SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

June 2017, Vol. 60 062103:1–062103:14
doi: 10.1007/s11432-015-5377-8

A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity

Yiyuan WANG[1,2], Dantong OUYANG[1,2], Liming ZHANG[1,2] & Minghao YIN[1,3*]

[1]*Symbol Computation and Knowledge Engineer of Ministry of Education, Jilin University, Changchun* 130012*, China;*
[2]*College of Computer Science and Technology, Jilin University, Changchun* 130012*, China;*
[3]*School of Computer Science and Information Technology, Northeast Normal University, Changchun* 130117*, China*

Received March 27, 2016; accepted May 8, 2016; published online February 9, 2017

Abstract The unicost version of well-known set covering problem (SCP) is central to a wide variety of practical applications for which finding an optimal solution quickly is crucial. In this paper, we propose a new local search-based algorithm for the unicost SCP which follows the general framework of the popular stochastic local search with a particular focus on the hyperedge selection strategy and weight diversity strategy. Specifically, a strategy as called hyperedge configuration checking strategy is proposed here to avoid the search trajectory which leads to local optima. Additionally, a new weight diversity strategy is proposed for the diversification of search results, by changing the weight of both covered and uncovered vertices in the current solution. The experimental results show that our algorithm significantly outperforms the state-of-the-art heuristic algorithm by one to two orders of magnitudes on the 85 classical instances. Also, our algorithm improves the current optimal solutions of 11 instances.

Keywords unicost set covering problem, hyperedge configuration checking, local search, weight diversity strategy, hyperedge selection strategy

Citation Wang Y Y, Ouyang D T, Zhang L M, et al. A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. Sci China Inf Sci, 2017, 60(6): 062103, doi: 10.1007/s11432-015-5377-8

1 Introduction

The unicost set covering problem (SCP) is a prototypical NP-complete (Non-deterministic Polynomial) problem [1] which consists in finding the smallest size subset whose union equals the set of all vertices when the cost of each hyperedge is equal. Otherwise, the problem is called the non-unicost SCP or weight SCP. This problem is known to be equivalent to the classical integer linear program [2] and minimum hitting set [3, 4] and can be formulated as a minimum vertex cover [5, 6] in hypergraphs. The SCP plays a prominent role in various areas of artificial intelligence and has been widely used due to its theoretical interest and significance in numerous practical applications like crew scheduling problem [7], model-based diagnosis [8, 9], multiple-query optimization problem [10, 11], and so on.

* Corresponding author (email: ymh@nenu.edu.cn)

The most popular practical algorithms for SCP can be divided into two classes: exact algorithms and incomplete algorithms. For exact algorithms, a large amount of effort has been made, including [12–18]. The advantage of these algorithms is that they guarantee the optimality of the solutions they find, but they may fail to get a solution within reasonable time in conditions of large-scale SCP instances. Comparatively, incomplete algorithms cannot determine with certainty whether the returned solution is best. However, for large-scale instances incomplete algorithms are often applied without substantial computational effort and usually perform surprisingly effectively on some random or hard instances.

In the last decade, many advances in incomplete algorithms have been designed. In [19], a local search algorithm for SCP is proposed with three characteristics: 3-flip neighborhood, searching the infeasible region, and using the information from the Lagrangian relaxation to reduce the size of the problem. In [20], a reactive tabu search algorithm with variable clustering for the unicost SCP is proposed. Then, a dynamic primal-dual algorithm [21] applies the tabu search based on primal intensive scheme with a Lagrangian-based dual intensive scheme. Different relaxation heuristics are designed for SCP [22]. Applying Meta-RaPS [23] to solve SCP is simple and easy to code. In [24], a GRASP algorithm incorporating a local improvement procedure based on the heuristics is proposed to solve SCP. Normalization rules [25] demonstrate their superiority to the classical Chv*á*tal rule, especially when solving large-scale and real-world instances. The other type of solving SCP method includes ABC (Artificial Bee Colony) [26, 27], ACO (Ant Colony Optimization) [28, 29], GA (Genetic Algorithm) [30], and so on. In this paper, we highlight the work of electromagnetism metaheuristic method EM [31] which after generating a pool of solutions to create the initial population applies a fixed number of local search and movement iterations and also uses mutation to further escape from local optima. The results show that this method outperforms most previous heuristic approaches.

In this paper, we follow this line of research by proposing a new local search algorithm for unicost SCP. As is well known, the cycling problem is a key issue of local search algorithms. Accordingly, several strategies have been proposed to tackle this. Among these, tabu mechanism [32] and random walk strategy [33] may be the two most widely used methods. Recently, the configuration checking strategy [5, 34–39], proposed by Cai et al., was another successful method in avoiding cycling search. The main idea of configuration checking is to take the circumstances of the solution component (i.e., a basic element of the solution, which can be a variable or a vertex, for example) into account when selecting a solution component to change its value. The tabu mechanism forbids reversing the recent changes, where the forbidding strength is controlled by a parameter called tabu tenure. Usually, this tabu tenure is set to be 1 to avoid manual parameter adjustment. For a given solution component, if it is forbidden to select by the tabu mechanism, it is also forbidden by the configuration checking strategy, while its reverse is not necessarily true. The random walk strategy with a probability picks the solution component randomly and with another probability greedily makes the best possible local move. If this random probability′s value is large, for a candidate solution, this strategy could lead to a random move. In this sense, the advantage of configuration checking over random walk strategy and tabu mechanism is that its forbidding strength is between forbidding strengths of the tabu mechanism and the random walk strategy and appears neither too weak nor too strong. Up to now, the configuration checking strategy has been successfully used to solve minimum vertex cover, which is seen as one of the best known combinatorial optimization problems in the past decades. Also, different kinds of variants of configuration checking have also been proposed, as can be seen in [34–36, 38].

