IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 15, 2017, accepted July 1, 2017, date of publication July 7, 2017, date of current version July 24, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2724065

Nonuniform Neighborhood Sampling Based Simulated Annealing for the Directed Feedback Vertex Set Problem

ZHIPENG TANG, QILONG FENG, AND PING ZHONG, (Member, IEEE)
School of Information Science and Engineering, Central South University, Changsha 410083, China

Corresponding author: Ping Zhong (ping.zhong@csu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61402542.

ABSTRACT The feedback vertex set problem (FVSP), a combinatorial optimization problem, finds a set of vertices that intersect all cycles of the directed graph. One of the cutting-edge heuristics for this problem is a simulated annealing (SA)-based algorithm named the SA-FVSP. In this paper, we propose an improved variant of the SA-FVSP by applying the nonuniform neighborhood sampling (NNS), namely, the SA-FVSP-NNS. The NNS is a general strategy for improving the SA-based algorithm. Its basic idea is to prioritize the neighbors which are closer to the global optimum by assigning them with higher sampling probabilities. By doing this, these neighbors are more likely to be selected in the sampling process. To apply this general strategy to the SA-FVSP, we propose the concepts of the priority function and the sampling function, respectively. The priority function utilizes the known heuristic rules of the FVSP to estimate and score the quality of neighbors, while the sampling function converts the scores computed by the priority function to sampling probabilities, which can directly guide the NNS process. Experiments indicate that the SA-FVSP-NNS algorithm outperforms the SA-FVSP.

INDEX TERMS Feedback vertex set, simulated annealing, nonuniform neighborhood sampling.

I. INTRODUCTION

Given a graph, a *feedback vertex set* is a set of vertices which intersect all cycles of the graph. In other words, the given graph will be acyclic if all vertices in the feedback vertex set are removed [1]. Obviously, the feedback vertex set is not unique in a graph. We call the one with minimum cardinality the *minimum feedback vertex set*. The *feedback vertex set problem* (FVSP) is a combinatorial optimization problem which aims to find a minimum feedback vertex set for a given graph.

The FVSP is well-known in theoretical computer science for a long time because its decision version for a directed graph is among the first 21 problems proven to be NP-complete [2]. It also has many applications in various areas, such as Bayesian inference [3], Operating systems [4], VLSI chip design [5]. Because of the significance of the FVSP, enormous research has been done to tackle this problem, especially from the perspectives of exact algorithms [6], [7], approximation algorithms [8], [9], and parameterized algorithms [10], [11]. However, there are only a few studies dedicating to solve the problem from the standpoint of metaheuristics [1].

In 2013, Galinier *et al.* [1] proposed a metaheuristic based algorithm for the FVSP, i.e. the *Simulated Annealing for the Feedback Vertex Set Problem* (SA-FVSP), which is regarded as the best existing heuristic algorithm for the FVSP in the directed graph so far. The SA-FVSP is based on simulated annealing (SA), which is a typical local search based metaheuristic [12]. *Local search* is a search strategy which finds local optimums for a combinatorial optimization problem by moving from one solution to another in the search space using local changes [13]. In order to apply a local search based metaheuristic algorithm (such as the SA) to a combinatorial optimization problem, we need to define a *local search approach* and then incorporate it into the metaheuristic algorithm. A local search approach can be defined as a triplet including: the *search space* (i.e. the set of *configurations*), the *evaluation function* and the *move mechanism* [1]. Configurations are the representation of feasible solutions, and all feasible configurations constitute the search space. The evaluation function is the quantification of the optimization target. The move mechanism defines how to move from a solution to one of its neighboring solutions. A *move* of a reference configuration can transform this configuration to one

of its neighbors in accordance with the move mechanism. The SA-FVSP defines a practical local search approach based on the topological ordering property of the directed graph, and then incorporates it into the SA metaheuristic. The SA starts from an initial configuration. Then it moves from a configuration to another to forage for the global optimum. In each iteration, it randomly picks up a feasible move of the current configuration, and then determine whether to apply the move to the current configuration according to the *Metropolis criterion*.

In this paper, we introduce the *nonuniform neighborhood sampling* (NNS) strategy to the SA-FVSP, and propose an improved variant of this algorithm, named the *Simulated Annealing for the Feedback Vertex Set Problem using Nonuniform Neighborhood Sampling* (SA-FVSP-NNS). More specifically, in the SA-FVSP, a move is selected with uniform probabilities in each iteration of the SA before it is applied to the current configuration. This is effective but not efficient because such a *sampling process* (i.e. the process of choosing a feasible move) is totally blind. To mitigate the blindness, we find that a number of underlying heuristic rules for the FVSP has been observed by previous studies. Based on some of them, we can predict that some moves are more likely to lead the current configuration to a global optimum than others. Thus, when randomly selecting a feasible move in the sampling process, if we assign these superior moves with higher probabilities, and mutatis mutandis, assign those inferior moves with lower probabilities, the algorithm can definitely find out local optimums more rapidly and finally converge to better solutions. This idea is called NNS strategy [12]. Therefore, in the SA-FVSP-NNS, the concept of the priority function is introduced to estimate and score the quality of moves, and the concept of the sampling function is introduced to guide the assignment of the sampling probability in accordance with the output of the priority function. Based on these two functions, the newly proposed algorithm prioritizes those superior moves to a certain degree in the sampling process. Extensive experiments are conducted based on a recognized benchmark proposed by Pardalos et al. to show the superiority of the SA-FVSP-NNS when compared with the SA-FVSP.

The rest of the paper is organized as follows. A literature review is first presented in Section II. The SA-FVSP is introduced in Section III. The details of the SA-FVSP-NNS are described in Section IV. Section V presents the experimental results. Finally, some concluding remarks are made in Section VI.

II. RELATED WORK

Optimization problems are ubiquitous in various fields of computer science, such as theoretical computer science [14], wireless sensor networks [15] and machine learning [16]. The goal of the problem is to search from all possible solutions for the best solution. Combinatorial optimization problems are those optimization problems whose solution space is discrete. The feedback vertex set problem (FVSP) is a typical combinatorial optimization problem.

The FVSP can be divided into several interrelated problems, whose common goal is to find a set of vertices with minimum cardinality (or minimum weight) that intersect all the cycles of a given graph. Depending on whether the given graph is directed or undirected, we have the FVSP in the directed graph or in the undirected graph. Similarly, depending on whether the given graph is weighted or unweighted, we have the weighted or unweighted FVSP.

The FVSP is important in theoretical computer science because it is NP-complete in general graphs [2], [17]. It also has wide practical applications in various areas [3]–[5], [18]. As mentioned in Section I, many studies has been done on this problem, including two comprehensive literature reviews [5], [19]. However, there are only a few studies focusing on designing heuristics for this problem [1].

Currently, there are five main heuristics for the unweighted FVSP, i.e. the GRASP and its variant, the SA-FVSP, the NewkLS_FVS and the SALS. A chronological literature review is conducted below.

