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, UK

A R T I C L E   I N F O

Article history:
Received 9 January 2014
Accepted 15 May 2015
Available online 22 May 2015

Keywords:
Combinatorial optimization
Unicost set covering problem
Row weighting local search

A B S T R A C T

The Set Covering Problem (SCP) is NP-hard. We propose a new Row Weighting Local Search (RWLS) algorithm for solving the unicost variant of the SCP i.e. USCPs where the costs of all sets are identical. RWLS is a heuristic algorithm that has three major components united in its local search framework: (1) a weighting scheme, which updates the weights of uncovered elements 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 has been evaluated on a large number of problem instances from the OR-Library and compared with other approaches. It is able to find all the best known solutions (BKS) and improve 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 to the hardest instance CYC11 considerably. RWLS is conceptually simple and has no instance-dependent parameters, which makes it a practical and easy-to-use USCP solver.

© 2015 Elsevier B.V. and Association of European Operational Research Societies (EURO) within the International Federation of Operational Research Societies (IFORS). All rights reserved.

1. Introduction

The Set Covering Problem (SCP) is a combinatorial optimization problem with many applications, ranging from crew scheduling in railways to job assignment in manufacturing and service location (Bautista & Pereira, 2006; Caprara, Fischetti, Toth, Vigo, & Guida, 1997). It can be described as follows: given a set of elements X, a set S = {s | s ⊆ X} and X = ∪s∈S s = X, where each subset in S is associated with a cost, and the goal is to find a set F ⊆ S whose union is X (which contains all elements from X) at the minimal total cost. If all subsets in S have identical cost, the problem is referred to as the unicost set covering problem (USCP). Although being a special case of SCP, the unicost version is generally considered to be even harder to solve (Yelbay, Birbil, & Bülbül, 2015) and is the subject of this paper.

Formally, an SCP instance is usually defined as an m × n zero-one matrix A = [a ij] m×n where a ij = 1 means column i j can cover row (element) i. The objective is to find a set of columns at the minimal cost to cover all rows. If the problem is a USCP, the objective is equivalent to finding the smallest set of columns that cover all rows. A candidate solution C can then be represented as a subset of N = {1, ..., n}. Such a solution is feasible if and only if ∑j∈C a ij ≥ 1, ∀i ∈ M = {1, ..., m}.

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

(1) a weighting scheme, which updates the weights of the uncovered rows in order to escape from local optima,
(2) different tabu strategies, which prevent possible cycles during the search, and
(3) a timestamp method to break ties, which makes the sets that are not moved into or out of the candidate solution for a long time are more likely to be selected.

In our experimental studies, RWLS has improved 14 best known solutions in the literature for 87 USCP benchmark instances from the OR-Library (Beasley, 1990b) and shown excellent performance. It is especially effective on the problems where the number of rows (elements) is much larger than the number of columns (sets). However, RWLS is also effective in other cases, for example, for the seven railway crew scheduling instances, with up to millions of columns and thousands of rows. Combining a problem size reduction technique...
from solving Lagrangian Relaxation, although RWLS obtains inferior results to a sophisticated algorithm (Yagiura, Kishida, & Ibaraki, 2006) from the literature for this set of railway crew scheduling instances, it still outperforms CPLEX12.51 consistently and succeeds in finding good solutions to all seven instances while 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 (Garey & Johnson, 1990). Many algorithms have been developed for solving the SCP. Exact approaches (Balas & Carrera, 1996; Beasley, 1987; Beasley & Jörnsten, 1992; Fisher & Kedia, 1990) are mostly based on branch-and-bound or branch-and-cut. Caprara, Fischetti, and Toth (2000) 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 one for SCPs is the greedy algorithm (Chvatal, 1979). Later, several randomized greedy algorithms (Feo & Resende, 1989; Vasko, 1984) are proposed. They usually produce better results than the deterministic greedy one. A variety of other heuristic algorithms have also been proposed, including some general meta-heuristics, such as Genetic Algorithms (Beasley & Chu, 1996), Simulated Annealing (Jacobs & Brusco, 1995) and Lagrangian Relaxation-based heuristics (Beasley, 1990a; Caprara, Fischetti, & Toth, 1999; Ceria, Nobili, & Sassano, 1998; Yagiura et al., 2006), among which the heuristic methods by Ceria et al. (1998), Caprara et al. (1999) and Yagiura et al. (2006) are able to achieve excellent results on the very large-scale instances by exploiting their specific features. In particular, the 3-flip neighborhood local search (3FNLS) method by Yagiura et al. (2006), which combines 3-flip local search, adaptive penalty weights and Lagrangian Relaxation, 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 Umetani and Yagiura (2007).

