sustainability

MDPI

Article

Multi-Objective Neighborhood Search Algorithm Based on Decomposition for Multi-Objective Minimum Weighted Vertex Cover Problem

Shuli Hu [1], Xiaoli Wu [1], Huan Liu [1], Yiyuan Wang [1], Ruizhi Li [2,*] and Minghao Yin [1,3,*]

[1] School of Computer Science and Information Technology, Northeast Normal University, Changchun 130117, China
[2] School of Management Science and Information Engineering, Jilin University of Finance and Economics, Changchun 130117, China
[3] Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun 130024, China
* Correspondence: lirz111@nenu.edu.cn (R.L.); ymh@nenu.edu.cn (M.Y.)

Received: 31 May 2019; Accepted: 24 June 2019; Published: 2 July 2019

Abstract: The multi-objective minimum weighted vertex cover problem aims to minimize the sum of different single type weights simultaneously. In this paper, we focus on the bi-objective minimum weighted vertex cover and propose a multi-objective algorithm integrating iterated neighborhood search with decomposition technique to solve this problem. Initially, we adopt the decomposition method to divide the multi-objective problem into several scalar optimization sub-problems. Meanwhile, to find more possible optimal solutions, we design a mixed score function according to the problem feature, which is applied in initializing procedure and neighborhood search. During the neighborhood search, three operators (*Add*, *Delete*, *Swap*) explore the search space effectively. We performed numerical experiments on many instances, and the results show the effectiveness of our new algorithm (combining decomposition and neighborhood search with mixed score) on several experimental metrics. We compared our experimental results with the classical multi-objective algorithm non-dominated sorting genetic algorithm II. It was obviously shown that our algorithm can provide much better results than the comparative algorithm considering the different metrics.

Keywords: multi-objective; decomposition; vertex cover; neighborhood search

1. Introduction

The task of optimization is to find the best possible solutions according to the fitness expressed by the problem objective function given some constraints. For the single-objective problems, there is only one goal to be optimized and the aim is to look for the best possible solution available or a good approximation of it. However, in the real world, there exist many problems that are associated with multiple metrics of performance, i.e. objectives. In such cases, we should optimize multiple goals simultaneously. Such problems are called Multi-objective Optimization Problems (MOP) [1,2]. For scientists and engineers, MOP is a significantly important research topic not only due to the features of such problems but also there are still many open problems in this area [3,4]. However, the fuzziness still lies on the fact that we cannot give the accepted definition of "optimum" for MOP as the single-objective problem. Therefore, we cannot compare the performance of different methods directly as the single-objective problem, which is the main bottleneck restricting the development of MOP [5].

In the past years, many surveys of multi-objective optimization techniques have been done from the mathematical programming perspective. For example, in [6], Pareto optimality is reviewed in connection with production theory, programming and economics. Lieberman [7]

analyzed 17 multi-objective mathematical programming methods and paid attention to their distinctive features and the issues of computational feasibility. Fishburen [8] reviewed the area of multi-objective evaluation theories and provided a perspective of how this fits in with the other areas. In [9], Evans discussed the advantages and disadvantages of the general approaches towards multi-objective mathematical programming and gave an overview of the specific solution techniques for multi-objective mathematical programming. The popular method to solve Constraint Optimization Problems (COP) is Branch and Bound (BB). Mini-Bucket Elimination (MBE) is a powerful mechanism for lower bound computation and has been extended to multi-objective optimization problems (MO-MBE) [10]. In [11], the authors embeded the MO-MBE in a multi-objective branch and bound search to compute the lower bound of the frontier of the problem. Actually, because of the limitations of branch and bound algorithms, it is hard to apply these methods to solve real-world problems. With the development in multi-objective optimization problem, it was recognized that evolutionary algorithms may be appropriate to solve the multi-objective optimization problem [12,13]. In the population, different individuals can explore the solutions in different directions concurrently, finally merging the effective information in the solution set. Moreover, evolutionary algorithms can handle complex problems and at the same time it have many features that are appropriate to solve multi-objective problems, such as discontinuities and disjoint spaces. Therefore, most present algorithms used to solve the multi-objective problems are based on the evolutionary methods; for example, in [14], an improved multi-objective evolutionary algorithm is proposed for the vehicle routing problem with time windows. Carvalho [15] presented a multi-objective evolutionary algorithm for optimal design of Yagi–Uda antennas. Meanwhile, to solve the multi-objective workflow scheduling in cloud, Zhu [16] also used the evolutionary approach to design the multi-objective algorithm. Recently, Zhang [17] proposed a popular method to solve MOP, that is, Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D). It decomposes the multi-objective optimization problem into a number of scalar optimization sub-problems. Because of the decomposition technique, MOEA/D is more efficient compared with the general evolutionary algorithm. Up to now, many researchers have paid attention to it and many works based on it are published [18,19].

Generally, the aim of MOP is not to find a solution but a solution set, which consists of all the non-dominated solutions. We say the solutions in this set are Pareto optimal solutions, which is the concept of Pareto optimal [1,2]. Now, we give the definition of "dominate". For a decision vector x, if for each objective there are always $f_i(x) \leq f_i(x')$ (for minimum multi-objective problem) and at least one objective such that $f_j(x) < f_j(x')$, we say that x dominates another decision vector x' denoted by $x \prec x'$. A decision vector x' is said to be Pareto optimal only if there exists no decision vector $x \in \Omega$ such that $x \prec x'$. Meanwhile, the Pareto front is the set of solutions that are all Pareto optimal. The shape of the curve may vary due to the different problem. Generally, the maximum multi-objective problem's curve is convex while the minimum multi-objective problem's curve is concave. However, the curve of the discrete multi-objective problem may be not as smooth as the continuous problem's curve.