In 1998, Pardalos *et al.* [20] proposed the *Greedy Randomized Adaptive Search Procedure (GRASP) for the FVSP*, which is the first known metaheuristic based algorithm for the FVSP in literature. It can be used in both the directed and undirected graph. The GRASP is a randomized metaheuristic which can produce high quality solutions for a wide range of combinatorial optimization problems [21]. An iteration of the GRASP consists of two consecutive searching phases, i.e. the construction phase and the local search phase [13]. With regard to the GRASP for the FVSP, an initial solution, i.e. a valid feedback vertex set, is constructed in the construction phase; then the constructed initial solution is fine-tuned to achieve the local optimum solution in the local search phase. The GRASP is a multi-start algorithm, which repeats this iteration for many times to prevent itself from trapping into only one local optimum solution. In the construction phase, in order to improve the quality of the initial solution, the *greedy function* is introduced to estimate how likely a vertex belongs to the minimum feedback vertex set (More specifically, if a vertex is more likely to belong to a minimum feedback vertex set, the greedy function will assign it with a larger score), and those vertices with higher scores have higher probabilities to be chosen into the initial solution. As for the local search phase, as Pardalos *et al.* [20] pointed out, the main difficulty of this phase is verifying whether a graph is acyclic or not. Although the reduction rules proposed by Levy and Low [22] is used in the algorithm to mitigate the difficulty, this phase is still the performance bottleneck of the algorithm.

In 2006, Cai *et al.* [23] improved the performance of the GRASP for the FVSP by modifying the computation process of the original algorithm, adding newly observed reduction rules and introducing a new greedy function.

In 2013, two local search based heuristic algorithm for the FVSP were proposed. The first one is the aforementioned

SA-FVSP, which is based on the SA [1], [24]. The SA-FVSP ingeniously utilizes the topological ordering property of the directed acyclic graph to avoid the time-consuming process of verifying whether a graph is acyclic or not repetitively, which was considered inevitable before. Therefore, it can obviously outperform the GRASP based algorithms (It is worth noting that the experiments to compare the performance gap between the GRASP and the SA-FVSP is elaborately designed so that little bias can occur). However, because of the utilization of the topological ordering property, the SA-FVSP can only be used for the FVSP in the directed graph. The second algorithm is the *k-opt Local Search Algorithm with a Randomized Scheme for the Feedback Vertex Set* (NewkLS_FVS) [25], which mainly focuses on the FVSP in the undirected graph. The metaheuristic used in this algorithm, i.e. the k-opt local search algorithm, is different from the SA-FVSP and the GRASP for the FVSP, which changes more than one vertex from the current solution each time. Nevertheless, this algorithm does not propose an efficient method to eliminate or avoid the process of repetitively verifying whether a graph is acyclic or not, so it also wastes a large amount of time on this operation just like the GRASP based algorithms.

In 2014, Qin and Zhou [26] proposed the *Simulated Annealing Local Searching Protocol for the Undirected FVSP* (SALS) based on the work of Galinier et al.. This algorithm is only applicable to the undirected graph version of the FVSP. It modifies the microscopic search rules of the SA-FVSP by defining a constrained order for the vertices outside the current configuration.

As for the heuristics for the weighted FVSP, there are three existing algorithms, i.e. *XTS* [27], *ITS* [28] and *MA* [29]. Carrabs et al. have given an excellent review on the heuristic algorithm for the weighted FVSP (see [29]).

The SA-FVSP-NNS improves the performance of the SA-FVSP by introducing nonuniform neighborhood sampling (NNS), which is a general strategy for improving the performance of the SA metaheurisitc. In some cases, NNS is made to simplify the neighborhood generation procedure, while in others it is used to guide the search more effectively. In fact, the sampling process of some problems may even be naturally nonuniform [12]. There are already a number of studies using this strategy to accelerate the SA based algorithm [30]–[35].

Dowsland and Thompson [12] provided a comprehensive review on this topic. We notice that Lin *et al.*'s [33] solution to the truck and trailer routing problem and Ropke and Pisinger solution to a pick up and delivery problem [34] share some similarities to the ideas behind the proposed SA-FVSP-NNS. We all try to bias the sampling process to some neighbors. In Lin et al.'s work, the algorithm generates several neighbors and uses the best of them as the trial solution, instead of making the accept/reject decision on every single neighbor. Unlike Lin et al.'s solution that uses the best neighbor directly, Ropke et al. includes some kinds of randomization in order to maintain an appropriate level of flexibility for the SA. In the proposed SA-FVSP-NNS, we define the sampling function taking into consideration the requirements of both efficiency and flexibility. That is also the reason why we introduce a parameter to randomize selection in the sampling function.

III. SA-FVSP ALGORITHM

In this section, we briefly introduce the SA-FVSP, which was proposed by Galinier *et al.* [1]. We first define the local search approach that the SA-FVSP uses. Configurations and the evaluation function are presented in Section III-A, and the move mechanism is presented in Section III-B. Then, in Section III-C, we incorporate the local search approach into the SA metaheuristic and describe the pseudo-code of the SA-FVSP.

A. CONFIGURATIONS AND EVALUATION FUNCTION

A directed graph can be denoted as $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of arcs. A directed graph with no directed cycles is called a *directed acyclic graph* (DAG). Suppose V' is a subset of V, the subgraph of G induced by V', denoted as $G(V')$, is the graph whose vertex set is V' and whose arcs are those arcs which belong to E and have two endpoints in V', i.e. $G(V') = (V', E \cap (V' \times V'))$.

Since V' is a subset of V, it is obvious that the induced subgraph $G(V')$ is a DAG if and only if $V - V'$ is a feedback vertex set of the original directed graph G. Thus, the FVSP is equivalent to finding a vertex subset V' of maximum cardinality such that $G(V')$ is a DAG. In the SA-FVSP, we only consider this equivalent version of the FVSP.

A topological ordering of a directed graph is an ordering of its vertices such that the starting point of every arc is in front of the ending point of the arc in the ordering. An important property of the topological ordering is that every DAG has at least one feasible topological ordering, and conversely, the existence of one topological ordering implies that the graph is a DAG.

In the SA-FVSP, configurations are represented as the topological ordering of the acyclic induced subgraph of G, as formally defined in Def. 1:

Definition 1 (Configuration): In a sequence S, the number of elements is denoted as $|S|$, and the i-th element is denoted as $S[i]$, for every $i = 1, 2, \ldots, q$ where $q = |S|$. The sequence $S = (S[1], S[2], \ldots, S[q])$ is a configuration if:

1) $S[1], S[2], \ldots$ and $S[q]$ are vertices in the set V and are all different.
2) $\forall i, j, (1 \leq i < j \leq q) \rightarrow (S[j], S[i]) \notin E$ (precedence constraint)

In addition, we denote the set of the vertices that appear in the sequence S as $Dom(S) = \{S[1], S[2], \ldots, S[q]\}$. A vertex in V is described as *numbered* or *unnumbered* depending on whether it belongs to the set $Dom(S)$ or not.