Lan, Depuy, and Whitehouse (2007) noticed that the cost information plays an important role in determining the performance of Genetic algorithm (Beasley & Chu, 1996), Simulated Annealing (Jacobs & Brusco, 1995) and the Lagrangian-based heuristic (Beasley, 1990a) and did not recommend these algorithms for USCPs. They proposed the Meta-RaPS approach that works effectively for both unicost and non-unicost SCPs. Yelbay et al. (2015) gave a detailed explanation of the usefulness and limitations of dual information from Lagrangian Relaxation or Linear Programming (LP) Relaxation and pointed out that the unicost problems may be more challenging than the non-unicost problems.

There are heuristics dedicated to specifically solving USCPs. Grossman and Wool (1997) compared nine heuristics, including several greedy variants and a neural network algorithm. In their report Grossman and Wool (1997), the randomized greedy variant R-Gr has the best performance on a large set of instances from the OR-Library (Beasley, 1990b). A newer GRASP algorithm incorporating a local improvement procedure from solving Satisfiability (SAT) has shown to be able to obtain better results than R-Gr (Bautista & Pereira, 2007).

The Electromagnetism Meta-heuristic (EM) proposed by Naj-Azimi, Toth, and Galli (2010) creates an 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 escape from local optima, mutation is also adopted. The computational results in Naj-Azimi et al. (2010) show that EM performs much better than GRASP, but in comparison with Meta-RaPS on 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 (Hoos & Stützle, 2005). Musliu (2006) 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 or 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 most of the best known solutions on 80 instances from the OR-Library (Beasley, 1990b) as uncost problems and 5 small instances from the Steiner triple systems (Fulkerson, Nemhauser, & Trotter, 1974).

Since in USCP instances all sets have the same cost, the task can be seen as minimizing the number of sets in a solution. RWLS utilizes the special property of USCP and uses a general search framework to iteratively reduce the size of the candidate solution.

3. Row weighting local search (RWLS)

3.1. Notations and definitions

As mentioned above, the USCP can be presented as an \( m \times n \) zero-one matrix. \( M \) and \( N \) indicates the set of rows (elements in \( X \)) and columns (subset \( s \in S \)), respectively. We define \( \sigma \) as the set of columns that are able to cover row \( i \) and \( I_j \) as the set of rows covered by column \( j \):

\[
\mathcal{I}_j = \{ i \in N | a_{ij} = 1 \}\quad i = 1, \ldots, m. \tag{1}
\]

\[
\mathcal{J}_j = \{ i \in M | a_{ij} = 1 \}\quad j = 1, \ldots, n. \tag{2}
\]

A candidate solution \( C \) is 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 \( j \) in \( C \) that satisfies \( i \in \mathcal{I}_j \).

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 Eq. (3).

\[
j.score(j) = \begin{cases} 
+ \sum_{\sigma(C_i)=0} i.weight & \text{if } j \notin C. \\
- \sum_{\sigma(C_i)=1} i.weight & \text{if } j \in C.
\end{cases} \tag{3}
\]

In Eq. (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 uncovered rows that \( j \) is able to cover. 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 from the definition of \( j.score \) that if we move \( j \) into or out of \( C \), the score of \( j \) should be negated. A row \( i \) with \( \sigma(C_i) > 1 \) has no contribution to the score value of the corresponding columns in \( I_j \), since it has been covered more than once by the candidate solution \( C \).

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

We also define a neighborhood relationship for columns as follows: 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 \( \text{neighbor}(j) \) contains all the neighbors of \( j \), defined as:

\[
\text{neighbor}(j) = \{ d \in N | d \neq j \land I_d \cap I_j \neq \emptyset \}, \quad j = 1, \ldots, n \tag{4}
\]
uncovered rows are maintained in a set named $L$. We examine the number of columns covering each row, i.e., only covered by one column. Such columns must be selected into the set $L$ with the adaptive weightings scheme. The overall procedure of RWLS is described as Algorithm 1.