In this paper, to make configuration checking applicable to the unicost SCP, a novel variant of configuration checking, hyperedge configuration checking, is proposed first. This strategy remembers the circumstance of a hyperedge when it is removed or added and prevents it from being added into the candidate solution if its circumstance has not been changed since its last changing, where the circumstance of a hyperedge refers to truth value of all its neighbors.

Second, a new hyperedge selection and weight diversity strategies are proposed. We consider some special vertices covered by one or more hyperedges of the candidate solution in the phase of adding or removing hyperedges, distinguishing this property from previous vertex properties. We apply the hyperedge configuration checking and this vertex property to design a novel hyperedge selection strategy which allows us to make better use of the relevant information of candidate solution to avoid the recently visited solution and minimize search stagnation. Furthermore, we also design a new weight diversity strategy for the diversification of search results.

Finally, a novel stochastic local search algorithm for unicost SCP called uSLC is proposed by incorporating all the above strategies: hyperedge configuration checking strategy, hyperedge selection strategy, and weight diversity strategy. We use the benchmark instances of the OR-Library [40] and BHOSLIB [41] to conduct extensive experiments to compare our algorithm against a state-of-the-art SCP algorithm, EM [31]. In the 70 random instances where the number of hyperedges is larger than the number of vertices, the experiment shows that our algorithm uSLC outperforms EM by one to two orders of magnitudes in terms of run time. In particular, some instances have the maximum speed-up of 103 times. In the rest 10 combinatorial and logical instances where the number of hyperedges is smaller than the number of vertices, the speed-up is also 9 times on average. In the BHOSLIB benchmark, our algorithm uSLC could find all optimal solutions. It is also proved that our algorithm improves 11 current best solution values.

The remainder of this paper is organized as follows: Section 2 provides some necessary background knowledge. Then, Sections 3 and 4 present the hyperedge configuration checking, a novel hyperedge selection strategy, and a new weight diversity strategy. After that, Section 5 develops our algorithm uSLC. Then, Section 6 shows the experimental results on 85 instances and gives a detailed analysis. Finally, Section 7 gives the conclusion and future direction.

2 Preliminaries

We call an undirected hypergraph G=$(V,\ E)$ with a vertex set V $\{v_1, v_2, \ldots, v_m\}$ of m vertices and a hyperedge set E $\{e_1, e_2, \ldots, e_n\}$ of n hyperedges whose union equals the vertex set V. An undirected hyperedge containing just two vertices is a simple undirected edge. Each vertex v is associated with a weight $w(v)$, which is assigned to 1 initially. A vertex v is said to be incident with a hyperedge e when $v \in e$.

The aim of unicost SCP is to find the smallest hyperedges to cover all vertices. A candidate solution CS is a subset of hyperedges. A vertex is covered by a candidate solution if this vertex is incident with a hyperedge in this candidate solution. Two hyperedges are neighbors when they cover at least one vertex simultaneously. We use $N(e)$ to denote the set of neighbor hyperedges of a hyperedge e.

The previous local search algorithms never take circumstance information into consideration, but only select variables based on their information, such as score [42, 43] and age [44]. However, configuration checking combines the circumstance information with the traditional strategies on the selecting procedure.

Configuration checking is an effective method to handle the cycling problem, which has already been successfully used in minimum vertex cover problem [5, 45], SAT (Boolean Satisfiability Problem) [34–36, 46], and MaxSAT [39]. The idea of configuration checking prevents local search algorithms from encountering a previously visited scenario. Configuration checking remembers the circumstance information of each variable and forbids selecting variable x whose circumstance information has not been changed since the last time x was selected. In the context of SAT and MaxSAT, the configuration of a variable x refers to a vector consisting of Boolean values of x's all neighboring variables [35, 37, 39], while in [38] the configuration of a variable x refers to the states of all clauses in which x appears.

Definition 1. A variable x is defined as a configuration changed variable if and only if after the last time x was flipped, at least one neighbor y of x has been flipped.

An implementation of the configuration checking strategy is to apply a Boolean array $confchange$ for variables, where $confchange(x)$=1 means x is a configuration changed variable, and $confchange(x)$=0 on the contrary. During the local search procedure, the variables with $confchange(x)$=0 are forbidden to be selected in the greedy mode.

In the beginning of local search, for each variable x, the value of $confchange(x)$ is initialized as 1. Afterward, when selecting variable x, $confchange(x)$ is set to 0, and for each variable y of x's neighbor,

$confchange(y)$ is set to 1.

3 Hyperedge configuration checking

In recent years, in the field of local search algorithms, quite a lot of heuristic strategies have been put forward. Among them, configuration checking strategy stands out for its accessible and superior experimental results. Inspired by this idea, we adopt the configuration checking strategy to solve unicost SCP. In this paper, this strategy is not exactly the same as the previous configuration checking. Compared with the previous one which is used to solve the minimum vertex cover in simple graph, the application of configuration checking is extended into hypergraph to solve the unicost SCP.

In our new algorithm, we propose an alternative configuration checking strategy, called hyperedge configuration checking. At first, before giving some rules of hyperedge configuration checking, the concept of hyperedge configuration is provided. The hyperedge configuration of a hyperedge refers to the truth values of all its neighbor hyperedges. We give the formal definition of hyperedge configuration in unicost SCP as follows.

Definition 2. Given a hypergraph G=(V,E) and a candidate solution CS, the hyperedge configuration of e is a vector consisting of Boolean values of all hyperedges in $N(e)$ under this candidate solution CS.