We notice that every configuration S corresponds to an induced DAG $G(Dom(S))$ and that $V - Dom(S)$ is a corresponding feedback vertex set of G. Therefore, the optimization target of the FVSP can be formalized as finding a configuration S which can minimize $|V - Dom(S)|$. Def. 2 is the definition of the evaluation function.

Definition 2 (Evaluation Function): Assume S is a configuration in the search space, the evaluation function $f(S)$ can be defined as

$$f(S) = |V - Dom(S)| \tag{1}$$

B. MOVE MECHANISM

In the SA-FVSP, a move consists of two steps: inserting a new vertex into a particular position of the current configuration and then deleting the vertices which would now violate the precedence constraint. Def. 3 is the formal definition of the move.

Definition 3 (Move): Assume S is a configuration. A move $< v, i >$, where $v \in V - Dom(S)$ and $i \in \{1, 2, \ldots, |S| + 1\}$, includes two steps: inserting v just before $S[i]$, and then removing from S the elements of $CV_-(v, i) \cup CV_+(v, i)$, where:

- $CV_-(v, i) = \{S[j] \in S : j \geq i \text{ and } (S[j], v) \in E\}$
- $CV_+(v, i) = \{S[j] \in S : j < i \text{ and } (v, S[j]) \in E\}$

Adopting the move $< v, i >$ to S, the so-obtained configuration is obviously still valid and can be referred as $S \oplus < v, i >$. For a particular graph and one of its configuration S, all moves applicable to S consist of the *move list* for S.

Instead of using the move list directly, the SA-FVSP defines a refined move list, named the *candidate list*. The candidate list is a subset of the move list with high-quality moves inside. Def. 4 gives the definition of the high-quality move.

Definition 4 (High-Quality Move): Assume S is a reference configuration. Assume

- $I_-(v) = \{i : S[i] \in N_-(v)\}$ and $I_+(v) = \{i : S[i] \in N_+(v)\}$, where $N_-(v)$ and $N_+(v)$ are the sets of the in-coming and out-going neighbors of the vertex v in the graph respectively.
- $i_-(v) = max(I_-(v)) + 1$ if $I_-(v) \neq \phi$; otherwise, $i_-(v) = 1$;
- $i_+(v) = min(I_+(v))$ if $I_+(v) \neq \phi$; otherwise, $i_+(v) = |S| + 1$.

A high-quality move (v, i), where $v \in V - Dom(S)$ and $i \in \{i_-(v), i_+(v)\}$, includes two steps: inserting v just before $S[i]$, and then removing from S the elements of $CV_-(v, i) \cup CV_+(v, i)$.

Intuitively speaking, for a reference configuration S and a vertex $v \in V - Dom(S)$, the move list for S allows v to be inserted into arbitrary positions of S, but the candidate list for S restricts v to be inserted into only two positions, i.e. the position just after its numbered in-coming vertices (i.e. in the position before $S[i_-(v)]$) and the position just before its numbered out-going neighbors (i.e. in the position before $S[i_+(v)]$). This restriction can significantly reduce the number of vertices which violate the precedent constraint after inserting a new vertex. Thus, the number of moves in the candidate list is far less than that in the move list. Besides, Galinier et al. proved that both the candidate list and the move list define the same local optimum solutions [1].

The performance of the move $< v, i >$ can be defined as $\delta(v, i) = f(S \oplus < v, i >) - f(S)$, which means its influence on the evaluation function. Because a move will insert a new vertex and then delete all conflicting vertices, we also have that

$$\delta(v, i) = -1 + |CV_+(v, i)| + |CV_-(v, i)|. \tag{2}$$

C. ALGORITHM

Since an effective local search approach for the FVSP has been proposed, we now can incorporate it into a metaheuristic. The SA-FVSP is based on the SA, which is a typical local search based metaheuristic, proposed by Kirkpatrick et al. [36]. Algorithm 1 is the pseudo-code of the algorithm.

Algorithm 1 SA-FVSP

Input: a directed graph G; parameters T_0, α, $maxMv$, $maxFail$
Output: the best configuration S_* found by the algorithm

```
1  Set T := T_0; nbFail := 0; S := (); S_* := ();
2  repeat
3      Set nbMv := 0; failure := true;
4      repeat
5          Choose a move < v, i > uniformly randomly
           from the candidate list;
6          Evaluate Δ := δ(v, i);
7          if Δ ≤ 0 or exp(−Δ/T) ≥ random(0, 1) then
8              Apply move < v, i > to configuration S;
9              Set nbMv := nbMv + 1;
10             if f(S) < f(S_*) then
11                 Set S_* := S;
12                 Set failure := false;
13             end
14         end
15     until nbMv = maxMv;
16     if failure = true then
17         Set nbFail := nbFail + 1
18     else
19         Set nbFail := 0
20     end
21     Set T := T × α;
22 until nbFail = maxFail;
```

At the beginning, the temperature is initialized to a parameter T_0 and the configuration is initialized as an empty sequence. There are two loops in the SA-FVSP. The inner loop (lines 4-15) is corresponding to trials performed by the SA: uniformly randomly choosing a move from the candidate list (line 5, called the *sampling process*); then using the Metropolis criterion to determine whether the move is accepted or not (line 7); applying the move to the current configuration if the move is accepted (line 8-13). The outer loop (line 2-22) is corresponding to stages of the algorithm.

During a certain stage, SA-FVSP runs the inner loop repeatedly until *maxMvt* "actual" moves have been performed, and then, decreases the temperature by multiplying α (line 21). This process is repeated until *maxFail* stages have been run without any improvement of the score of the current best configuration S_* (i.e. the value computed by the evaluation function).

IV. SA-FVSP-NNS ALGORITHM

We notice that, in the sampling process of the SA-FVSP, every feasible move are chosen from the candidate list with the same probability. This sampling strategy is called *uniform neighborhood sampling*. Although adopting uniform probabilities is effective, it is not efficient. A natural idea is that if we can predict which moves are superior (i.e. the moves which can prompt the configuration to more rapidly converge to a minimum feedback vertex set) by utilizing some underlying heuristic rules of the FVSP, we can improve the performance of the SA-FVSP by prioritizing these superior moves in the sampling process (more specifically, when choosing a move from the candidate list, superior moves are assigned with larger sampling probabilities whereas inferior moves are assigned with smaller sampling probabilities). This idea is called *nonuniform neighborhood sampling* (NNS), which is a general technique for improving the performance of the SA based algorithm [12].