A preprocessing step is necessary when there are rows that are only covered by one column. The set $L$ represents the initial solution. The set $L$ is initialized as $\emptyset$, representing the initial solution. The set $L$ is constructed greedily in a phase.

### 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 adaptive weighting scheme. The overall procedure of RWLS is described as Algorithm 1.

A preprocessing step is necessary when there are rows that are only covered by one column. Such columns must be selected into the candidate solution, and the rows they cover can be removed from the problem. We examine the number of columns covering each row, i.e., $|J_i|$ for each $i$, and for rows that are only covered by one single column, the corresponding column is selected into the solution and marked permanently not to be removed. This preprocessing step has a time complexity of at most $O(n')$, where $n'$ is the number of ones in the matrix $A$.

#### 3.2.1. Initialization

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

$ADD(j)$ is a simple operator, in which the score of $j$ is negated, and then the scores of $\text{neighbor}(j)$ are updated according to Eq. (3). The $\text{canAddSolutions of neighbor}(j)$ are set to true. Note that when $ADD(j)$ is called, the removal of newly covered rows from $L$ is conducted inside the operator.

#### 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 $C$. Then becomes a 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, then 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 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 (the one closest to zero) in $C$. RWLS keeps track of two previously added columns in the tabu list, i.e., an 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 a column removal, a row is randomly selected from the uncovered row set $L$ and the column with the highest score and $j$.canAddToSolution = true is chosen to be added to $C$.

### 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

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

### Algorithm 2 Initialization.

1: function INIT($C$ candidate solution)
2: for $j \in N$ do
3: $j$.score $\leftarrow 0$
4: $j$.timestamp $\leftarrow 1$
5: $j$.canAddToSolution $\leftarrow$ true
6: end for
7: $L \leftarrow \emptyset$
8: for $i \in M$ do
9: $i$.weight $\leftarrow 1$
10: add $i$ into $L$
11: for $d \in J_i$ do
12: $d$.score $\leftarrow d$.score + $i$.weight
13: end for
14: end for
15: $C \leftarrow \emptyset$
16: while $L \neq \emptyset$ do
17: $j \leftarrow \text{rand}((d \in N \wedge d$.score $= \max\{d1$.score $| d1 \in N \wedge C\}))$
18: $ADD(j)$
19: end while
20: end function

### Algorithm 3 Local search improvement.

1: function LOCALSEARCH( )
2: $step \leftarrow 1$
3: while stop criteria not satisfied do
4: $L = \emptyset$
5: update the best solution
6: select $j \in C$ with the greatest score
7: $ADD(j)$
8: end while
9: select $j \in C \wedge j \notin \text{tabu list}$ with greatest score and the oldest if there is a tie
10: $REMOVE(j)$
11: $r \leftarrow \text{rand}(L)$
12: select $d \in \{d1 \in J_r | d1$.canAddToSolution $= \text{true}\}$ with the greatest score and the oldest if there is a tie
13: $ADD(d)$
14: for $i \in L$ do
15: $i$.weight $\leftarrow i$.weight + 1
16: end for
17: put $d$ to tabu list
18: $j$.timestamp $\leftarrow step$
19: $d$.timestamp $\leftarrow step$
20: $step \leftarrow step + 1$
21: end while
22: end function
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, as shown in Algorithms 4 and 5.

When column $j$ is added or removed, the scores of $j$ and its neighbors are calculated according to Eq. (3) and Eq. (4), respectively. The `canAddToSolution` of neighbors$j$ are updated to true. Only when $j$ is removed, `canAddToSolution` is set to false. The time complexity of these two operators depends 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.

\[
\gamma(j, d) = I_d \cap I_j
\]

(5)

\[
q = \max|\{\gamma(j, d) : d \in \text{neighbor}(j)\}|
\]

(6)

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

\[
t = \max|\{I_r : r \in M\}|
\]

(7)

\[
p = \max|\{I_j : j \in N\}|
\]

(8)

where $\forall j \in N$, $|\text{neighbor}(j)| \leq t$, 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 $O(tp)$. 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 evaluate it on a large number of instances from the OR-Library (Beasley, 1990b) as well as instances from the Steiner Triple Systems (STS) (Fulkerson et al., 1974). There are 87 SCP instances from the OR-Library, of which 70 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 previous work on the USCP (Bautista & Pereira, 2007; Grossman & Wool, 1997; Musliu, 2006; Naji-Azimi et al., 2010), we convert the non-unicost instances into USCPs by ignoring the cost information.

