. Beasley, K. Jörnsten, Enhancing an algorithm for set covering problems, European Journal of Operational Research 58 (2) (1992) 293–300.

[10] A. Caprara, P. Toth, M. Fischetti, Algorithms for the set covering problem, Annals of Operations Research 98 (1-4) (2000) 353–371.

[11] V. Chvatal, A greedy heuristic for the set-covering problem, Mathematics of Operations Research 4 (3) (1979) 233–235.

[12] F. J. Vasko, An efficient heuristic for large set covering problems, Naval Research Logistics Quarterly 31 (1) (1984) 163–171.

[13] T. A. Feo, M. G. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (2) (1989) 67–71.

[14] J. E. Beasley, P. C. Chu, A genetic algorithm for the set covering problem, European Journal of Operational Research 94 (2) (1996) 392–404.

[15] L. W. Jacobs, M. J. Brusco, Note: A local-search heuristic for large set-covering problems, Naval Research Logistics 42 (7) (1995) 1129–1140.

[16] J. Beasley, A lagrangian heuristic for set-covering problems, Naval Research Logistics (NRL) 37 (1) (1990) 151–164.

[17] S. Ceria, P. Nobili, A. Sassano, A lagrangian-based heuristic for large-scale set covering problems, Mathematical Programming 81 (2) (1998) 215–228.

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

[19] M. Yagiura, M. Kishida, T. Ibaraki, A 3-flip neighborhood local search for the set covering problem, European Journal of Operational Research 172 (2) (2006) 472–499.

[20] S. Umetani, M. Yagiura, Relaxation heuristics for the set covering problem, Journal of the Operations Research Society of Japan 50 (4) (2007) 350–375.

[21] G. Lan, G. W. DePuy, G. E. Whitehouse, An effective and simple heuristic for the set covering problem, European Journal of Operational research 176 (3) (2007) 1387–1403.

[22] T. Grossman, A. Wool, Computational experience with approximation algorithms for the set covering problem, European Journal of Operational Research 101 (1) (1997) 81–92.

[23] J. Bautista, J. Pereira, A GRASP algorithm to solve the unicost set covering problem, Computers & Operations Research 34 (10) (2007) 3162–3173.

[24] Z. Naji-Azimi, P. Toth, L. Galli, An electromagnetism metaheuristic for the unicost set covering problem, European Journal of Operational Research 205 (2) (2010) 290–300.

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

[26] N. Musliu, Local search algorithm for unicost set covering problem, in: Advances in Applied Artificial Intelligence, Springer, 2006, pp. 302–311.

[27] D. Fulkerson, G. Nemhauser, L. Trotter, Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of steiner triple systems, in: M. Balinski (Ed.), Approaches to Integer Programming, Vol. 2 of Mathematical Programming Studies, Springer Berlin Heidelberg, 1974, pp. 72–81.

[28] J. Thornton, Clause weighting local search for SAT, Journal of Automated Reasoning 35 (1-3) (2005) 97–142.

[29] J. Thornton, D. N. Pham, S. Bain, V. Ferreira Jr, Additive versus multiplicative clause weighting for sat, in: AAAI, Vol. 4, 2004, pp. 191–196.

[30] F. Hutter, D. A. Tompkins, H. H. Hoos, Scaling and probabilistic smoothing: Efficient dynamic local search for sat, in: Principles and Practice of Constraint Programming-CP 2002, Springer, 2006, pp. 233–248.

[31] P. Morris, The breakout method for escaping from local minima, in: Proceedings of the eleventh national conference on Artificial intelligence, AAAI'93, AAAI Press, 1993, pp. 40–45.

[32] E. Balas, A. Ho, Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study, Springer, 1980.

[33] H. Harborth, H. Nienborg, Maximum number of edges in a six-cube without four-cycles, Institute für Mathematik, Techn. Univ., 1994.

[34] J. Ostrowski, J. Linderoth, F. Rossi, S. Smriglio, Solving large steiner triple covering problems, Operations Research Letters 39 (2) (2011) 127–131.

[35] M. G. Resende, R. F. Toso, J. F. Gonçalves, R. M. Silva, A biased random-key genetic algorithm for the steiner triple covering problem, Optimization Letters 6 (4) (2012) 605–619.