In order to apply NNS to the SA-FVSP, we have to answer two questions: how to estimate how superior a move is, and how to convert the estimation to the sampling probability, which can guide the sampling process directly. We describe the solutions to these two problems in the following Section IV-A and Section IV-B respectively. In Section IV-C, we combine the SA-FVSP with NNS strategy and propose the *Simulated Annealing for the Feedback Vertex Set Problem using Nonuniform Neighborhood Sampling* (SA-FVSP-NNS).

A. PRIORITY FUNCTION

When delving into combinatorial optimization problems, we can usually find out some useful properties and heuristic rules, some of which can be used to accelerate the optimization process of local search based metaheuristics. For the FVSP, we quantify this kind of properties and heuristic rules as a function, which can estimate and score how superior a move is (i.e. how likely a move makes the current configuration approach a global optimum), and use this function to guide the sampling process. We call this newly introduced function the *priority function*, which is defined in Def. 5.

Definition 5 (Priority Function): A priority function $\omega(S, \mu)$ is a two-variable function whose input is the current configuration S and a move μ in the candidate list for S, and whose output is a numeric value, i.e. the score of the move μ. Higher score means that the move μ is more likely to lead the current configuration to a global optimum. More formally, For two moves μ_1 and μ_2 in the candidate list for S, if μ_1 is more likely to lead S to a global optimum than μ_2, we have $\omega(S, \mu_1) \geq \omega(S, \mu_2)$.

Because the SA-FVSP tries to tackle this problem by finding the largest induced acyclic subgraph, rather than finding the minimum feedback vertex set directly (see Section III-A), the priority function should assign the vertices which are more likely to belong to the minimum feedback vertex set with smaller scores, and mutatis mutandis, assign the vertices which are less likely to belong to the minimum feedback vertex with larger scores. Note that the concept of the greedy function in the GRASP (see Section II) is just opposite to the concept of the priority function, because the greedy function gives larger scores to the promising vertices, rather than unpromising ones. Therefore, we can directly transform the greedy function to the priority function with minor modification.

So far, the best known greedy function $h(v)$ is proposed by Cai et al. [23]. Here is its mathematical formula:

$$h(v) = deg^-(v) + deg^+(v) - \lambda \times |deg^-(v) - deg^+(v)| \quad (3)$$

where v is a vertex in the graph, $deg^-(v)$ and $deg^+(v)$ are the indegree and the outdegree of the vertex respectively and λ is a parameter. Cai et al. suggested that the best performance can be achieved when $\lambda = 0.3$ [23].

This greedy function is based on two intuitive heuristic rules of the FVSP. The first is that if the indegree and outdegree of a vertex are large, this vertex is more likely to belong to the minimum feedback vertex set. The second is that if the difference between indegree and outdegree of a vertex is small, this vertex is more likely to belong to the minimum feedback vertex set. λ plays the role of balancing the effectiveness of these two heuristic rules.

Based on this greedy function, we define a feasible priority function ω_N for the SA-FVSP (see Def. 6).

Definition 6 (Concrete Priority Function): Suppose S is a reference configuration, $< m, i >$ is a move in the candidate list for S, we can define a concrete priority function for the SA-FVSP-NNS as

$$\omega_N(S, < m, i >) = -h(m) \quad (4)$$

Owing to the minus sign before $h(m)$, ω_N conforms to the definition of the priority function. We can also notice that ω_N is independent from the current configuration S and the position where the vertex insert. This is a useful property which will be used in Section IV-C.

B. SAMPLING FUNCTION

The priority function cannot be applied to the sampling process directly because it does not conform to the probability axiom [37]. Thus, we still need to convert the scores computed by the priority function to probabilities, which can be directly used in the sampling process. We call this kind of probabilities the *sampling probability*. In Def. 7, we define a new function similar to the priority function, named the sampling function. This function directly outputs the sampling probability of each move, rather than the score.

Definition 7 (Sampling Function): A sampling function $P(S, \mu)$ is a two-variable function whose input is the current

Algorithm 2 SA-FVSP-NNS

Input: a directed graph G; parameters T_0, α, $maxMv$, $maxFail$, θ
Output: the best configuration S_* found by the algorithm

Compute the values of ω_N for every vertex;
Compute the values of P_N based on ω_N and θ;
Set $T := T_0$; $nbFail := 0$; $S := ()$; $S_* := ()$;
repeat
  Set $nbMv := 0$; $failure := true$;
  repeat
    Choose a move $< v, i >$ from the candidate list with the probability P_N;
    Evaluate $\Delta := \delta(v, i)$;
    if $\Delta \leq 0$ *or* $exp(-\Delta/T) \geq random(0, 1)$ **then**
      Apply move $< v, i >$ to configuration S;
      Set $nbMv := nbMv + 1$;
      if $f(S) < f(S_*)$ **then**
        Set $S_* := S$;
        Set $failure := false$;
      end
    end
  until $nbMv = maxMv$;
  if $failure = true$ **then**
    Set $nbFail := nbFail + 1$
  else
    Set $nbFail := 0$
  end
  Set $T := T \times \alpha$;
until $nbFail = maxFail$;

configuration S and a move μ in the candidate list for S, and whose output is an sampling probability. Moves will be chosen with this probability in the sampling process. For a certain reference configuration S_0, $P(S_0, \mu)$ must satisfy the following two conditions:

1) $P'(\mu) = P(S_0, \mu)$ is a probability function, which obeys the probability axiom.
2) For two moves μ_1 and μ_2, if $\omega(S_0, \mu_1) \geq \omega(S_0, \mu_2)$, $P(S_0, \mu_1) \geq P(S_0, \mu_2)$.

The first condition guarantees that the output of the sampling function is a probability. The second condition implies that superior moves are assigned with higher probabilities.

The major problem is how to transform a priority function to a sampling function. Apparently, there are many feasible approaches. The approach we use is based on *group segmentation*, which is described as follows: group moves with similar priorities together and then assign moves in the same group with the same probability and moves in different groups with different probabilities. This method can assure that the superior moves are prioritized to some extent yet inferior moves do not lose opportunities to be a dark horse. Therefore, the SA still maintains a certain level of flexibility

TABLE 1. Parameters.

Parameter	Value	Meaning
T_0	0.6	Initial temperature
α	0.99	Parameter for decreasing the temperature
$maxMv$	$5 \times n$ [1]	Number of moves performed in each stage
$maxFail$	10	Number of stages without improvement
θ	5	Number of vertices in segmentation groups

like the scheme proposed by Ropke et al. (see Section II). Def. 8 gives the formal definition of this approach.

Definition 8 (A Concrete Sampling Function): For a given reference configuration S, suppose the candidate list for S is $\{< m_0, i_0 >, < m_1, i_1 >, \ldots, < m_{z-1}, i_{z-1} >\}$ where z is the number of moves in the candidate list. Without loss of generality, we further suppose $\omega(< m_0, i_0 >) \leq \omega(< m_1, i_1 >) \leq \ldots \leq \omega(< m_{z-1}, i_{z-1} >)$.