4.1. Problem instances

Table 1 contains the details of the 70 random instances, divided into 12 problem sets (from 4 to NRH) with the number of rows ranging from 50 to 1000 and the number of columns spanning from 500 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 A. Sets 4 to 6 are from Balas and Ho (1980), A to E are from Beasley (1877), and NRE to NRH are from Beasley (1990a).

As non-unicost SCPs, the random instances from 4 to 6, A to D, NRE and NRF, as well as instances NRG1 to NRG4, are relatively easy to solve and their optima are known. The mixed integer linear programming tool CPLEX can solve them in reasonable time (Caprara et al., 2000). 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 (Grossman & Wool, 1997).

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 Grossman and Wool (1997).

The STS instances are unicost problems with regular structures, such as \(|I| = 3, \forall i \in M \text{ and } |J_i \cap J_J| = 1, V_J = \emptyset \). They are generally regarded as very difficult for the existing algorithms (Yagiura et al., 2006). The perl script used to generate the STS instances can be found from the website\(^2\). In STS problems, the number m is also much larger than n.

For the 7 railway crew scheduling instances, because of their very large sizes, we will consider them separately 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 \(\times 10^7\), and to \(1 \times 10^8\) for the combinatorial and STS problems at first, to show the best solutions that RWLS is able to achieve, and then we give direct comparisons to the most effective heuristics found in the literature.

#### 4.2.1. Comparison of best solutions found by different algorithms

Tables 2 and 3 contain the best results found by GRASP (Bautista & Pereira, 2007), EM (Naji-Azimi et al., 2010), local search algorithm by Musliu (2006) and RWLS on the random instances. For convenience, in the rest of this paper, we will refer to the local search algorithm proposed by Musliu (2006) as Musliu. The best known solution (BKS) value for each instance is also included in the table and we highlight those improved by RWLS in starred format. For each instance, 10 runs are executed by RWLS with different random seeds, in addition to the best solution value, we also report the number of runs detecting the best, and the average time over these “successful” runs.

It can be observed from Table 2 that the best solution values of EM are generally better than those of GRASP. EM found the BKS of NRE1, which is 16, whereas Musliu’s algorithm has achieved the remaining BKSs on these instances. However, RWLS is able to surpass Musliu, since it has discovered all the BKSs and even improved 12 of them.

Table 3 contains the best solution values found by GRASP (Bautista & Pereira, 2007), EM (Naji-Azimi et al., 2010), Meta-RaPS (Lan et al., 2007), Musliu (2006), 3FNLS and RWLS on combinatorial and STS instances. Since 3FNLS has not been tested on the CLR and CYC problems by its authors, the best results obtained by 3FNLS on these instances are unknown in the literature. The same is true for GRASP, EM and Meta-RaPS on the STS problems.

From Table 3, we find that the best solutions obtained by RWLS are better than those of the other four approaches. RWLS has improved 2

---

\(^2\) http://www.co.cm.is.nagoya-u.ac.jp/~yagiura/scp/scp/
Table 2
The best solutions found by GRASP (Bautista & Pereira, 2007), EM (Naji-Azimi et al., 2010), Musliu (2006) and RWLS on the random USCP instances. The starred boldface indicates the improvement over the best known solution in the literature.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>BKS</th>
<th>GRASP</th>
<th>EM</th>
<th>Musliu</th>
<th>RWLS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>Time</td>
</tr>
</tbody>
</table>

- RWLS uses a maximum search iteration of $3 \times 10^7$ as its stopping criterion.
- The best solution values of A.2, D.1 and NRE4 of Musliu were found after parameter tuning, and the computation times were not reported.
- The time of EM was the computation time of finding the best. The time of Musliu and RWLS was the average time over the successful runs that found the corresponding best.
BKSs on these 10 combinatorial problems. In particular, the best solution value of CYC11 is improved from 4088 to 3968. The only instance on which RWLS did not achieve the BKS is CYC10, whose BKS is 1792 (Harborth & Niemborg, 1994). However, the best solution from RWLS on CYC10 is still better than those from GRASP, EM, Meta-RaPS and Musliu.

Combining the results in Tables 2 and 3, we can see that RWLS has improved 14 BKSs in total, at the expense of more computation time on some instances. Since Musliu’s algorithm has the best overall performance among all other algorithms, we choose Musliu’s algorithm for further comparison.