As is known, minimum vertex cover is a core optimization problem, which has been widely applied in multiple areas such as industrial machine assignment, network security, and pattern recognition [20,21]. The Minimum Weighted Vertex Cover Problem (MWVCP) is a generalized version of Minimum Vertex Cover Problem (MVCP) [22,23]. There are also plentiful crafted algorithms to solve MWVCP. For example, Li [24] proposed an efficient local search framework for the MWVCP. However, in real-world problems, there are usually more than one decision and the vertex may have more than one associated weight. Subsequently, the Multi-objective Minimum Weighted Vertex Cover Problem (MMWVCP) arises. That is, each vertex is associated with several weights and we need to find a vertex subset minimizing every sum weight of different single type weights. Although many researchers have devoted their efforts toward the MWVCP and many algorithms have been successfully applied to solve it, few algorithms are proposed for MMWVCP. To our knowledge, the work in [11] is the first to analyze the lower bound of the frontier of MMWVCP and we are the first to propose the method

for solving MMWVCP. The researchers also pay attention to the different objectives for the vertex cover problem. In [25], the authors minimized the total weights of vertexes in the vertex cover and the LP-value at the same time, which is the value of optimal solution of the LP for the graph obtained by removing the vertexes in the vertex cover and the corresponding covered edges. Due to the different objectives, the method in [25] cannot be applied to solve the multi-objective minimum weighted vertex cover problem.

Our main contribution is proposing an effective algorithm to solve the multi-objective minimum weighted vertex cover. Firstly, we adopt the decomposition technique to decompose the multi-objective optimization problems into several scalar optimization sub-problems. Thus, most of the strategies having been used in the minimum weighted vertex cover problems algorithms can be introduced into our algorithm. We combine the degree score *Dscore* function with the weight score *Wscore* and design the score function *WDscore*, which can optimize the objective considering the number of vertexes and the sum of vertexes' weights simultaneously. The new score function is applied in the initializing phase and the iterated neighborhood search. To guarantee the quality and diversity of the population, in the initializing procedure, we construct a restricted candidate list considering the *WDscore* of the candidate vertexes. Meanwhile, for each individual in the population, we perform an iterated neighborhood search to improve the current solution. In the neighborhood search, for each step, we execute one operator choosing from three operators *Delete*, *Swap*, and *Add* according to the values of *WDscore*. Finally, a multi-objective algorithm, which is based on iterated neighborhood search and decomposition, is proposed to solve MMWVCP. For simplicity, we only focus on two objectives, i.e. the Bi-objective Minimum Weighted Vertex Cover Problem (BMWVCP).

In the next section, we introduce some background knowledge about multi-objective optimization problem and the technique decomposition. This allows us to know how the algorithm MOEA/D works. Then, in Section 3, our new algorithm including the important components is described in detail. We present the results of our experiment carried out on the benchmark instances in Section 4. Finally, we draw the conclusions and put forward some ideas for further research.

2. Preliminaries

2.1. Multi-Objective Minimum Weighted Vertex Cover

The definition of a multi-objective optimization problem (MOP) can be formulated as the following Equation (1):

$$\begin{aligned} &minimize\ F(x) = (f_1(x), f_2(x), f_3(x), ..., f_m(x))^T \\ &subject\ to\ x \in \Omega \end{aligned} \tag{1}$$

where Ω is the decision (variable) space and R^m is the objective space. The function $F : \Omega \to R^m$ consists of m real-valued objective functions.

Before introducing the multi-objective minimum weighted vertex cover problem, we give some relevant background knowledge. Given an undirected graph $G(V, E)$, where $V = \{1, 2, ..., n\}$ is the vertex set and $E = \{1, 2, ..., z\}$ is the edge set, the Vertex Cover Problem (VCP) consists of finding a subset of vertexes $Sub \subset V$ such that $\forall (u, v) \in E$, either $u \in Sub$ or $v \in Sub$. The Minimum Vertex Cover Problem (MVCP) is to find a vertex cover with minimum size. In the weighted version, each vertex v is associated with a weight $w(v)$ and the Minimum Weighted Vertex Cover Problem (MWVCP) is to find a vertex cover Sub with minimum $f(Sub) = \sum_{v \in Sub} w(v)$.

Based on the above notions, we give the definition of Multi-objective Minimum Weighted Vertex Cover Problem (MMWVCP) as follows:

Definition 1. *(Multi-objective Minimum Weighted Vertex Cover Problem, MMWVCP) Given an undirected weighted graph $G(V, E, w)$, each vertex v has m associated weight, $w_1(v), w_2(v), ..., w_m(v)$, and the task is to*

minimize the m objective functions $f_i(S) = \sum_{S[v]=1} w_i(v) (i = 1, 2, ..., m)$. *The MMWVCP can be formally defined as follows:*

$$\begin{aligned} minimize\ F(S) &= (f_1(S), ..., f_m(S))^T \\ f_i(S) &= \sum_{S[v]=1} w_i(v) \end{aligned} \tag{2}$$

where S is a binary vector and in our paper we call S as a solution or individual. When $S[v]$ *is equal to 1, we say the vertex v is in the solution S. Otherwise, we say the vertex v is not in the solution S. When a vertex is added to the solution, we set* $S[v] = 1$. *Similarly, we set* $S[v] = 0$ *when deleting a vertex from the solution S. For the BMWVCP, each vertex is associated with two weights and there are two objectives to optimize. Therefore, BMWVCP can be seen as a special case of MMWVCP for* $m = 2$.

2.2. Decomposition of Multi-Objective Optimization

At present, most algorithms used to solve multi-objective problems treat the MOP as a whole. Therefore, we need to consider every objective during searching the solution space. However, we can compare their objective functions directly and just optimize single objective if we associate each individual in the population with the particular scalar optimization problem. Meanwhile, many conventional and useful strategies which are initially applied in scalar optimization problems cannot be directly embedded in the non-decomposition algorithms. Hence, Zhang [17] proposed decomposition technique for the multi-objective problems which decomposes the initial MOP into several scalar optimization sub-problems. They combined the decomposition technique with the evolutionary algorithm to solve the multi-objective optimization problems and named the algorithm as MOEA/D. At each generation, the individuals in the population are the best solutions found so far considering the relevant sub-problems. Inspired by MOEA/D, we use the weight vector to decompose the multi-objective minimum weighted vertex cover problem into a number of single objective minimum weighted vertex cover sub-problems. In such a case, many approaches, which have been successfully used to solve minimum weighted vertex cover problem, can be applied in our algorithm. Now, we give the method of decomposition in our algorithm.