Suppose a *segmentation group* is a set of moves which have similar priorities. The move $< m_k, i_k >$, where $0 \leq k \leq z-1$, will be assigned into the $\rho(k)$-th segmentation group. Below is the formula of the function $\rho(k)$:

$$\rho(k) = \lceil \frac{k}{\theta} \rceil + 1 \tag{5}$$

where $1 \leq \theta \leq z$ is a parameter which controls the number of moves in each group.

A sampling function P_N based on the concept of the segmentation group can be defined as

$$P_N(S, < m_k, i_k >) = \frac{\rho(k)}{\sum_{j=0}^{z-1} \rho(j)} \tag{6}$$

Note that, if $\theta = 1$, all moves will be assigned with different probabilities, and conversely, if $\theta = z$, this sampling function will assign uniform probabilities to each move. Therefore, θ can be regarded as a parameter to control the influence degree of nonuniform neighborhood sampling.

Like the priority function ω_N, the sampling function P_N is also independent from the current configuration and the position where the vertex inserts. We can see the benefits of this property in the next section.

C. ALGORITHM

Since we already have a concrete sampling function, it is easy to modify the original SA-FVSP to support NNS. In fact, we just need to add the process of computing the sampling probability and modify the sampling process to choose a move from candidate list in accordance with the computed sampling probability.

As have been mentioned in Section IV-A and Section IV-B, the priority function ω_N and the sampling function P_N do not depend on the current configuration and the position where the vertex inserts. That is to say, these two functions only depend on the vertex. Therefore, we do not need to compute them in every iteration. Instead, we can just compute

[1] Henceforward, n and m denote the number of vertices and the number of the arcs in the benchmark graph respectively

TABLE 2. Experimental results obtained by running 1000 CPU time units.

Graph		SA-FVSP-NNS				SA-FVSP			
n	m	min	avg	max	dev	min	avg	max	dev
1000	30000	909	914.90	921	2.567	920	923.85	927	2.080
1000	25000	890	896.90	903	4.098	906	909.95	915	2.085
1000	20000	868	873.60	880	3.470	885	891.70	897	3.257
1000	15000	826	834.60	845	4.821	848	857.95	865	3.827
1000	10000	744	758.45	769	6.659	787	800.05	815	6.530
1000	5000	555	571.20	585	8.841	621	637.80	650	8.122
1000	4500	514	533.30	549	9.885	587	608.45	627	10.122
1000	4000	488	499.60	515	7.813	555	573.10	593	9.534
1000	3500	451	465.10	483	7.203	528	542.65	562	9.991
1000	3000	411	428.50	448	8.553	492	506.75	526	10.881
500	7000	399	406.05	409	2.479	415	417.75	422	2.142
500	6500	391	398.45	403	2.710	409	412.80	416	2.182
500	6000	388	392.35	396	2.007	405	407.60	410	1.428
500	5500	374	382.35	389	3.678	394	400.70	406	2.512
500	5000	362	370.30	376	3.348	386	390.75	396	2.773
500	3000	285	296.00	304	4.324	322	331.35	339	4.788
500	2500	250	260.45	266	4.566	288	299.45	313	6.289
500	2000	211	218.35	237	6.381	253	260.15	271	5.180
500	1500	147	157.95	169	6.144	186	202.80	216	7.257
500	1000	83	89.55	95	3.278	108	119.75	130	5.549
100	1400	75	76.55	78	0.805	78	79.45	81	0.669
100	1300	74	75.40	77	0.735	77	78.25	80	0.887
100	1200	71	73.10	75	1.179	76	76.95	78	0.669
100	1100	69	71.45	73	1.117	73	74.75	77	0.887
100	1000	67	70.30	72	1.269	71	73.20	74	0.980
100	600	51	53.10	55	1.338	56	58.70	60	1.187
100	500	46	47.30	49	0.900	49	52.85	55	1.459
100	400	35	37.60	39	1.114	42	43.85	46	1.152
100	300	26	28.30	30	1.054	31	33.85	37	1.711
100	200	10	11.80	14	0.980	14	16.05	18	1.322
50	900	39	39.65	40	0.477	40	40.55	41	0.487
50	800	38	39.30	40	0.640	39	39.65	40	0.477
50	700	36	36.90	38	0.539	37	38.50	40	0.742
50	600	35	35.90	37	0.625	36	37.20	38	0.600
50	500	31	32.85	34	0.572	33	34.75	36	0.698
50	300	22	23.85	25	0.792	25	26.75	28	1.135
50	250	20	21.25	22	0.698	23	24.15	25	0.726
50	200	16	17.70	19	0.714	18	20.20	22	0.871
50	150	10	10.75	12	0.622	12	13.90	16	0.995
50	100	3	3.00	3	0.000	3	3.50	4	0.500

them for every vertex in advance to accelerate the algorithm. Therefore, the process of computing the sampling probability can be added before the loops of the simulated annealing.

The newly proposed algorithm is called the *Simulated Annealing for the Feedback Vertex Set Problem using Nonuniform Neighborhood Sampling* (SA-FVSP-NNS). Algorithm 2 is its pseudo-code. We can notice that it shares nearly the same structure with the original SA-FVSP. The only two differences between the two algorithms are the newly added sampling probability computation (see Line 1 and 2 of Algorithm 2) and the sampling process (see Line 7 of Algorithm 2).

V. EXPERIMENTAL ANALYSIS

In this section, experiments are conducted to compare the performance of the SA-FVSP-NNS and the SA-FVSP. The detailed performance comparison between the SA-FVSP and the GRASP can be seen in the paper [1].

To achieve fairness, we do the following methods to control experimental variables:

1) We use the same computer to test these two algorithms, which is ThinkPad T420i with Intel Core i3-2350M CPU, 4GB RAM and Windows 7 operating system.
2) Different programmers usually have distinct programmatic preferences, which may cause that different implementations for the same algorithm have different performances. In order to eliminate this kind of error. We completely re-implement the SA-FVSP using C++. Then, because of the similar structure of the SA-FVSP and the SA-FVSP-NNS (see Section IV-C), we carefully implement the SA-FVSP-NNS by merely doing minor modifications on the code of the SA-FVSP so that algorithm implementation and code quality will not be a factor which bring about the differences of the experimental results.
3) We use a publicly recognized benchmark, which was generated by Pardalos *et al.* [20], to do experiments. The experiments of the GRASP for the FVSP and the SA-FVSP were also conducted on this

TABLE 3. Experimental results obtained by running 10000 CPU time units.