Given a hypergraph G, the hypergraph configuration checking can be described as follows. When selecting a hyperedge and adding it into a candidate solution, for a hyperedge e, if the circumstance of e never changes, that is, the hyperedge configuration of e has not changed since e's last changing, then it should not be added. This strategy effectively prevents the local search algorithm from encountering a previous scenario.

We implement the hyperedge configuration checking strategy with a $|N|$-size Boolean array ECC whose size equals the number of hyperedges in unicost SCP. Each hyperedge keeps an ECC value, that is, $ECC[e]$=1 means this hyperedge could be selected in the next adding procedure, and on the contrary otherwise. During the search procedure, the ECC array is maintained as follows.

ECC-RULE1. At the start of local search, for each hyperedge e, $ECC[e]$ is initialized as true.

ECC-RULE2. When removing e from a candidate solution CS, $ECC[e]$ is reset to false, and for $e' \notin CS$ and $e' \in N(e)$, $ECC[e']$ is reset to true.

ECC-RULE3. When adding e into a candidate solution CS, for $e' \notin CS$ and $e' \in N(e)$, $ECC[e']$ is reset to true.

After removing one hyperedge, the ECC values of all neighbors of this hyperedge not in the CS are assigned as true and the ECC value of this removed hyperedge is assigned as false. When adding one hyperedge, the changing process is the same as the removing phase except the ECC value of the current hyperedge is set to false.

4 Hyperedge selection strategy and weight diversity strategy

During the local search process, the core operation is how to select a hyperedge to remove from or add into a candidate solution, which directly affects the efficiency and quality of the solution. For this reason, we use the vertex property to define the new score function and combine this function with the proposed hyperedge configuration checking to guide a novel hyperedge selection strategy scheme, which includes two rules: REMOVE-RULE and ADD-RULE. Also, after removing and adding hyperedges, we need to update the weights of some vertices to search the solution spaces as much as possible. Therefore, we also give the new weight diversity strategy to update the covered and uncovered vertices, respectively, by the candidate solution.

4.1 A novel hyperedge selection strategy

Some covered special vertices by the candidate solution are considered and we use the covered degree to define these special vertices, which is formally defined as follows.

Definition 3. For a hypergraph G and a candidate solution CS, the covered degree of a vertex v is defined as the number of hyperedges, including this vertex v, in the candidate solution. A vertex with the covered degree d is said to be d-covered vertex.

This definition is similar to the definition of satisfaction degree of clauses in SAT [34]. According to Definition 3, a d-covered vertex is covered by the candidate solution if $d > 0$, otherwise uncovered on the contrary. In the following, we give the formal definition of d-score and d-rscore.

Definition 4. For a hyperedge e and a candidate solution CS, d-score(e) is defined as the sum of weights of d-covered vertices that would become (d+1)-covered by adding this hyperedge into CS. Similarly, d-rscore(e) is defined as the sum of weights of (d+1)-covered vertices that would become d-covered vertices through removing this hyperedge from CS.

The previous SCP algorithms only use 0-score and 0-rscore to guide the local search. This simple score function is not enough information to exactly guide the local search. In this work, d-covered vertices are the most useful to solve the unicost SCP when adding hyperedges into the candidate solution, as these vertices may become uncovered easily in the next process for computing unicost SCP. Thus, it is beneficial when taking into account the transformations between d-covered and (d+1)-covered vertices in the candidate solution. Based on these considerations, we design a new score function, which is formally defined as follows.

Definition 5. For a hypergraph G=(V, E), a hyperedge e, and a candidate solution CS, the score function denoted by score is a function such that

$$\mathrm{score}(e) = \begin{cases} \displaystyle\sum_{d=0}^{m-1} d - \mathrm{score}(e), & e \notin CS, \\ \quad 0 - \mathrm{rscore}(e), & e \in CS, \end{cases} \tag{1}$$

where m is the number of vertices in the hypergraph G.

0-score and 0-rscore describe the greediness of adding and removing hyperedges at the current local search iteration. Also, d-score can forecast the subsequent greediness of adding hyperedges, which would have more d-covered and (d+1)-covered vertices. To combine with the current greediness and subsequent greediness, a new score function is a linear combination of d-score and (d+1)-score. For most instances, the score function is more effective to compute unicost SCP than only using 0-score and 0-rscore. This new function is simple, but is very useful to compute the unicost SCP with some forward predictions.

Our novel hyperedge selection strategy scheme applies hyperedge configuration checking and new score function to determine which hyperedge to be removed or added, which is maintained according to the following rules.

REMOVE-RULE. Picking one hyperedge e, which has the smallest score value, breaking ties in favor of the oldest one.

ADD-RULE. Picking one hyperedge e with $ECC[e]$=true, which has the greatest score value, breaking ties in favor of the oldest one.

When finding one hyperedge removed from CS, we try to find one hyperedge with the smallest score value, which means that this process should keep a candidate solution CS less uncovered hyperedges to not influence damaging the effectiveness of candidate solution. When selecting one hyperedge to add into CS, this strategy speeds up to find the solution in the search space with finding the hyperedge with the greatest score value. To avoid visiting the previous candidate solution, ECC value of this added hyperedge should be true.

4.2 A novel weight diversity strategy

In this subsection, we present a new weight diversity strategy, which plays an important role in our algorithm.

The proposed method works as follows. In each iteration, we use a new strategy to maintain vertex weighting diversity. After removing and adding hyperedges, vertex weighs of the uncovered vertices in a candidate solution CS are increased by the value of parameter p. On the other hand, vertex weights of covered vertices of a candidate solution are decreased by another parameter p. This new preventative strategy is defined as follows.

DIVER-RULE1. $w(v)$=$w(v)$+p, for uncovered vertices of the candidate solution.

DIVER-RULE2. If $w(v)> p$, then $w(v)$=$w(v)-p$, for covered vertices of the candidate solution.