As mentioned above, given an objective vector $F(x) = (f_1(x), ..., f_m(x))^T$, we introduce an associated weight vector $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$, with $\sum_{i=1}^{m} \lambda_i = 1$ and $\lambda_i \geq 0$. Then, the optimal solution to the following scalar optimization problem:

$$g(x|\lambda) = \sum_{i=1}^{m} \lambda_i \times f_i(x) \tag{3}$$

When the weight vector λ is non-negative, Equation (3) has an optimal solution which is Pareto optimal to Equation (1). According to Equation (3), the multi-objective optimization problems can be decomposed into several sub-problems. For a different weight vector, there is always a sub-problem corresponding to it. Generally, the number of sub-problems is equal to the size of the population.

3. Multi-Objective Iterated Neighborhood Search Based on Decomposition Algorithm (MONSD)

In this section, we first present the framework of our new algorithm which is based on neighborhood search and decomposition. Subsequently, the three main components, namely the score function, initializing procedure and iterated neighborhood search, are described in detail.

3.1. Framework of MONSD

Our algorithm MONSD (Multi-objective Neighborhood Search based on Decomposition) is based on decomposition and it also integrates the neighborhood search according to the features of multi-objective minimum weighted vertex cover. The framework of MONSD is outlined in Algorithm 1. Firstly, we generate *Pop* (the size of population) evenly distributed weight vectors $\lambda = \{\lambda^1, \lambda^2, ..., \lambda^{Pop}\}$.

That is, the multi-objective minimum weighted vertex cover is decomposed into Pop sub-problems. The ith sub-problem is defined by Equation (3) with $\lambda = \lambda^i$. Then, MONSD calls the *Init_greedy* procedure to greedily produce Pop individuals to compose the current population P_c. The population P_{non} is used to store the non-dominated individuals found thus far. Subsequently, the algorithm executes a series of iterations until the stopping criterion is met. In one iteration, MONSD utilizes neighborhood search to improve the solutions in P_c. If the solution is improved, the signal boolean variable $isImproved$ is assigned as true. $isImproved$ is used to mark whether the solution gets the improvement after the neighborhood search or the perturbation process. Otherwise, we apply perturbation to the current solution. Namely, we randomly delete several vertexes and randomly add some vertexes to guarantee the solution is a vertex cover. The purpose of the perturbation is to increase the diversity and exploit more solution space. Meanwhile, if the solution is improved after perturbing, we also set the value $isImproved$ as true. At the end of iteration, if $isImproved$ is equal to true, the individual S_i will be replaced by S_i'. However, we always update the population P_{non} no matter whether the solution S_i is improved. Because we have decomposed the original multi-objective, the sub-problems cannot exactly indicate the objective. Even though the sub-problem's objective value is not much better, the original multi-objective values may be better than those solutions in the non-dominated population. In next subsection, we describe the components of MONSD in details.

Algorithm 1: The Multi-objective Neighborhood Search Algorithm based on Decomposition (MONSD).

Input: A weighted undirected graph $G(V, E)$
Output: A non-dominated solution set

```
uniformly initialize Pop weight vectors λ = {λ^1, λ^2, ..., λ^Pop};
greedily generate Pop individuals to compose the population P_c = {S_1, S_2, ..., S_Pop};
initialize P_non with non-dominated individuals in P_c;
while the stopping criterion is not met do
    for each solution S_i in P_c do
        isImproved ← false;
        S_i' ← NeighborhoodSearch(S_i);
        if g(S_i'|λ^i) < g(S_i|λ^i) then
            isImproved ← true ;
        else
            randomly delete some vertexes from S_i';
            randomly add several vertexes to S_i' until S_i' becomes a vertex cover;
            if g(S_i'|λ^i) < g(S_i|λ^i) then
                isImproved ← true ;
        if isImproved is true then
            S_i ← S_i';
        update P_non;
return P_non ;
```

3.2. The Score Function

After decomposing, each sub-problems can be seen as a single objective minimum weighted vertex cover problem. For the single objective minimum weighted vertex cover problem, many strategies have been successfully used to explore the search space. Among these strategies, the score function plays a critically important role in the whole algorithm which can guide the search to approach the possible best solutions. For the minimum vertex cover problem and the minimum weighted vertex cover problem, many kinds of score function have been applied in the algorithms. There are two

basic kinds of score function: degree function and weighted score function. In our algorithm, we define a new score function to adapt to the new problem which is applied in initializing procedure and neighborhood search. Our new score function mixes the *Dscore* based on the degree and *Wscore* based on the weight. We give these definitions as follows.

The nature of our *Dscore* function is similar to the degree of the vertex, thus we name it as *Dscore* and it is defined as Equation (4), where $e(v)$ is the edge set adjacent to the vertex v. If the vertex is not in the solution, the score value is used to evaluate how many edges would become covered from uncovered by adding the vertex to the solution. On the contrary, if the vertex is in the solution, the score value is used to evaluate how many edges would become uncovered by deleting the vertex from the solution and the value is non-negative. Accordingly, in the initializing phase, we initialize the score value of a vertex as the number of edges adjacent to it.

$$Dscore[v] = \begin{cases} |\{e|e \in e(v) \text{ and } e \text{ is only covered by } v\}|, \\ v \in solution \\ |\{e|e \in e(v) \text{ and } e \text{ is not covered}\}|, \\ v \notin solution \end{cases} \tag{4}$$

When one vertex is added to the solution, then its neighbor vertexes' *Dscore* values are decreased by one. On the contrary, when one vertex is deleted from the solution, its neighbor vertexes' *Dscore* values are increased by one. Considering the values of *Dscore*, we prefer the vertexes with bigger *Dscore*, which makes the number of vertexes in the solution smaller. On the contrary, when we select vertexes to remove from the solution, we prefer the vertexes with smaller *Dscore*, which makes fewer edges become uncovered.