Since the results of the other approaches are from the literature, we list the computer configurations used by each algorithm as below:

- GRASP (Bautista & Pereira, 2007): programmed in C, running on a Pentium 4 1800 MHz CPU machine with 512 MB RAM under Linux System. The stopping criterion is a predefined maximum number of iterations. The computation times in finding the best reported solutions were not given.
- EM (Naji-Azimi et al., 2010): programmed in C, running on an Intel Core Duo 1.7GHz CPU with 1 GB RAM. It stops when an indicated global time is reached.
- Meta-RaPS (Lan et al., 2007): running on an Intel Pentium IV 1.7 GHz PC.
- Musliu (2006): written in C++, running on an Intel Pentium 4, 2.4GHz CPU with 512MB RAM machine. For each instance, 10 runs are performed, and the average time over the runs detecting the best is reported.
- 3FNLS (Yagiura et al., 2006): implemented in C, running on a Sun Ultra 2 Model 2300 (two Ultra SPARC II 300 MHz processors with 1 GB RAM) workstation, 10 runs are performed for each instance using various time limits.

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 superior performance in finding high quality solutions. Is it because it used much longer computation time than other algorithms or is it because of the novel combination of different search operators and the adaptive weighting scheme? To answer such questions in detail, we will compare RWLS against Musliu’s algorithm as well as 3FNLS (Yagiura et al., 2006). We have seen from Tables 2 and 3 that Musliu’s algorithm has the best solution values among other existing algorithms on the USCP instances from the OR-Library. The 3FNLS algorithm, on the other hand, represents one of the most effective algorithms that combine 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-uniset, very large-scale railway crew scheduling instances. However, the effectiveness of 3FNLS has not been tested on the OR-Library instances as uniset problems.

In order to make direct comparison between these three algorithms, we asked Musliu for the executables of his solver on Linux. For 3FNLS, we asked Yagiura to share with us their source code of 3FNLS, which is written in c. Similar to 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 3.2 GHz CPU, 4GB RAM machine under the 64bit Linux system. Due to the randomness of the three algorithms, we follow the experiment setup as Musliu, perform 3FNLS and RWLS 10 independent trials on each instance with 10 consecutive integer numbers ranging from 11 to 20 as random seeds. The computational results are presented as the best solution value (best), average solution value (avg) among the 10 trials, the number of trials finding the best, as well as the average time (time) over those “successful” runs that delivered the best. We follow the suggestion of Musliu, set the tabu factor for the random problems (4 to NRH) to 0.05, and 0.15 for the combinatorial and STS problems, respectively. The stopping criteria of the three algorithms are set to the same time limits. Note that because of the different features each algorithm possesses, it is difficult to set time limits fair to each algorithm. However, the times reported in Musliu (2006) provide a reference of the hardness of these USCP instances from OR-Library. Hence we set the time limits for each set of instances roughly according to Musliu’s algorithm.

Tables 4 and 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 times to achieve the same best and smaller averages. Moreover, RWLS has obtained 9 better solutions than Musliu and 3FNLS among 50 instances. As for the larger instances from sets NRE to NRH, the overall solution values of RWLS are still better than those of Musliu and 3FNLS, both of the best and the average solution values, while the time reported by RWLS are generally much larger than those of Musliu’s algorithm. Observed that the computational times of Musliu are always very small even given larger time limits, we suspect that Musliu’s algorithm might be quickly stuck in some local optima after certain iterations.

Table 4 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 it tends to consume a little more computation time on some small size problems. RWLS also outperforms 3FNLS on all the combinatorial and STS problems both in terms of best solution values and computational times. It is interesting that 3FNLS’s solution values are generally better than those of Musliu although 3FNLS’s performance on USCPs is not previously investigated by its authors. However, when compared to RWLS, it is easy to see that our algorithm outperforms 3FNLS, for it can always obtain better solutions within shorter runtimes, especially on the larger instances.

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 it tends to consume a little more computation time on some small size problems. RWLS also outperforms 3FNLS on all the combinatorial and STS problems both in terms of best solution values and computational times. It is interesting that 3FNLS’s solution values are generally better than those of Musliu although 3FNLS’s performance on USCPs is not previously investigated by its authors. However, when compared to RWLS, it is easy to see that our algorithm outperforms 3FNLS, for it can always obtain better solution values within shorter runtimes, especially on the larger instances with many more rows than columns, such as CYC11 and STS1215.