Graph		SA-FVSP-NNS				SA-FVSP			
n	m	min	avg	max	dev	min	avg	max	dev
1000	30000	906	910.40	914	2.035	917	919.65	922	1.590
1000	25000	886	892.15	896	2.574	902	906.00	908	1.612
1000	20000	855	866.90	872	3.419	879	885.15	889	2.903
1000	15000	819	826.75	831	2.718	842	850.80	857	3.140
1000	10000	734	742.50	751	4.945	776	788.15	794	5.237
1000	5000	504	517.35	526	5.977	589	599.00	611	6.648
1000	4500	464	474.65	487	6.452	545	559.25	573	7.049
1000	4000	406	416.60	428	6.192	489	510.10	521	7.848
1000	3500	353	365.80	377	6.361	440	455.05	466	6.607
1000	3000	297	305.95	320	5.608	374	394.40	413	8.622
500	7000	398	402.10	406	1.758	411	414.80	419	2.015
500	6500	390	393.85	396	1.652	405	409.50	413	2.247
500	6000	382	386.45	389	1.962	399	402.75	406	2.321
500	5500	372	376.00	379	1.844	391	394.30	397	1.900
500	5000	358	363.70	369	2.452	380	385.45	388	2.334
500	3000	281	286.40	293	2.764	314	320.70	326	3.303
500	2500	244	250.55	255	3.201	281	289.55	295	3.721
500	2000	199	205.15	211	3.167	239	246.20	253	3.295
500	1500	132	137.65	143	2.868	173	183.00	189	3.950
500	1000	63	66.95	71	2.037	85	92.95	100	3.905
100	1400	74	75.15	76	0.726	76	77.70	79	0.900
100	1300	73	74.20	75	0.748	75	76.95	79	0.865
100	1200	70	71.80	73	0.812	74	75.35	76	0.572
100	1100	68	69.70	71	0.843	71	73.20	74	0.927
100	1000	67	68.40	69	0.663	68	71.55	73	1.071
100	600	49	51.25	53	1.043	55	56.40	58	1.020
100	500	43	45.30	46	0.843	48	51.20	53	1.288
100	400	31	34.85	36	1.108	39	41.05	43	0.921
100	300	25	26.35	27	0.726	30	31.65	34	1.108
100	200	10	10.90	11	0.300	13	14.00	16	0.775
50	900	38	38.75	39	0.433	39	39.45	40	0.497
50	800	37	38.15	39	0.572	38	38.80	39	0.400
50	700	35	35.85	37	0.572	36	37.60	38	0.583
50	600	35	35.15	36	0.357	35	36.20	37	0.510
50	500	31	32.00	33	0.447	32	33.75	35	0.622
50	300	22	22.60	23	0.490	24	24.80	26	0.748
50	250	19	19.75	21	0.622	22	22.95	24	0.669
50	200	16	16.50	17	0.500	18	18.75	20	0.536
50	150	9	10.15	11	0.477	10	12.55	14	0.921
50	100	3	3.00	3	0.000	3	3.00	3	0.000

benchmark, so we cannot use some favorable instances to deliberately aggrandize the performance of the SA-FVSP-NNS.

In this benchmark, there are 40 graph in total, which can be grouped into four groups. Each group includes 10 graphs. These 10 graphs have the same number of vertices but different number of arcs. The graphs in these four groups contain 50, 100, 500 and 1000 vertices respectively. This benchmark can be downloaded from http://mauricio.resende.info/data/index.html.

4) We keep the parameter T_0, *maxMv*, α and *maxFail* the same in the experiments of the two algorithms. Table 1 presents the value of these parameters. Garlinier et al. have given a detailed explanation of how to select appropriate parameters and why selecting these values in their paper [1].
5) Every experiment runs 20 times with different random seeds in order to reduce errors.

In Section V-A, we will compare the experimental results of the two algorithms by limiting the running time. In Section V-B, we will fix the parameters, run the two algorithms without the running time limit and then compare their obtained results.

A. LIMITING RUNNING TIME

In this set of experiments, we compare the experimental results of the SA-FVSP-NNS and the SA-FVSP by limiting the running time to certain amounts, i.e. 1000 and 10000 units of the CPU time. Limiting the running time to 1000 CPU time units can help us compare the performance gap when given a short running time. Similarly, limiting the running time to 10000 CPU time units can help us compare the long-term performance gap. The parameter T_0, α, *maxMv* and θ used in these experiments are equal to the values in Table 1. *maxFail* is set to $+\infty$ in order to prevent the program from stopping before the time limit. All graphs in the benchmark are tested for 20 times.

TABLE 4. Experimental results obtained by running without the time limit.

Graph		SA-FVSP-NNS					SA-FVSP				
n	m	min	avg	max	dev	time	min	avg	max	dev	time
1000	30000	905	907.95	911	1.687	32475.20	916	918.95	922	1.499	24684.00
1000	25000	885	890.40	893	1.960	36808.30	900	903.75	908	2.447	35241.55
1000	20000	861	864.55	867	1.658	36599.35	880	883.10	886	1.997	34396.35
1000	15000	818	821.55	826	1.936	61148.30	839	847.60	853	3.292	49052.15
1000	10000	726	737.40	743	4.271	74956.80	778	783.90	791	3.345	62547.45
1000	5000	503	507.15	514	2.688	156732.00	580	587.95	596	3.853	130023.90
1000	4500	453	460.75	468	4.700	160055.60	541	547.75	555	3.858	135604.15
1000	4000	387	403.00	410	4.806	169096.10	485	496.65	507	5.677	145780.15
1000	3500	343	349.20	356	3.043	185211.95	429	441.10	450	5.186	168794.75
1000	3000	280	289.80	298	4.142	223634.75	367	376.10	384	4.134	190028.10
500	7000	395	401.15	405	2.651	14674.00	411	415.10	418	2.166	14175.45
500	6500	389	392.70	396	2.216	17659.35	405	407.70	412	1.819	14719.90
500	6000	379	384.95	391	2.924	18753.40	395	401.75	404	2.467	18643.75
500	5500	371	375.70	379	2.124	19059.65	390	395.00	398	1.949	17372.90
500	5000	358	362.25	366	1.946	25108.75	380	384.65	388	1.768	19132.60
500	3000	280	284.40	289	2.107	35185.80	312	318.40	322	2.417	29454.55
500	2500	243	248.85	254	2.937	38008.05	278	285.95	293	3.500	35284.35
500	2000	196	200.85	207	2.594	43872.65	237	244.05	250	3.667	38229.85
500	1500	125	132.55	138	3.154	62137.65	175	179.35	184	2.435	48745.40
500	1000	58	61.70	64	1.819	68717.75	86	89.00	92	1.732	64455.00
100	1400	74	76.80	78	0.927	1136.15	76	78.90	81	0.995	1224.85
100	1300	74	75.45	77	0.669	1393.10	76	78.35	80	0.853	1106.10
100	1200	71	73.65	75	0.853	1271.80	74	76.85	79	1.195	1092.70
100	1100	69	71.05	73	1.071	1337.75	72	74.85	77	1.236	1170.10
100	1000	68	70.30	72	0.954	1356.80	71	73.30	75	1.054	1304.65
100	600	50	52.40	54	1.020	1965.60	57	58.90	60	0.943	1655.35
100	500	44	46.60	49	1.158	2275.75	51	52.95	56	1.322	2132.70
100	400	34	36.50	38	1.118	2399.80	39	42.40	44	1.200	2207.05
100	300	25	27.55	29	0.921	2702.65	30	32.70	35	1.145	2849.10
100	200	11	11.45	12	0.497	3552.15	13	14.70	17	1.100	3084.85
50	900	39	40.00	41	0.447	376.90	40	41.05	42	0.669	340.45
50	800	39	39.80	41	0.510	361.45	40	40.50	41	0.500	361.95
50	700	36	37.05	38	0.740	471.50	38	39.10	40	0.625	405.65
50	600	35	36.40	38	0.663	469.95	37	37.85	39	0.572	420.45
50	500	32	33.20	34	0.600	484.50	34	35.30	37	0.843	544.90
50	300	23	24.25	25	0.698	708.40	25	26.90	28	0.700	690.85
50	250	20	21.35	23	0.853	742.00	23	24.65	26	0.963	657.70
50	200	17	17.95	19	0.669	711.50	18	20.20	22	1.030	875.20
50	150	10	11.30	12	0.640	831.25	13	14.05	16	0.921	788.75
50	100	3	3.00	3	0.000	875.90	3	3.75	5	0.622	902.10