We then use two rules to prioritize vertices for selecting the next move hyperedge. To the best of our knowledge, two proposed diversity rules use vertex weight for learning the previous visiting paths, which is different from the previous method. In [5], the edge weighting with forgetting in the NuMVC works as follows. In each iteration, edge weights of uncovered edges are increased by one. When the average weight achieves threshold, all weights are reduced to forget the earlier weighting decisions applying the formula $w(e)$=$w(e)*\alpha$.

5 A new local search algorithm for unicost SCP

We use the hyperedge selection and weight diversity strategies to improve the local search algorithm and propose a new algorithm called uSLC for the unicost SCP computation. We outline our algorithm uSLC, as described below.

Algorithm 1 uSLC algorithm

```
 1: initialize $ECC$ according to ECC-RULE 1
 2: initialize the weight of vertices and the score of hyperedges
 3: generate the initial candidate solution $CS$
 4: $CS^* \leftarrow CS$
 5: while stop criterion is not satisfied do
 6:   if there are no uncovered vertices then
 7:     $CS^* \leftarrow CS$
 8:     $e \leftarrow$ select one hyperedge from $CS$ according to REMOVE-RULE
 9:     $CS \leftarrow CS \setminus \{e\}$
10:     update $ECC$ according to ECC-RULE 2
11:     continue
12:   end if
13:   $e \leftarrow$ select one hyperedge from $CS$ according to REMOVE-RULE
14:   $CS \leftarrow CS \setminus \{e\}$
15:   update $ECC$ according to ECC-RULE 2
16:   select an uncovered vertex $v$ randomly
17:   $e \leftarrow$ select one hyperedge from $v$ according to ADD-RULE
18:   $CS \leftarrow CS \cup \{e\}$
19:   update $ECC$ according to ECC-RULE 3
20:   diversify vertex weight according to DIVER-RULE 1 and DIVER-RULE 2
21: end while
22: return $CS^*$
```

In the beginning, ECC is initialized as 1 for each hyperedge, which means all hyperedges are allowed to be added into a candidate solution CS. Also, the weight of each vertex is initialized as 1, and the score values of hyperedges are computed accordingly. The current candidate solution CS is generated with the greedy algorithm. The greedy initialization algorithm finds a solution for the unicost SCP by iteratively selecting the hyperedge that covers as many remaining uncovered vertices as possible until CS covers all vertices. Then, CS^* stores this solution. When we say the oldest one, we refer to the one with the minimum step number value, in which one hyperedge is picked most recently.

After the initialization, the loop (lines 5–21) is executed until a stop criterion is satisfied. During the search procedure, whenever a better solution is found, the best solution CS^* will be updated (line 7), which means uSLC will try to find the solution of smaller size. Then, our algorithm picks one hyperedge removed from CS according to REMOVE-RULE (line 8), so that our algorithm can go to search for a set cover of size $|C|$=$|C^*|-1$ (line 9). Then, our algorithm updates ECC values with ECC-RULE2 (line 10).

Table 1 Instances features

Set	Vertices	Hyperedges	Density	No. of instances	Problem type
SCP4	200	1000	2	10	Random
SCP5	200	2000	2	10	Random
SCP6	200	1000	5	5	Random
SCPA	300	3000	2	5	Random
SCPB	300	3000	5	5	Random
SCPC	400	4000	2	5	Random
SCPD	400	4000	5	5	Random
SCPE	50	500	20	5	Random
SCPNRE	500	5000	10	5	Random
SCPNRF	500	5000	20	5	Random
SCPNRG	1000	10000	2	5	Random
SCPNRH	1000	10000	5	5	Random

In each iteration of this loop, uSLC swaps two hyperedges in the following ways. If finding a hyperedge with the smallest score value, then this selected hyperedge is removed from a candidate solution, breaking ties in favor of the oldest hyperedge (line 13). After removing the picked hyperedge from the candidate solution (line 14), uSLC updates the ECC according to ECC-RULE2 (line 15). uSLC keeps track of the hyperedge last inserted into a candidate solution CS and prevents it from being added immediately. After removing a hyperedge, uSLC selects a vertex randomly from all uncovered vertices (line 16) and picks one of the hyperedges covering this vertex to add into a candidate solution as follows: if some hyperedges of this vertex satisfy $ECC[e]=1$, then uSLC selects the one with the highest score value, breaking ties in favor of the oldest hyperedge (line 17). Along with adding this hyperedge, the value of ECC is updated accordingly (line 19).

At the end of each iteration, uSLC updates the weights of vertices (line 20). Our algorithm uses the weight diversity strategy. Specifically, the weights of uncovered vertices are increased by p and then the weights of covered vertices are decreased by p.

6 Results

In this section, to demonstrate the effectiveness of our improving algorithm, we compare uSLC with EM [31], which is the best computing unicost SCP solver. To evaluate the performance of uSLC, we adopt two performance measurements, the run time (measured in CPU seconds) and the size of solution value.

The benchmarks. The instances of random, combination, and logic domains are selected from OR-Library [40] and BHOSLIB [41] generated randomly based on the model RB [46–50]. In total, 85 problem instances were selected. The characteristics of the instances selected are summarized in Tables 1 and 2. As previous studies are devoted to the unicost problem, the weights associated with the vertices appearing in each instance were ignored.

Implementation. Our algorithm uSLC is implemented in C and compiled by gcc with the −O2 option. For all instances, the random seed used in the execution of uSLC has been fixed to 0.

Computation platform. It is important to note that Naji-Azimi [31] carried out the computation experiments on a 1.7 GHz PC. Therefore, our experiment is also run at 1.7 GHz and with a 2 GB RAM under LINUX.