Because we have decomposed the multi-objective minimum weighted vertex cover into several sub-problems, we define a weighted score function to evaluate the effect of one vertex to the single sub-problems. For one sub-problem, the weighted score can be defined as Equation (5). Once the vector λ is fixed, the *Wscore* of each vertex is assigned and constant during the neighborhood search. Considering the values of *Wscore*, when we select the vertexes to add to solution, we prefer to the vertexes with smaller *Wscore*, which makes objective value smaller. On the contrary, when we select vertexes to remove from the solution we prefer to the vertexes with bigger *Wscore*.

$$Wscore[v] = \sum_{i=1}^{m} \lambda_i w_i(v) \tag{5}$$

From the above definitions of *Dscore* and *Wscore*, we can see that the *Dscore* motivates the algorithm to use fewer vertexes to optimize the objective while the *Wscore* motivates the algorithm to select vertexes with smaller weight to optimize the objective. Based on the definitions, we design a more effective score function combining the *Dscore* with *Wscore*. It is defined in Equation (6).

$$WDscore[v] = \frac{Dscore[v]}{Wscore[v]} \tag{6}$$

According to the definition of *WDscore*, to effectively optimize the objective, we select the vertexes with bigger *WDscore* to add to the solution and select the vertexes with smaller to remove from solution. The experimental results show the *WDscore* is more effective than *Dscore* and *Wscore*.

We use an example to illustrate the difference of the score functions in Figure 1. There are two small graphs with five vertexes and four edges and v_3 is with different weight in the two graphs. Such graphs may be a component of a bigger graph. Each vertex is associated with a tuple (*Wscore*, *Dscore*, *WDscore*). Because *Wscore* keeps constant, once the λ is fixed, we assign an integer ranging in [20, 120] to each vertex as its *Wscore*. The triple tuple of each vertex is initialized as shown in Figure 1. For the graph in Figure 1a, v_3 is with bigger degree and bigger weight. If we use the *Dscore*, we will choose v_3 to cover the four edges. However, the weight of v_3 is greater than the sum weight of the remaining

four vertexes, and we should add v_1, v_2, v_4 and v_5 to the solution to cover the edges. If we use the *Wscore* or *WDscore*, we can change the situation. Initially, the *WDscore* of v_5 is smaller than v_3's, but the *WDscore* of v_5 will decrease after inserting v_1, v_2 or v_4. Similarly, the *Wscore* will also lose the effectiveness in the second graph. We can see that *WDscore* can perform well in the two situations. However, *Dscore* and *Wscore* will lose the advantage in the corresponding situation.

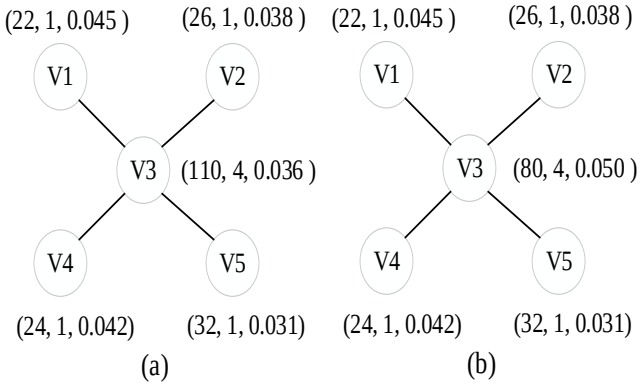

Figure 1. The example illustrating the difference of score functions. (**a**) the *Dscore* loses the effectiveness (**b**) the *Wscore* loses the effectiveness.

3.3. The Procedure of Initializing

The initial population is essentially important in algorithms, which can lead the algorithm to different search directions. On the one hand, we should guarantee the population quality which will seriously influence the final quality of Pareto-front. On the other hand, to find more non-dominated solutions, we should guarantee the population diversity. To balance the quality and diversity, we adopt constructing RCL (short for Restricted Candidate List), which has been widely used in local search and approximate algorithms [26,27]. The procedure of initializing population called *Init_greedy* is outlined in Algorithm 2.

Algorithm 2: Init_greedy(Pop).

Input: the size of population Pop
Output: a population P_c

```
IndiNum ← 0;
while IndiNum < Pop do
    Initialize the score value of each vertex ;
    Initialize the uncovered edge link ;
    while the S_IndiNum is not a vertex cover do
        maxscore ← the maximum WDscore value of the vertexes not in S_IndiNum;
        minscore ← the minimum WDscore value of the vertexes not in S_IndiNum;
        limitscore ← miniscore + α × (maxscore − miniscore);
        for the vertex v not in S_IndiNum do
            if WDscore[v] ≥ limitscore then
                add v to RCL ;
        randomly select one vertex from RCL to add to S_IndiNum;
        update the WDscore value and the uncover edge link ;
    if S_IndiNum is different from the other individuals in P_c then
        IndiNum++;
        Insert S_IndiNum into P_c ;
```

After executing $Init\_greedy$, there will be Pop individuals to compose the population P_c. When $Init\_greedy$ generates one individual, it firstly assigns the score value of every vertex according to Equations (4)–(6). Then, all edges will be added to uncovered edges link. Subsequently, the procedure will construct RCL. First, it finds the maximum score ($maxscore$) and minimum score ($minscore$) values of the vertexes not in $S_{IndiNum}$. Second, $Init\_greedy$ sets the limited score as $minscore + \alpha \times (maxscore - minscore)$. Only if the vertex's score is greater than the limited score do we add it to RCL. The parameter α is used to control the greediness of the procedure. A greater α indicates a greater greediness. Then, $Init\_greedy$ randomly selects one vertex from RCL and adds it to $S_{IndiNum}$. At the end of one iteration, we judge whether the $S_{IndiNum}$ is a vertex cover and different from the individuals existing in P_c. The procedure doe not stop until it generates Pop different individuals. In Algorithm 2, we present the initialize procedure for the version MONSD with $WDscore$ (MONSD^{WDscore}). For the version MONSD with $Dscore$ or $Wscore$, we should update the score function.