Table 2 and Table 3 show the experimental results obtained by running the SA-FVSP-NNS algorithm and the SA-FVSP algorithm within 1000 and 10000 CPU time units respectively. These two tables share the same structure. Column 1 and 2 represent the number of vertices and the number of arcs in the benchmark graphs respectively. Column 3-6 show the experimental results of the SA-FVSP-NNS algorithm. Column 3 is the minimum cardinality of the feedback vertex set obtained during the 20 times' running. Column 4 is the average cardinality and Column 5 is the maximum cardinality. Column 6 shows the standard deviation. Column 7-10 are structurally the same as Column 3-6, but they show the experimental results of the SA-FVSP algorithm.

We can observe from the experimental results that, no matter the time limit is 1000 or 10000 CPU time units, the average cardinalities of the feedback vertex sets obtained by the SA-FVSP-NNS is significantly less than those obtained by the SA-FVSP. In most of cases, the minimum cardinalities obtained by the SA-FVSP are even larger than the maximum cardinalities obtained by the SA-FVSP-NNS. Thus, it conveys a strong evidence that the SA-FVSP-NNS can obtain better solutions in a given time limit than the SA-FVSP.

B. NO LIMITING RUNNING TIME

In this set of experiments, we compare the experimental results of the SA-FVSP-NNS and the SA-FVSP without the running time limit like Section V-A. Therefore, the two algorithms will run without disturbance and halt after performing *maxFail* unproductive stages. The parameter T_0, α, *maxMv*, *maxFail* and θ used in these experiments are equal to the values in Table 1. All graphs in the benchmark are tested for 20 times.

Table 4 shows the experimental results of these two algorithms when the running time limit is removed. Column 1 and 2 again represents the number of vertices and the number of arcs in the benchmark graphs respectively. Column 3-7 show the experimental results of the

SA-FVSP-NNS algorithm. Column 3 is the minimum cardinality of the feedback vertex set obtained during the 20 times' running. Column 4 is the average cardinality and Column 5 is the maximum cardinality. Column 6 shows the standard deviation. Column 7 shows the total running time of the algorithms (The running time of other ancillary operations is excluded). The measurement unit of this column is the CPU time unit. 1 second is equivalent to 1000 CPU time units. Column 8-12 are structurally the same as Column 3-7, but they show the experimental results of the SA-FVSP algorithm.

We can observe from the experimental results that, even if we do not limit the running time and let the algorithms halt on their own, the average cardinalities of the feedback vertex sets obtained by the SA-FVSP-NNS is also significantly less than those obtained by the SA-FVSP. Moreover, similar to the previous subsection, the minimum cardinalities obtained by the SA-FVSP are even larger than the maximum cardinalities obtained by the SA-FVSP-NNS in most cases. Therefore, the experimental results conveys a strong evidence that the SA-FVSP-NNS can acquire better solutions than the SA-FVSP when there is no time limit. It is worth mentioning that the SA-FVSP-NNS usually terminates later than the SA-FVSP. This means that the SA-FVSP-NNS is more capable of breaking the consecutive failed stages and jumping out of the traps of local optimum solutions, so it usually has more expansive search range in the search space than the SA-FVSP.

VI. CONCLUSION

The simulated annealing (SA) based SA-FVSP algorithm proposed by Garlinier et al. is one of the cutting-edge heuristic algorithm for the feedback vertex set problem (FVSP) in the directed graph. In this paper, we improved the performance of this algorithm by applying nonuniform neighborhood sampling (NNS) strategy and proposed the SA-FVSP-NNS algorithm. NNS is a general strategy to improve the performance of the SA metaheuristic. It assigns different probabilities to different moves so that the moves which are more likely to lead the current configuration to a global optimum are prioritized in the sampling process.

To improve the SA-FVSP using this strategy, we proposed the concept of the priority function ω and the sampling function P. The former one estimates and scores the quality of neighbors based on some practical heuristic rules of the FVSP; the latter one converts the scores computed by the priority function to sampling probabilities, which can then be directly used to guide the nonuniform sampling process. We also proposed a concrete priority function ω_N and a concrete sampling function P_N. The definition of ω_N is based on the concept of the greedy function in the GRASP, a previous heuristic algorithm for the FVSP. The definition of P_N is based on the idea of group segmentation, which can prioritize superior moves, and in the meantime, maintain randomization and flexibility of the SA to some extent.

Experiments were conducted to compare the performance of the SA-FVSP-NNS and the SA-FVSP. The experimental results indicated that the cardinality of the feedback vertex set found by the SA-FVSP-NNS is significantly smaller than the SA-FVSP no matter the running time is limited or not.

As for further studies, we want to explore how to modify the parameters in the SA-FVSP-NNS to achieve the best performance. In addition, we observe that the implementation of NNS in the SA-FVSP-NNS relies heavily on the heuristic rules used in the priority function. Thus, if we can find out more effective and accurate heuristic rules from the abundant existing research results on the FVSP and utilize them in the priority function, the performance of the SA-FVSP-NNS can be further improved. This will also be a subject of our future work.

ACKNOWLEDGMENT

The authors would like to thank the Editor-in-Chief, the Associate Editor, and the anonymous referees for their comments and suggestions. In addition, they do appreciate the help from Philippe Galinier, who is one of the authors of the SA-FVSP algorithm.