In our experiments, each instance terminates upon finding an optimal solution or reaching a given cutoff time which is set to 4000 s for each instance except for the challenging instance SCPCYC11, for which the cutoff time is set to 18000 s. We use the time required to reach the optimum as the main criterion for comparing our algorithm with EM. Our tested instances are divided into two groups, including the first group where the number of hyperedges is larger than the number of vertices and the second group where the number of hyperedges is smaller than the number of vertices.

Table 2 Details of the individual instances

Instance	Vertices	Hyperedges	Density	Problem type
scpclr10	511	210	12.3	Combinatorial
scpclr11	1023	330	12.4	Combinatorial
scpclr12	2047	495	12.5	Combinatorial
scpclr13	4095	715	12.5	Combinatorial
scpcyc06	240	192	2.1	Logical
scpcyc07	672	448	0.9	Logical
scpcyc08	1792	1024	0.4	Logical
scpcyc09	4608	2304	0.2	Logical
scpcyc10	11520	5120	0.8	Logical
scpcyc11	28160	11264	0.02	Logical
frb30-15-1	17827	450	14.1	Model RB
frb30-15-2	17874	450	14.2	Model RB
frb30-15-3	17809	450	14.1	Model RB
frb30-15-4	17831	450	14.1	Model RB
frb30-15-5	17794	450	14.1	Model RB

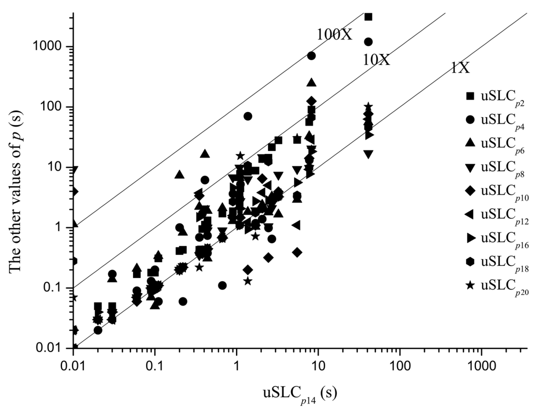

Figure 1 Scatter plot: uSLC$_{p14}$ versus the other values of parameter p.

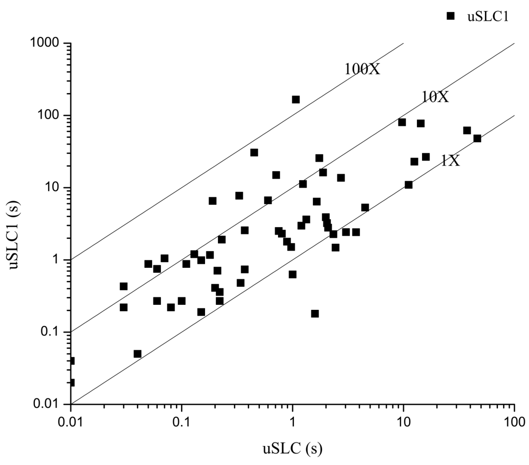

Figure 2 Scatter plot: uSLC versus uSLC1.

First, there is a parameter p in our algorithm uSLC , which specifies a novel weight diversity strategy. We investigate how uSLC performs with different settings of this parameter. The investigation is carried out on OR-Library. Figure 1 presents the performance of uSLC with various values of p. As we can see from Figure 1, the parameter p assigned to 14 yields relatively good performance for most cases and exhibits better robustness over the instances than other assignments. Therefore, the parameter p is assigned to 14.

We design four algorithms: uSLC1 without hyperedge configuration checking; uSLC2 without a novel hyperedge selection strategy, which would randomly select the removed and added hyperedges; uSLC3 without a novel weight diversity strategy, where uncovered vertices will be increased by one and when the average vertex weight is larger than threshold, we will update all vertex weights; and uSLC with a novel hyperedge selection strategy based on hyperedge configuration checking and a novel weight diversity strategy.

Figures 2–4 plot the run times of our approach uSLC versus the algorithms uSLC1, uSLC2, and uSLC3, along with the 1X, 10X, and 100X lines, clearly showing the superiority of the proposed strategies and hyperedge configuration checking. As shown, uSLC is clearly faster than uSLC1, uSLC2, and uSLC3.

To have a fair comparison, we should only consider the instances whose solution values are the same

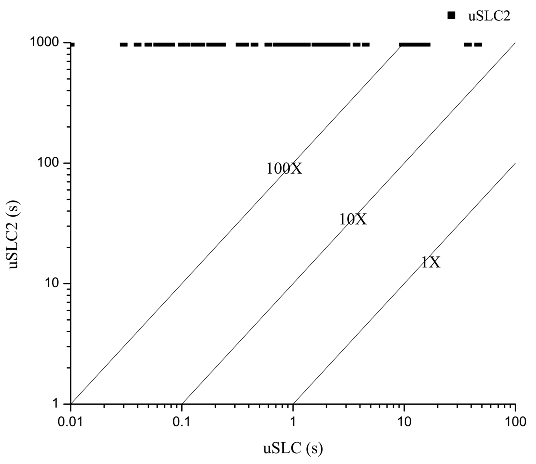

Figure 3 Scatter plot: uSLC versus uSLC2.

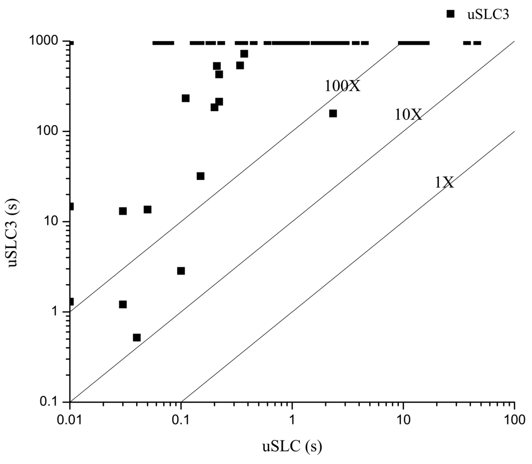

Figure 4 Scatter plot: uSLC versus uSLC3.

with that obtained by our algorithm uSLC and EM and this comparison is shown in the following tables giving the run time.