Combining the results of Tables 4 and 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.
In summary, 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 demonstrates the advantages of RWLS in solving USCP. 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 the three algorithms. For Musliu and 3FNLS, they both define a penalty function as their fitness functions. Specifically, Musliu’s fitness function is defined as the number of uncovered rows plus the cardinality of removed or added columns to be selected immediately in the next iteration. The main advantage of our tabu mechanism is that it is restricted, as well as not allowing the last two removed or added columns to be selected immediately in the next iteration. The best solutions previously reported by Musliu are obtained by exploiting different tabu factors for each individual instance, which is beyond our work; thus we only use the suggested parameters in our experiments. 3FNLS does not use any tabu mechanisms.

Second, the tabu mechanisms are different. Musliu defines a tabu list whose size is the product of a tabu factor and the cardinality of the initial solution. It stores the information of the columns which are removed or added in the previous 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 general for different instances, whereas the performance of Musliu heavily depends on fine tuning of the tabu factor for different kinds of instances. In fact, the best solutions previously reported by Musliu are obtained by exploiting different tabu factors for each individual instance, which is beyond our work; thus we only use the suggested parameters in our experiments.

Table 5

Computational results from Musliu, 3FNLS and RWLS on the random NRE to NRH, combinatorial and STS problems, obtained by running on the same Intel Core 3.25 GHz CPU with 4GB RAM under the Linux system.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Musliu</th>
<th>3FNLS</th>
<th>RWLS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Average</td>
<td>#Best</td>
</tr>
<tr>
<td>NRE1</td>
<td>17</td>
<td>17.2</td>
<td>8</td>
</tr>
<tr>
<td>NRE2</td>
<td>17</td>
<td>17.2</td>
<td>8</td>
</tr>
<tr>
<td>NRE3</td>
<td>17</td>
<td>17.2</td>
<td>8</td>
</tr>
<tr>
<td>NRE4</td>
<td>17</td>
<td>17.1</td>
<td>9</td>
</tr>
<tr>
<td>NRE5</td>
<td>17</td>
<td>17.4</td>
<td>6</td>
</tr>
<tr>
<td>NRF1</td>
<td>10</td>
<td>10.8</td>
<td>2</td>
</tr>
<tr>
<td>NRF2</td>
<td>10</td>
<td>10.8</td>
<td>2</td>
</tr>
<tr>
<td>NRF3</td>
<td>10</td>
<td>10.6</td>
<td>4</td>
</tr>
<tr>
<td>NRF4</td>
<td>10</td>
<td>10.9</td>
<td>1</td>
</tr>
<tr>
<td>NRF5</td>
<td>10</td>
<td>10.8</td>
<td>2</td>
</tr>
<tr>
<td>NRG1</td>
<td>62</td>
<td>62.6</td>
<td>4</td>
</tr>
<tr>
<td>NRG2</td>
<td>62</td>
<td>62.2</td>
<td>8</td>
</tr>
<tr>
<td>NRG3</td>
<td>63</td>
<td>63.0</td>
<td>10</td>
</tr>
<tr>
<td>NRG4</td>
<td>63</td>
<td>63.2</td>
<td>8</td>
</tr>
<tr>
<td>NRG5</td>
<td>62</td>
<td>63.1</td>
<td>1</td>
</tr>
<tr>
<td>NHR1</td>
<td>34</td>
<td>34.8</td>
<td>2</td>
</tr>
<tr>
<td>NHR2</td>
<td>35</td>
<td>35.0</td>
<td>10</td>
</tr>
<tr>
<td>NHR3</td>
<td>35</td>
<td>35.0</td>
<td>11</td>
</tr>
<tr>
<td>NHR4</td>
<td>34</td>
<td>34.9</td>
<td>1</td>
</tr>
<tr>
<td>NHR5</td>
<td>34</td>
<td>34.9</td>
<td>1</td>
</tr>
<tr>
<td>CLR10</td>
<td>25</td>
<td>25.1</td>
<td>9</td>
</tr>
<tr>
<td>CLR11</td>
<td>23</td>
<td>23.0</td>
<td>10</td>
</tr>
<tr>
<td>CLR12</td>
<td>23</td>
<td>23.0</td>
<td>10</td>
</tr>
<tr>
<td>CYC06</td>
<td>60</td>
<td>60.0</td>
<td>10</td>
</tr>
<tr>
<td>CYC07</td>
<td>144</td>
<td>144.0</td>
<td>10</td>
</tr>
<tr>
<td>CYC08</td>
<td>349</td>
<td>349.9</td>
<td>6</td>
</tr>
<tr>
<td>CYC09</td>
<td>809</td>
<td>813.0</td>
<td>3</td>
</tr>
<tr>
<td>CYC10</td>
<td>1894</td>
<td>1909.9</td>
<td>1</td>
</tr>
<tr>
<td>CYC11</td>
<td>4270</td>
<td>4271.1</td>
<td>1</td>
</tr>
<tr>
<td>STS243</td>
<td>198</td>
<td>201.7</td>
<td>10</td>
</tr>
<tr>
<td>STS405</td>
<td>343</td>
<td>345.0</td>
<td>4</td>
</tr>
<tr>
<td>STS429</td>
<td>649</td>
<td>649.8</td>
<td>4</td>
</tr>
<tr>
<td>STS1215</td>
<td>1119</td>
<td>1119.0</td>
<td>10</td>
</tr>
</tbody>
</table>