3.4. The Iterated Neighborhood Search

The iterated neighborhood search is an important component in our algorithm, which focuses on one solution and deeply explores it to find more promising solutions. A neighbor is typically defined by a move operator (mv). Namely, a move operator (mv) is applied to a given solution S and S is accordingly transformed to a neighbor of S', denoted by $S' = S \oplus mv$. Then, the neighborhood N of S can be defined as: $N(S) = \{S' | S' \leftarrow S \oplus mv, mv \text{ is an operator}\}$. In our neighborhood search, there are three complementary neighborhoods defined by three basic operators. These neighborhoods are explored in a combined manner employing a rule that selects the most favorable neighbor solution with the best score.

The score function $WDscore$ is used to evaluate the benefit if we delete or add one vertex considering the $Dscore$ and $Wscore$. Based on the $WDscore$ mechanism, we design three move operators (denoted by $Delete$, Add and $Swap$). The three operators are defined as:

$Delete(S)$: The $Delete$ operator consists in deleting one vertex from S while guaranteeing S is always a vertex cover. If the operator can find an appropriate vertex, it will return the vertex number, otherwise it will return -1 indicating the operator is not feasible. Based on the nature of $WDscore$, we can find that such vertexes are with $WDscore$ equal to 0. If there are more than one vertex with $WDscoer$ equal to 0, we always select the vertex with the biggest score ($Wscore$) to ensure the quality of the neighbors of S. The neighborhood defined by the $Delete$ operator is given by $N_1(S) = \{S' | S' \leftarrow S \oplus Delete(v)\}$. The $Delete$ operator is described in Algorithm 3.

Algorithm 3: Delete(S).

Input: a solution S
Output: a vertex v to be delete

```
DelVtx ← −1;
maxweight ← −1;
for each vertex v in S do
    if WDscore[v] = 0 and maxweight < Wscore[v] then
        maxweight ← Wscore[v];
        DelVtx ← v;
return DelVtx;
```

$Swap(S)$: The $Swap$ operator consists in exchanging one vertex ($DelVtx$) in S with one vertex ($AddVtex$) not in S while guarantee S always is a vertex cover, which is outlined in Algorithm 4. The vertex pair with smaller $Wscore[AddVtx] - Wscore[DelVtx]$ has a higher priority. If the value is less than 0, it indicates the objective value will be improved if we execute the swap operator. Because the swap operator does not change the number of vertexes in the solution, we should consider the

benefit brought for the objective by swapping two vertexes directly. Therefore, we use the $Wscore$ to evaluate the vertex pair. Similar to the operator $Delete$, there may not be an appropriate vertex pair. In this case, it will return $(-1, -1)$, which indicates that the operator is not feasible; otherwise, it will return a pair of vertex number. The neighborhood defined by the move operator is given by $N_2(S) = \{S' | S' \leftarrow S \oplus Swap(v, u)\}$.

Algorithm 4: Swap(S).

Input: a solution S
Output: a pair vertexes (u, v)

```
DelVtx ← −1, AddVtx ← −1;
minscore ← 0;
Queue edge_v ;
for each vertex v in S do
    while !edge_v.empty() do
        edge_v.pop();
    for each edge e adjacent to v do
        if e is only covered by v then
            edge_v.push(e);
    for each vertex u not in S do
        if minscore > Wscore[u] − Wscore[v] and u can cover all the edges in edge_v then
            minscore ← Wscore[u] − Wscore[v];
            DelVtx ← v;
            AddVtx ← u;
return (DelVtx, AddVtx);
```

$Add(S)$: The Add operator consists adding a vertex not in S. Before adding a vertex, the S is already a vertex cover, thus we can judge that S is always a vertex cover after adding one vertex. Because $Delete$ and $Swap$ are greedy operators, we randomly select a vertex from the vertex set not in S to increase diversity. It returns the selected vertex number. The neighborhood defined by the Add operator is given by $N_3(S) = \{S' | S' \leftarrow S \oplus Add(v)\}$. The Add operator is described in Algorithm 5.

Algorithm 5: Add(S).

Input: a solution S
Output: a vertex v to be added

```
AddVtx ← −1;
CandidateVtx ← ϕ;
for each vertex v do
    if v not in S then then
        CandidateVtx ← CandidateVtx ∪ {v};
AddVtx ← randomly select from CandidateVtx;
return AddVtx;
```

Based on the definitions of three operators, we can see the $Delete$ and $Swap$ move always make the single objective value decrease, while Add always increases the single objective value. Considering the above features of three operators, we create a combined neighborhood. To effectively optimize the objective, when we choose the operator between $Delete$ and $Swap$, we directly compare the value of $Wscore[DelVtx_1]$ and $Wscore[DelVtx_2] - Wscore[AddVtx_2]$ and choose the operator with the bigger value. The iterated neighborhood search is described in Algorithm 6.

Algorithm 6: NeighborhoodSearch(S).

Input: a solution S
Output: an optimal solution S^*

for *each vertex* v **do**
  compute $Dscore[v]$, $Wscore[v]$, $WDscore[v]$;
$Inter \leftarrow 0$, $S^* \leftarrow S$;
while $Inter < MaxStep$ **do**
  $DelVtx_1 \leftarrow Delete(S)$;
  $AddVtx_1 \leftarrow Add(S)$;
  $(DelVtx_2, AddVtx_2) \leftarrow Swap(S)$;
  if $DelVtx_1 \neq -1$ *and* $(DelVtx_2, AddVtx_2) = (-1, -1)$ **then**
    $S \leftarrow S \backslash \{DelVtx_1\}$;
  else if $DelVtx_1 = -1$ *and* $(DelVtx_2, AddVtx_2) \neq (-1, -1)$ **then**
    $S \leftarrow S \backslash \{DelVtx_2\} \cup \{AddVtx_2\}$;
  else if $DelVtx_1 \neq -1$ *and* $(DelVtx_2, AddVtx_2) \neq (-1, -1)$ **then**
    if $Wscore[DelVtx_1] > Wscore[DelVtx_2] - Wscore[AddVtx_2]$ **then**
      $S \leftarrow S \backslash \{DelVtx_1\}$;
    else
      $S \leftarrow S \backslash \{DelVtx_2\} \cup \{AddVtx_2\}$;
  else
    $S \leftarrow S \cup \{AddVtx_1\}$;
  if $g(S^*|\lambda^{S^*}) > g(S|\lambda^S)$ **then**
    $S^* \leftarrow S$;
return S^* ;