Tables 3 and 4 summarize the experimental results on the first group by our algorithm uSLC and the reference algorithm EM. For each algorithm, we show the solution value obtained and the run time with which the best solution is found.

As is obvious from Table 3, uSLC shows significant superiority to SCP4-6 and SCPA-E. It significantly outperforms EM in terms of run time. Particularly, for the instances of SCP5, the run time of our algorithm is almost 218 times and more than two orders of magnitudes less than that of EM. The experimental results on the SCPA, SCPB, SCPC, SCPD, and SCPE are presented in Table 3. For these 25 instances, the speed-up is 113 times on average. More importantly, the instance scpc1 shows a maximum speed-up of 1152 times on these five instance sets.

Table 4 shows the comparative results obtained on the SCPNRE, SCPNRF, SCPNRG, and SCPNRH. For SCPNRE, SCPNRF, and SCPNRG, the run time of uSLC is always much shorter than that of the corresponding EM. uSLC can find the same size of solution value as EM except scpnre1. However, for SCPNRH, EM consumes much less run time than our algorithm in solving scpnrh4.

EM improved four additional best known solutions (scp46, scp48, scp64, and scpd1) and first found the current best solution values of scpnrg1 and scpnrg2 with different parameter settings. Our proposed algorithm uSLC can also obtain the same solution values of these instances with fixed parameter settings. Therefore, the effectiveness and performance of uSLC are not subject to its parameters. More importantly, uSLC improves the current best solution values for 11 instances as reported in Table 5.

Table 6 lists the number of optimal or best solutions found by four state-of-the-art algorithms, uSLC, EM [31], GRASP [24], and R-Gr [51] with the cutoff time of 15000 s. uSLC finds all the best-known solutions for the 70 unicost random instances in the first group except for scpnre1.

For 10 instances in the second group, we use the new initialization method where the size of the candidate solution is initialized to the given upper bound, because with the previous initialization one cannot obtain a good initial solution. The experimental results of EM and uSLC on the SCPCLR and SCPCYC are presented in Table 7. For SCPCLR, uSLC dramatically outperforms EM once again except scpclr12. For SCPCYC, our algorithm runs faster than EM on the scpcyc6, scpcyc7, scpcyc8, scpcyc9, and scpcyc11, while the solution values of scpcyc10 by EM could be found more quickly than that of our algorithm. Note that in [31], through applying different parameters and seeds, EM obtains better results in cases of scpcyc9, scpcyc10, and scpcyc11. In fact, our algorithm can obtain the same results using different parameters. The performance of uSLC on the BHOSLIB is displayed in Table 8. Our algorithm uSLC finds all the optimal solutions for five instances and exhibits great performance on the BHOSLIB benchmark.

Table 3 Experiment results on the SCP4–6 and A–E

Instance	EM_sol	EM_t (s)	uSLC_sol	uSLC_t (s)
scp41	38	22.7	38	0.22
scp42	37	1.57	37	0.02
scp43	38	3	38	0.15
scp44	38	74.38	38	3.33
scp45	38	2.4	38	0.76
scp46	38	3.17	38	0.03
scp47	38	17	38	0.65
scp48	38	3.07	38	0.09
scp49	38	0.57	38	0.11
scp410	38	6.72	38	2.6
scp51	34	6.84	34	0.4
scp52	34	220.05	34	0.43
scp53	34	25.91	34	2.26
scp54	34	27.16	34	0.1
scp55	34	5.84	34	0.43
scp56	34	297.62	34	0.37
scp57	34	6.34	34	0.06
scp58	34	61.41	34	0.19
scp59	35	33.3	35	0.04
scp510	34	7.87	34	10.8
scp61	21	1.87	21	0.29
scp62	20	79.71	20	0.94
scp63	21	0.08	21	0.01
scp64	21	10.89	21	0.11
scp65	21	2.09	21	0.21
scpa1	39	28.85	39	1.71
scpa2	39	182.56	39	1.32
scpa3	39	140.21	39	0.61
scpa4	38	18.74	38	0.56
scpa5	38	7.81	38	0.43
scpb1	22	44.32	22	0.64
scpb2	22	7.39	22	0.15
scpb3	22	7.04	22	0.52
scpb4	22	29.04	22	0.79
scpb5	22	42.86	22	0.87
scpc1	43	921.96	43	1.06
scpc2	43	1023.79	43	1.99
scpc3	43	918.46	43	3.12
scpc4	43	28.49	43	8.22
scpc5	43	1007.09	43	7.47
scpd1	25	49.06	25	1.63
scpd2	25	18.76	25	2.34
scpd3	25	73.53	25	0.37
scpd4	25	50.56	25	5.47
scpd5	25	212.56	25	1.06
scpe1	5	0.01	5	0.01
scpe2	5	0.01	5	0.01
scpe3	5	0.01	5	0.01
scpe4	5	0.05	5	0.01
scpe5	5	0.02	5	0.01