- Time limit for instance set NRE–NRH is set to 100 seconds.
- Time limit for CLR10–CLR13, CYC06–CYC08 and STS243 is set to 1000 seconds.
- Time limit for CYC09–CYC11 and STS243–STS729 instances is set to 10000 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 detecting the best.
- We emphasize our better solutions than Musliu and 3FNLS with boldface.
all rows a constant weight of 1 and thus may get stuck in a local optimum after certain iterations. Different from Musliu, 3FNLS adaptively adjusts the weights of rows during its local search, but unlike our weighting scheme, 3FNLS uses complex weighting adjust techniques that tend to consume more computation time.

Fourth, although 3FNLS has a sophisticated local search procedure, it adopts the information from solving Lagrangian Relaxation for prioritizing columns during the search. However, in some situations, information from Lagrangian Relaxation would become useless, such as for the STS problems (Yagiura et al., 2006).

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 the row weighting scheme as RWLS-1.

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 appears that the row weighting scheme helped RWLS to avoid being trapped in a local optimum and to continue exploring the solution space. Such phenomenon can be observed on other instances as well.

4.4. Evaluation RWLS on the railway crew scheduling instances

Table 7 gives the details of the railway crew scheduling instances from the OR-Library (Beasley, 1990b). 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 approach to deal with such instances is to use Lagrangian relaxation and its dual information to reduce the number of columns. This has been shown to be very effective for the non-unicost instances (Caprara et al., 1999; Ceria et al., 1998; Yagiura et al., 2006). We will incorporate such a technique into RWLS.

More precisely, we use the problem size reduction technique from 3FNLS (Yagiura et al., 2006), which is based on Lagrangian relaxation and uses the subgradient method to solve the core problem defined by Caprara et al. (1999). 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 Yagiura et al. (2006) 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.

In order to compare the results with yet another solver CPLEX, we run our algorithm and 3FNLS on an Intel Duo Core 2.4GHz CPU with 2 GB RAM machine, which has CPLEX12.5.3 installed. We set the maximum runtime for RWLS-R and 3FNLS to 100 seconds for instances RAIL507, RAIL516, and RAIL582, and 1000 seconds for RAIL2536, RAIL2586, RAIL4284, and RAIL4872 because of their larger sizes. For each instance, the results of ten independent runs of RWLS-R and 3FNLS are shown in Table 8. To our best knowledge, no results have ever 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, RAIL516, RAIL582), CPLEX is able to solve them to optimality in 1375.57 seconds, 6.49 seconds and 158.67 seconds, respectively. However, because the last four instances (RAIL2536, RAIL2586, RAIL4284, RAIL4872) are very large, CPLEX failed to produce solutions on these instances, even when more than ten thousands of seconds are given.

According to Table 8, RWLS-R 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 found by 3FNLS on RAIL582 is inferior to those found by CPLEX and RWLS-R. However, on the four larger ones (RAIL2536, RAIL2586, RAIL4284, RAIL4872), 3FNLS is able to obtain better solutions than RWLS-R. We note that the lower bounds found by 3FNLS on instances RAIL2536, RAIL2586, RAIL4284 and RAIL4872 are 363, 505, 579 and 857, respectively.