We take an example to explain how the iterated neighborhood search works. As shown in Figure 2, the graph is with 12 vertexes and 19 edges. Each vertex is associated with a triple tuple with the same meaning as in Figure 1. The black vertexes are in the solution while the white ones are not in the solution. At each iteration, one operator is performed and the score values are updated subsequently. By executing the procedure of initializing, an initial solution is generated and the objective $f(S)$ is 224 (the order of adding vertexes can be $v_{11}, v_9, v_8, v_5, v_1, v_3, v_{10}, v_{12}$). At the first iteration, we perform the swap operator and the objective becomes 223. At the second iteration, there is no feasible *Delete* or *Swap* operator. We randomly select a vertex to add, such as the vertex v_4. After updating the score values, we find that two vertexes can be deleted v_1 and v_3. v_3 is deleted due to the bigger weight. At the next iteration, there is no feasible *Delete* or *Swap* operator again and v_{10} is selected to add to the solution. Then, v_7 and v_{10} are deleted in the following two iterations. Due to the limitation of space, we show them in one iteration here. After deleting v_7 and v_{10}, the objective is 212, which is the minimum objective over the graph.

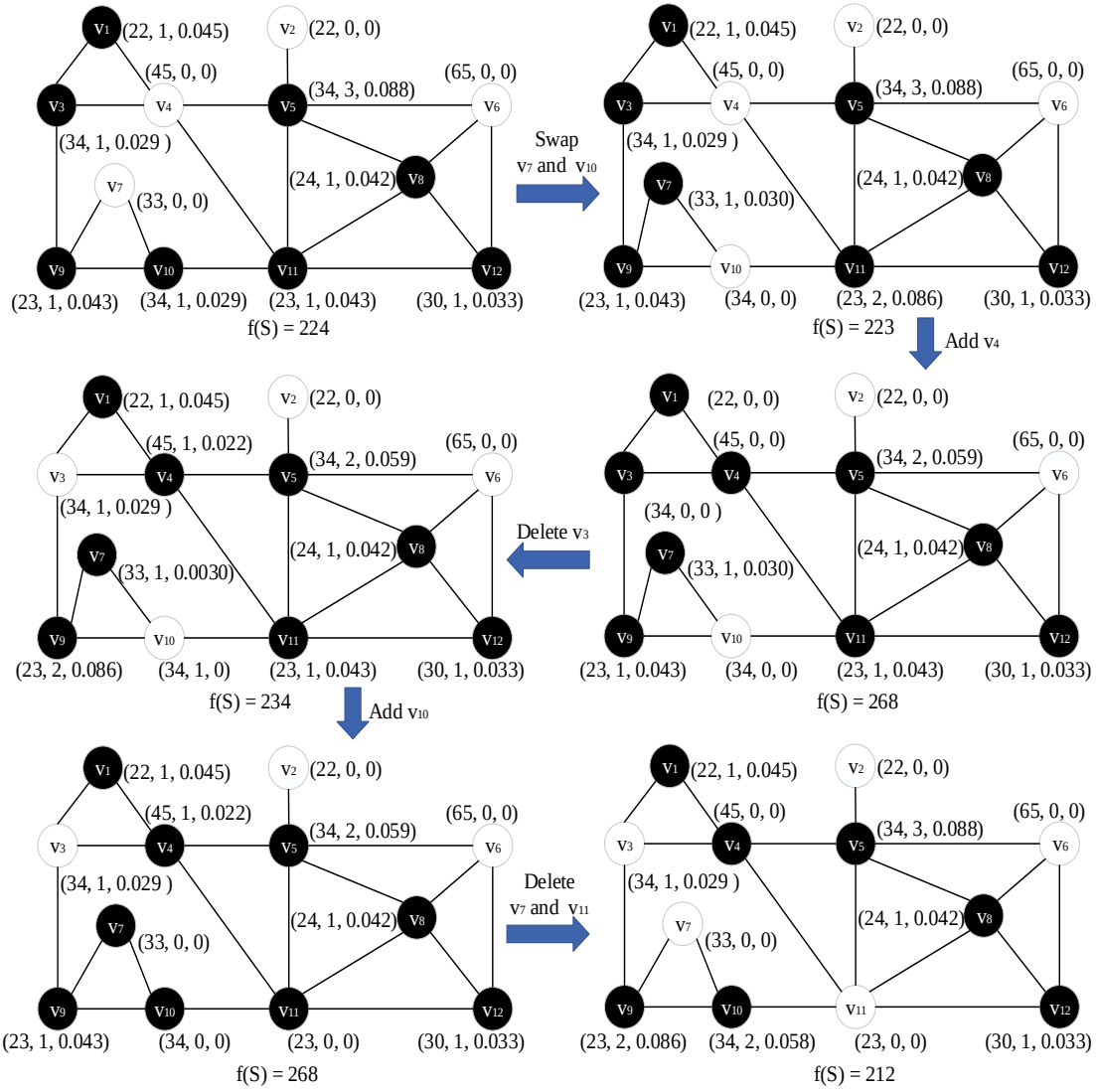

Figure 2. The illustration of the iterated neighborhood search.

During the neighborhood search, if we only consider the *Dscore*, v_1 and v_3 would have the same probability to be deleted in the third iteration, even though v_3 is with the bigger *Wscore*, which would lead the search to converge more slowly. Actually, the version MONSD^{Wscore} only considering the *Wscore* needs to use the *Dscore* to judge whether there exists the feasible *Delete* or *Swap* operator. Therefore, the version MONSD^{Wscore} and MONSD^{WDscore} have little difference during neighborhood search if they have the same initial solution. However, the important difference between MONSD^{Wscore} and MONSD^{WDsocre} exists in the initialization procedure which plays a critical role in the framework. For example, v_2 has a higher priority to be added to the solution in the version MONSD^{Wscore} due to the smaller weight. v_6 has the same possibility with v_{12} to add to the solution in the version MONSD^{Dscore}, although v_6 has the bigger weight. Therefore, we should take the *Dscore* and *Wscore* into account in the initializing phase and neighborhood search if we want to optimize the objectives well. The experimental results also indicate that the version MONSD^{WDscore} is more effective than the other two versions.