Table 4 Experiment results on the SCPNR E–H

Instance	EM_sol	EM_t (s)	uSLC_sol	uSLC_t (s)
scpnre1	16	1305.72	17	20.78
scpnre2	17	22.61	17	0.03
scpnre3	17	50.67	17	14.16
scpnre4	17	43.31	17	2.94
scpnre5	17	10.78	17	0.02
scpnrf1	10	145.71	10	17.41
scpnrf2	10	1325.44	10	155.45
scpnrf3	10	1399.77	10	52.69
scpnrf4	10	120.2	10	40.24
scpnrf5	10	1651.14	10	8.1
scpnrg1	63	101.75	63	5.45
scpnrg2	63	127.87	63	2.35
scpnrg3	63	124.47	63	1
scpnrg4	63	117.15	63	0.51
scpnrg5	63	32.38	63	6.51
scpnrh1	34	755.72	34	452.92
scpnrh2	34	464.4	34	439.31
scpnrh3	34	1760.62	34	75.25
scpnrh4	34	227.73	34	809.13
scpnrh5	34	1912.47	34	561.47

Table 5 Improved solution value found by uSLC

Instance	EM_sol	uSLC_sol	uSLC_t (s)
scp46	37	37	2.41
scp48	37	37	2.4
scp64	20	20	1.1
scpa2	39	38	17.15
scpa4	38	37	265.59
scpd1	24	24	7.1
scpd2	25	24	475.14
scpd3	25	24	304.93
scpd4	25	24	269.9
scpd5	25	24	755.43
scpnrg1	62	61	266.17
scpnrg2	62	61	137.43
scpnrg3	63	61	172.58
scpnrg4	63	61	149.03
scpnrg5	63	61	275.61

Table 6 Number of optimal/best-known solutions

	SCP4-6	SCPA-E	SCPE-H	Total
uSLC	25/25	25/25	19/20	69/70
EM	25/25	19/25	15/20	59/70
GRASP	17/25	13/25	14/20	44/70
R-Gr	1/25	7/25	6/20	14/70

7 Summary and future work

This paper presents the local search algorithm named uSLC for solving the unicost SCP. We propose some new vertices weight rules to design the hyperedge selection scheme and weight diversity strategy to escape from current and previous local optimal solutions. Also, we use the hyperedge configuration checking

Table 7 Experiment results on the SCPCLR and SCPCYC

Instance	EM_sol	EM_t (s)	uSLC_sol	uSLC_t (s)
scpclr10	25	0.57	25	0.01
scpclr11	23	15.53	23	0.97
scpclr12	23	109.69	23	430.87
scpclr13	23	3539.45	23	1152.7
scpcyc6	60	0.08	60	0.03
scpcyc7	144	1.97	144	1.21
scpcyc8	344	303.4	344	30.95
scpcyc9	812	407.63	812	316.47
scpcyc10	1915	1892.06	1915	2645.95
scpcyc11	4272	12922.03	4272	11161.3

Table 8 Experiment results on the BHOSLIB

Instance	uSLC_sol	uSLC_t (s)
frb30-15-1	420	382.45
frb30-15-2	420	7.15
frb30-15-3	420	64.03
frb30-15-4	420	33.77
frb30-15-5	420	629.44

strategy to remember the relevant information of removed and added hyperedges. The experimental results show that uSLC performs essentially better than EM on most instances.

As for future work, given the success of uSLC in this paper, we consider that it may further improve the current algorithm of computing the unicost SCP if we combine other ideas such as the ACO [29]. Also, we would like to test our algorithm on other instances, including some hitting set instances. Envisioned research directions about the proposed strategies include applying the new score function to other local search algorithms and trying to find some other important properties of local search algorithms.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 61272208, 61370156, 61402196, 61503074, 61672261), Natural Science Foundation of Zhejiang Province (LY16F020004), and Program for New Century Excellent Talents in University (Grant No. NCET-13-0724). The authors of this paper express sincere gratitude to all the anonymous reviewers for their hard work.

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Karp R M. Reducibility among combinatorial problems. In: Complexity of Computer Computations. New York: Plenum Press, 1972. 85–103

2 Chakrabarty K. Test scheduling for core-based systems using mixed-integer linear programming. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2000, 19: 1163–1174

3 van Bevern R. Towards optimal and expressive kernelization for d-Hitting Set. Comput Comb, 2012, 7434: 121–132

4 Ausiello G, D'Atri A, Protasi M. Structure preserving reductions among convex optimization problems. J Comput Syst Sci, 1980, 21: 136–153

5 Cai S W, Su K L, Luo C, et al. NuMVC: An efficient local search algorithm for minimum vertex cover. J Artif Intell Res, 2014, 46: 687–716

6 Dinur I, Safra S. On the hardness of approximating minimum vertex cover. Ann Math, 2005, 162: 439–485

7 Caprara A, Fischetti M, Toth P. A heuristic method for the set covering problem. Oper Res, 1999, 47: 730–743

8 Reiter R. A theory of diagnosis from first principles. Artif Intell, 1987, 32: 57–95

9 Zhao X F, Ouyang D T. Improved algorithms for deriving all minimal conflict sets in model-based diagnosis. In: Proceedings of the Intelligent Computing 3rd International Conference on Advanced Intelligent Computing Theories and Applications. Berlin: Springer, 2007. 157–166

10 Angel E, Bampis E, Gourvès L. On the minimum hitting set of bundles problem. Theor Comput Sci, 2009, 410: 4534–4542

11 Sellis T K. Multiple-query optimization. ACM Trans Database Syst, 1988, 13: 23–52

12 Avella P, Boccia M, Vasilyev I. Computational experience with general cutting planes for the Set Covering problem. Oper Res Lett, 2009, 37: 16–20

13 Björklund P, Värbrand P, Yuan D. A column generation method for spatial TDMA scheduling in ad hoc networks. Ad Hoc Netw, 2004, 2: 405–418

14 Hemazro T D, Jaumard B, Marcotte O. A column generation and branch-and-cut algorithm for the channel assignment problem. Comput Oper Res, 2008, 35: 1204–1226