4.5. How good is cplex in solving uscp?

For non-unicost SCPs, it has been shown that the random instances from sets 4 to 6, A to D, NRE, NRF and NRG1 to NRG4 can be solved to optimality by CPLEX in reasonable time (Caprara et al., 2003).

Table 6 The effectiveness of the row weighting scheme on CYC11. RWLS is the original algorithm, while RWLS-1 is the algorithm without the adaptive weighting scheme. Results are obtained with random seed 11.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWLS</td>
<td>4799</td>
</tr>
<tr>
<td>RWLS-1</td>
<td>4799</td>
</tr>
</tbody>
</table>

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 candidate solution permanently
4: initialize 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 initializing the candidate
11: solution as the best found solution
12: end while
13: end function

Table 7
Details of the railway instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>m</th>
<th>n</th>
<th>Density</th>
<th>Instance</th>
<th>m</th>
<th>n</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAIL507</td>
<td>507</td>
<td>63,009</td>
<td>1.2</td>
<td>RAIL2586</td>
<td>2586</td>
<td>920,683</td>
<td>0.4</td>
</tr>
<tr>
<td>RAIL516</td>
<td>516</td>
<td>47,311</td>
<td>1.3</td>
<td>RAIL4284</td>
<td>4284</td>
<td>1,092,610</td>
<td>0.2</td>
</tr>
<tr>
<td>RAIL524</td>
<td>524</td>
<td>55,515</td>
<td>1.2</td>
<td>RAIL4872</td>
<td>4872</td>
<td>968,072</td>
<td>0.2</td>
</tr>
<tr>
<td>RAIL2536</td>
<td>2536</td>
<td>1,081,841</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* We turn this set of instances into unicost problems by ignoring the cost information.

Table 8
Test railway instances as unicost problems. Italic indicates the solution values that are worse than those produced by 3FNLS.

<table>
<thead>
<tr>
<th>Instance (RWLS-R)</th>
<th>3FNLS</th>
<th>CPLEX12.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Average</td>
<td>#Best</td>
</tr>
<tr>
<td>RAIL507</td>
<td>99</td>
<td>96.9</td>
</tr>
<tr>
<td>RAIL516</td>
<td>134</td>
<td>134.1</td>
</tr>
<tr>
<td>RAIL524</td>
<td>126</td>
<td>126.3</td>
</tr>
<tr>
<td>RAIL2536</td>
<td>381</td>
<td>381.6</td>
</tr>
<tr>
<td>RAIL2586</td>
<td>520</td>
<td>521.6</td>
</tr>
<tr>
<td>RAIL4284</td>
<td>597</td>
<td>599.4</td>
</tr>
<tr>
<td>RAIL4872</td>
<td>882</td>
<td>884.6</td>
</tr>
</tbody>
</table>

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

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

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

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

In this paper, we have introduced a new local search heuristic, named 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 integrated with RWLS, including an adaptive 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 when adding or removing a column (set). RWLS successfully hybridized these general strategies into its local search framework.

5. Conclusion and future work

In this paper, we have introduced a new local search heuristic, named 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 integrated with RWLS, including an adaptive 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 when adding or removing a column (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 (Beasley, 1990b) and Steiner triple systems (Fulkerson et al., 1974), 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 in terms of the best solutions found. It has improved 14 best known solutions in the literature. For the combinatorial instance CYC11, the best known solution value is improved 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 showed the effectiveness of RWLS in dealing with such large USCP instances by incorporating the problem size

2000). However, the USCP is generally considered to be harder to solve than non-unicost SCPs (Yelbay et al., 2015). In order to find out the difficulties of the random USCP instances from 4 to 6 and A to D, we apply CPLEX to these 45 instances.

In Table 9, 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, 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.

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 improves the BKSs.
reduction technique from Yagiura et al. (2006). This is the first time that the seven railway crew scheduling instances from the OR-Library 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 algorithm RWLS-R was outperformed by 3FNLS on the four larger railway crew scheduling instances. 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 to N. Musliu and M. Yagiura for providing their executable and source code of their algorithms, respectively. We appreciate the valuable comments of the anonymous reviewers, which have been very helpful in improving the quality and presentation of this paper.

This research work was supported by an EPSRC grant (No. EP/I010297/1), an EU FP7 IRSES grant (No. 247619) 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 and 61329302, 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