REFERENCES

[1] P. Galinier, E. Lemamou, and M. W. Bouzidi, "Applying local search to the feedback vertex set problem," *J. Heurist.*, vol. 19, no. 5, pp. 797–818, 2013.
[2] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. New York, NY, USA: Springer, 1972, pp. 85–103.
[3] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth, "Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference," *SIAM J. Comput.*, vol. 27, no. 4, pp. 942–959, 1998.
[4] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. Hoboken, NJ, USA: Wiley, 2015.
[5] P. Festa, P. M. Pardalos, and M. G. Resende, "Feedback set problems," in *Handbook of Combinatorial Optimization*. New York, NY, USA: Springer, 1999, pp. 209–258.
[6] T. Orenstein, Z. Kohavi, and I. Pomeranz, "An optimal algorithm for cycle breaking in directed graphs," *J. Electron. Test.*, vol. 7, no. 1, pp. 71–81, 1995.
[7] F. V. Fomin, S. Gaspers, and A. V. Pyatkin, "Finding a minimum feedback vertex set in time $O(1.7548^n)$," in *International Workshop on Parameterized and Exact Computation*. Berlin, Germany: Springer, 2006, pp. 184–191.
[8] G. Even, J. S. Naor, B. Schieber, and M. Sudan, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, no. 2, pp. 151–174, 1998.
[9] C. Demetrescu and I. Finocchi, "Combinatorial algorithms for feedback problems in directed graphs," *Inf. Process. Lett.*, vol. 86, no. 3, pp. 129–136, 2003.
[10] J. Chen, Y. Liu, S. Lu, B. O'sullivan, and I. Razgon, "A fixed-parameter algorithm for the directed feedback vertex set problem," *J. ACM*, vol. 55, no. 5, p. 21, 2008.
[11] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger, "Improved algorithms for feedback vertex set problems," *J. Comput. Syst. Sci.*, vol. 74, no. 7, pp. 1188–1198, 2008.
[12] K. A. Dowsland and J. M. Thompson, "Simulated annealing," in *Handbook Natural Computing*. Heidelberg, Germany: Springer, 2012, pp. 1623–1655.
[13] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.
[14] Q. Feng, J. Wang, and J. Chen, "Matching and weighted p_2-packing: Algorithms and kernels," *Theor. Comput. Sci.*, vol. 522, pp. 85–94, Feb. 2014.
[15] Y. Zhang, X. Sun, and B. Wang, "Efficient algorithm for k-barrier coverage based on integer linear programming," *China Commun.*, vol. 13, no. 7, pp. 16–23, Jul. 2016.

[16] B. Gu and V. S. Sheng, "A robust regularization path algorithm for ν-support vector classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 5, pp. 1241–1248, May 2017.
[17] M. R. Garey and D. S. Johnson, *Computers and Intractability*, vol. 29. New York, NY, USA: Freeman, 2002.
[18] Z. Tang, A. Liu, and C. Huang, "Social-aware data collection scheme through opportunistic communication in vehicular mobile networks," *IEEE Access*, vol. 4, pp. 6480–6502, 2016.
[19] J. Guo-Hong *et al.*, "Algorithms for feedback set problems: A survey," *Comput. Sci.*, vol. 38, no. 1, pp. 40–47, 2011.
[20] P. M. Pardalos, T. Qian, and M. G. Resende, "A greedy randomized adaptive search procedure for the feedback vertex set problem," *J. Combinat. Optim.*, vol. 2, no. 4, pp. 399–412, 1998.
[21] M. G. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures: Advances, hybridizations, and applications," in *Handbook Metaheuristics*. Springer, 2010, pp. 283–319.
[22] H. Levy and D. W. Low, "A contraction algorithm for finding small cycle cutsets," *J. Algorithms*, vol. 9, no. 4, pp. 470–493, 1988.
[23] X. Cai, J. Huang, and G. Jian, "A search algorithm for computing minimum feedback vertex set of a directed graph," *Comput. Eng.*, vol. 32, no. 4, pp. 67–69, 2006.
[24] Y. S. Kardam, "Metaheuristic techniques for solving optimization problems in graph theory," Ph.D. dissertation, Dept. Math., Dayalbagh Edu. Inst., Agra, India, Sep. 2014.
[25] Z. Zhang, A. Ye, X. Zhou, and Z. Shao, "An efficient local search for the feedback vertex set problem," *Algorithms*, vol. 6, no. 4, pp. 726–746, 2013.
[26] S.-M. Qin and H.-J. Zhou, "Solving the undirected feedback vertex set problem by local search," *Eur. Phys. J. B, Condensed Matter Complex Syst.*, vol. 87, no. 11, pp. 1–6, 2014.
[27] L. Brunetta, F. Maffioli, and M. Trubian, "Solving the feedback vertex set problem on undirected graphs," *Discrete Appl. Math.*, vol. 101, no. 1, pp. 37–51, 2000.
[28] F. Carrabs, R. Cerulli, M. Gentili, and G. Parlato, "A tabu search heuristic based on k-diamonds for the weighted feedback vertex set problem," in *Network Optimization*. Heidelberg, Germany: Springer, 2011, pp. 589–602.
[29] F. Carrabs, C. Cerrone, and R. Cerulli, "A memetic algorithm for the weighted feedback vertex set problem," *Network*, vol. 64, no. 4, pp. 339–356, 2014.
[30] D. T. Connolly, "An improved annealing scheme for the QAP," *Eur. J. Oper. Res.*, vol. 46, no. 1, pp. 93–100, 1990.
[31] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Ann. Oper. Res.*, vol. 41, no. 4, pp. 421–451, 1993.
[32] J. W. Greene and K. J. Supowit, "Simulated annealing without rejected moves," *IEEE Trans. Comput.-Aided Design Integr.*, vol. CAD-5, no. 1, pp. 221–228, Jan. 1986.
[33] S.-W. Lin, V. F. Yu, and S.-Y. Chou, "Solving the truck and trailer routing problem based on a simulated annealing heuristic," *Comput. Oper. Res.*, vol. 36, no. 5, pp. 1683–1692, 2009.
[34] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transp. Sci.*, vol. 40, no. 4, pp. 455–472, 2006.
[35] J. M. Thompson and K. A. Dowsland, "A robust simulated annealing based examination timetabling system," *Comput. Oper. Res.*, vol. 25, no. 7, pp. 637–648, 1998.
[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[37] K. L. Chung and F. AitSahlia, *Elementary Probability Theory: With Stochastic Processes and An Introduction to Mathematical Finance*. New York, NY, USA: Springer, 2012.

ZHIPENG TANG received the bachelor's degree in 2016. He is currently pursuing the master's degree with the School of Information Science and Engineering, Central South University, China. He has authored three SCI papers. His research interests include heuristic algorithms, the services-based network, and the crowd sensing networks.

QILONG FENG received the Ph.D. degree in computer science from Central South University, China, in 2010. His current research interests include computer algorithms and parameterized algorithms.

PING ZHONG (M'13) received the Ph.D. degree in communication engineering from Xiamen University, China, in 2011. She is currently a Lecturer with the Department of Computer Science and Technology, Central South University. Her research interests include machine learning, data mining, and networks protocol design. She is a member of the ACM, the CCF, and the IEICE.