4. Computational Results

4.1. Benchmark

To evaluate the performance of our algorithm MONSD for the multi-objective minimum weighted vertex cover problem, experiments were carried out on some instances taken from [28], which have been widely used for minimum weighted vertex cover. Because of the feature of the multi-objective, we selected 48 instances, which were divided into small-scale, moderate-scale and large-scale for illustrating the results clearly. Each instance was an N-vertex and M-edge undirected graph and each vertex was associated with a weight that was randomly drawn from a uniform distribution of the interval [20, 120]. For simplicity, in this study, we tested our algorithm on the bi-objective minimum weighted vertex cover problem. Therefore, we associated each vertex with a second weight according to the same method. The instances are named as *VC-N-M*, where N is the number of vertexes and M is the number of edges.

4.2. Experimental Protocol

Our proposed algorithm MONSD was coded in C++ and compiled using GNU G++. We conducted our experiment on a computer with Intel(R) Xeon(R) CPU E7-4830 with 2.13 GHz. The stop criterion of our algorithm was the time limit 500 s on the small and moderate graphs and 1000 s on the large instance. In this study, the maximum step of neighborhood search was set as 3000, which is relatively smaller compared with most algorithms. Because in our algorithm we need to maintain a population and apply neighborhood search to each individual, we set the maximum step a little smaller to avoid consuming much time. We set the size of the population as 50 and the parameter α used in constructing RCL as 0.8. Here, the size of P_{non} was also set to 50. In the implement, we recorded the number of non-dominated solutions in the P_{non} and found that it did not exceed the size of population in all instances. Therefore, for MMWVCP, we think it is reasonable to set the size of P_{non} to 50.

4.3. Performance Metrics

In this study, we mainly used the following two metrics to do performance evaluation:

1. Hyper-volume indication (I_H): It is a comprehensive metric that can evaluate the convergence and diversity of the algorithm. Besides, it can give an intuitive value. For the two objectives, I_H indicates the area covered by a set of non-dominated solutions. The value I_H is calculated based on the objective values and we normalized it into [0, 1]. Let $p = \{f_1, f_2, ..., f_m\}$ be one point in objective space. We normalized the point as the equation: $f_i^* = (f_i - F_{(i,min)})/(F_{(i,max)} - F_{(i,min)})$, where $F_{(i,min)}$ and $F_{(i,max)}$ are the maximal and minimal value of the ith objective and they were found on all Pareto Fronts obtained by all compared algorithms. There is an example in Figure 3, where (0.2, 0.8) and (0.8, 0.2) are two normalized points and (1.2, 1.2) is the reference point. In this case, I_H is the area surrounded by dashed lines.

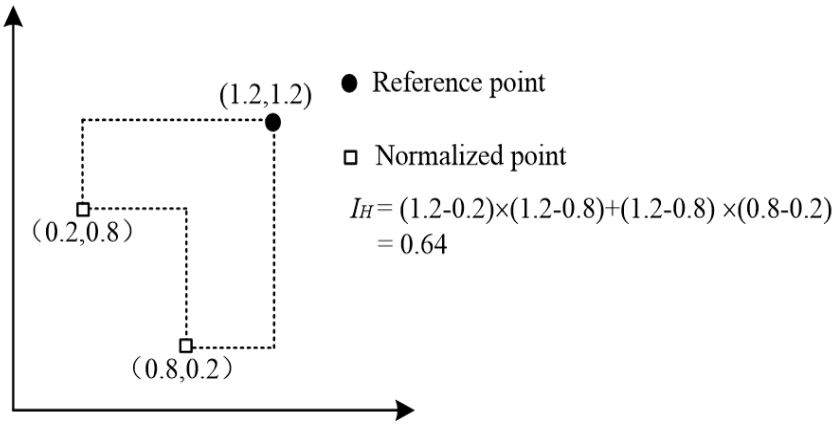

Figure 3. Hyper-volume indicator I_H.

2. Set coverage (*C-metric*): A and B are two final solution sets gained by different algorithms, which are approximations to the Pareto Front of a MOP. $C(A, B)$ indicates the percentage of the solutions in B that are dominated by at least one solution in A. $C(A, B)$ can be formally defined as follows:

$$C(A, B) = \frac{|\{S \in B | \exists S' \in A : S' \text{ dominates } S\}|}{|B|} \quad (7)$$

$C(A, B)$ = 100% implies that, for each solution S in B, there always exists at least one solution in A which dominates S. Likewise, $C(A, B)$ = 0 indicates that there is no solution in B which is dominated by one solution in A.

4.4. Comparison Results with NSGA-II

Non-Dominated Sorting Genetic Algorithm II (NSGA-II) is a popular multi-objective evolutionary algorithm which is based on the elitist strategy and fast non-dominated sorting [29] and has been used in many studies [30–32]. NSGA-II adopts the fast non-dominated sorting approach, maintains a mating pool by combining parent and child populations and selects the elitists from the offspring. NSGA-II has been widely applied in many problems due to its effectiveness. To prove the performance of our new algorithm MONSD, we conducted the comparisons with NSGA-II. For NSGA-II, we set the size of the population as 50 and the cut-off time was 500 and 1000 s on the small and moderate instances and large instances, respectively.

The experimental results are shown in Tables 1 and 2. We mark the better results in bold. Given the I_H, MONSD performed better than NSGA-II on all instances. NSGA-II obtained I_H bigger than 1 only on two instances, whereas MONSD obtained I_H bigger than 1 on 31 instances out of 33. I_H could evaluate the convergence and diversity of the algorithm. Therefore, the results of I_H shown in Tables 1 and 2 indicate that our algorithm performed better than NSGA-II considering the convergence and diversity. Subsequently, we paid attention to the performance metrics *C-metric*, which focuses on the quality of individual solution in the final non-dominated set. On the small and moderate instances, MONSD obtained 100% on 30 instances out of 33. On the other three instances, however, NSGA-II could not get 100%. Meanwhile, even on those three instances MONSD could obtain relatively diverse curves when observing the distribution of the final non-dominated sets, as discussed in the following section. On the large instances, as shown in Table 2, MONSD could obtain 100% and performed better than NSGA-II on all 15 instances when considering *C-metric*.