15 Caprara A, Toth P, Fischetti M. Algorithms for the set covering problem. Ann Oper Res, 2000, 98: 353–371

16 Pereira J, Averbakh I. The robust set covering problem with interval data. Ann Oper Res, 2013, 207: 217–235

17 Yelbay S B, Birbil I, Bülbül K. The set covering problem revisited: an empirical study of the value of dual information. J Ind Manag Optimiz, 2015, 11: 575–594

18 Galinier P, Hertz A. Solution techniques for the large set covering problem. Discret Appl Mathematics, 2007, 155: 312–326

19 Yagiura M, Kishida M, Ibaraki T. A 3-flip neighborhood local search for the set covering problem. Eur J Oper Res, 2006, 172: 472–499

20 Kinney G W, Barnes J W, Colletti B W. A reactive tabu search algorithm with variable clustering for the unicost set covering problem. Int J Oper Res, 2007, 2: 156–172

21 Caserta M. Tabu search-based metaheuristic algorithm for large-scale set covering problems. Metaheuristics Progress Complex Syst Opt, 2007, 39: 43–63

22 Umetani S, Yagiura M. Relaxation heuristics for the set covering problem. J Oper Res Soc Jpn, 2007, 50: 350–375

23 Lan G, DePuy G W, Whitehouse G E. An effective and simple heuristic for the set covering problem. Eur J Oper Res, 2007, 176: 1387–1403

24 Bautista J, Pereira J. A GRASP algorithm to solve the unicost set covering problem. Comput Oper Res, 2007, 34: 3162–3173

25 Ablanedo-Rosas J H, Rego C. Surrogate constraint normalization for the set covering problem. Eur J Oper Res, 2010, 205: 540–551

26 Sundar S, Singh A. A hybrid heuristic for the set covering problem. Oper Res, 2012, 12: 345–365

27 Crawford B, Soto R, Cuesta R, et al. Application of the artificial bee colony algorithm for solving the set covering problem. Sci World J, 2014, 2014: 189164

28 Mulati M H, Constantino A A. Ant-Line: a line-oriented ACO algorithm for the set covering problem. In: Proceedings of the IEEE International Conference of the Chilean Computer Science Society, Curico, 2011. 265–274

29 Ren Z G, Feng Z R, Ke L J, et al. New ideas for applying ant colony optimization to the set covering problem. Comput Ind Eng, 2010, 58: 774–784

30 Beasley J E, Chu P C. A genetic algorithm for the set covering problem. Eur J Oper Res, 1996, 94: 392–404

31 Naji-Azimi Z, Toth P, Galli L. An electromagnetism metaheuristic for the unicost set covering problem. Eur J Oper Res, 2010, 205: 290–300

32 Glover F. Tabu search-part I. ORSA J Comput, 1989, 1: 190–206

33 Selman B, Kautz H A, Cohen B. Noise strategies for improving local search. In: Proceedings of National Conference on Artificial Intelligence, Seattle, 1994. 337–343

34 Cai S W, Su K L. Comprehensive score: towards efficient local search for sat with long clauses. In: Proceedings of the International Joint Conference on Artificial Intelligence, Beijing, 2013. 489–495

35 Cai S W, Su K L. Local search with configuration checking for SAT. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence, Boca Raton, 2011. 59–66

36 Luo C, Cai S W, Wu W, et al. Double configuration checking in stochastic local search for satisfiability. In: Proceedings of National Conference on Artificial Intelligence, Quėbec, 2014. 2703–2709

37 Cai S W, Su K L. Local search for boolean satisfiability with configuration checking and subscore. Artif Intell, 2013, 204: 75–98

38 Luo C, Cai S W, Su K L, et al. Clause states based configuration checking in local search for satisfiability. IEEE Trans cybern, 2014, 45: 1014–1027

39 Luo C, Cai S W, Wu W, et al. CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans Comput, 2015, 64: 1830–1843

40 Beasley J E. OR-Library: distributing test problems by electronic mail. J Oper Res Soc, 1990, 41: 1069–1072

41 Xu K, Boussemart F, Hemery F, et al. A simple model to generate hard satisfiable instances. In: Proceedings of the International Joint Conference on Artificial Intelligence, Edinburgh, 2005. 337–342

42 Selman B, Levesque H J, Mitchell D G. A new method for solving hard satisfiability problems. In: Proceedings of National Conference on Artificial Intelligence, San Jose, 1992. 440–446

43 Li C M, Huang W Q. Diversification and determinism in local search for satisfiability. Lect Notes Comput Sci, 2005, 3569: 158–172

44 Gent I P, Walsh T. Towards an understanding of hill-climbing procedures for SAT. In: Proceedings of National Conference on Artificial Intelligence, Washington, 1993. 28–33

45 Cai S W, Su K L, Sattar A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif Intell, 2011, 175: 1672–1696

46 Xu K, Li W. Many hard examples in exact phase transitions. Theor Comput Sci, 2006, 355: 291–302

47 Xu K, Li W. Exact phase transitions in random constraint satisfaction problems. J Artif Intell Res, 2000, 12: 93–103

48 Xu K, Boussemart F, Hemery F, et al. Random constraint satisfaction: easy generation of hard (satisfiable) instances. Artif Intell, 2007, 171: 514–534

49 Gao J, Wang J N, Yin M H. Experimental analyses on phase transitions in compiling satisfiability problems. Sci China Inf Sci, 2015, 58: 032104

50 Huang P, Yin M H. An upper (lower) bound for Max (Min) CSP. Sci China Inf Sci, 2014, 57: 072109

51 Grossman T, Wool A. Computational experience with approximation algorithms for the set covering problem. Eur J Oper Res, 1997, 101: 81–92