Table 1. Experimental results of MONSD in comparison to NSGA-II on small and moderate instances instances.

Intances	I_H		*C-Metric*	
	MONSD	**NSGA-II**	**MONSD, NSGA-II**	
			C(A,B)	**C(B,A)**
VC-100-100	**1.22**	1.08	0.00	**54.17**
VC-100-250	**1.21**	0.94	0.00	**28.57**
VC-100-500	**1.21**	0.18	**100.00**	0.00
VC-100-750	**1.08**	0.27	**100.00**	0.00
VC-100-1000	**1.16**	0.20	**100.00**	0.00
VC-100-2000	**0.96**	0.06	**100.00**	0.00
VC-150-150	**1.09**	0.59	**100.00**	0.00
VC-150-250	**1.21**	0.70	**100.00**	0.00
VC-150-500	**1.12**	0.19	**100.00**	0.00
VC-150-750	**1.22**	0.33	**100.00**	0.00
VC-150-1000	**1.18**	0.22	**100.00**	0.00
VC-150-2000	**1.08**	0.25	**100.00**	0.00
VC-150-3000	**1.17**	0.07	**100.00**	0.00
VC-200-250	**1.21**	0.28	**100.00**	0.00
VC-200-500	**1.22**	0.19	0.00	**46.15**
VC-200-750	**1.21**	0.17	**100.00**	0.00
VC-200-1000	**1.16**	0.19	**100.00**	0.00
VC-200-2000	**1.23**	0.29	**100.00**	0.00
VC-200-3000	**1.17**	0.11	**100.00**	0.00
VC-250-250	**1.20**	0.34	**100.00**	0.00
VC-250-500	**1.19**	1.04	**100.00**	0.00
VC-250-750	**1.22**	0.22	**100.00**	0.00
VC-250-1000	**1.16**	0.26	**100.00**	0.00
VC-250-2000	**1.30**	0.11	**100.00**	0.00
VC-250-3000	**1.23**	0.16	**100.00**	0.00
VC-250-5000	**1.26**	0.10	**100.00**	0.00
VC-300-300	**1.14**	0.24	**100.00**	0.00
VC-300-500	**1.20**	0.22	**100.00**	0.00
VC-300-750	**1.27**	0.21	**100.00**	0.00
VC-300-1000	**1.21**	0.18	**100.00**	0.00
VC-300-2000	**1.29**	0.21	**100.00**	0.00
VC-300-3000	**1.29**	0.20	**100.00**	0.00
VC-300-5000	**1.14**	0.08	**100.00**	0.00
Average	**1.19**	0.30	**90.91**	3.91

Table 2. Experimental results of MONSD in comparison to NSGA-II on large instances.

Intances	I_H		*C-Metric*	
	MONSD	NSGA-II	MONSD, NSGA-II	
			C(A,B)	C(B,A)
VC-500-500	**1.07**	0.68	**100.00**	0.00
VC-500-1000	**1.26**	0.44	**100.00**	0.00
VC-500-2000	**1.02**	0.27	**100.00**	0.00
VC-500-5000	**1.29**	0.39	**100.00**	0.00
VC-500-10000	**1.36**	0.24	**100.00**	0.00
VC-800-500	**1.22**	0.79	**100.00**	0.00
VC-800-1000	**1.20**	0.09	**100.00**	0.00
VC-800-2000	**1.24**	0.41	**100.00**	0.00
VC-800-5000	**1.17**	0.09	**100.00**	0.00
VC-800-10000	**1.28**	0.10	**100.00**	0.00
VC-1000-1000	**1.22**	0.72	**100.00**	0.00
VC-1000-5000	**0.98**	0.07	**100.00**	0.00
VC-1000-10000	**1.27**	0.09	**100.00**	0.00
VC-1000-15000	**1.39**	0.08	**100.00**	0.00
VC-1000-20000	**1.23**	0.08	**100.00**	0.00
Average	**1.21**	0.30	**100.00**	0.00

From the analysis of experimental results, we found that MONSD performed much better than NSGA-II when considering the quality of the final non-dominated sets.

4.5. The Effectiveness of the Score Function

In this section, we show the results of our algorithm with different functions to prove the effectiveness of our new score function *WDscore*. Therefore, we conducted our experiment with the algorithm MONSD combined with *WDscore* (MONSD^{WDscore}), and the algorithm MONSD combined with *Wscore* (MONSD^{Wscore}) and the algorithm MONSD combined with *Dscore* (MONSD^{Dscore}). In Tables 3 and 4, we use *WDscore*, *Wscore* and *Dscore* to indicate the different versions, respectively. There was little difference when considering the time used by each version. Hence, we do not show the time due to the limitation of space.

4.5.1. Experiments on Small-Scale and Moderate-Scale Instances

In this section, we show the experimental results on small-scale and moderate-scale instances. We mainly compared the performance of three algorithms in terms of I_H and *C-metric* and the results are summarized in Table 3. In Table 3, the column I_H shows the final results obtained by MONSD^{WDscore}, MONSD^{Wscore} and MONSD^{Dscore}. From the definition of Hyper-volume, we know that the bigger the value is, the better the result is. We mark the better results in bold. We found that the version MONSD^{WDscore} could perform better than the other versions except the *VC*-200-200. However, on this instance, MONSD^{WDscore} could obtain the I_H value of 1.23 while MONSD^{Wscore} provides the I_H value of 1.25. The gap between the two results is small compared with those gaps on the other instances. Overall, we found that MONSD^{Dscore} performed relatively poorly because it only applies the *Dscore*, which is based on vertexes' degrees and does not consider the weights. Therefore, MONSD^{Dscore} may lead to results with fewer vertexes but with bigger weights, meaning the objectives cannot be optimized well. In the last row, we list the average values I_H of different versions, which also indicates the MONSD^{WDscore} performed much better.

In Table 3, we list the *C-metric* values of MONSD^{WDscore} compared with MONSD^{Wscore} and MONSD^{Dscore} in the column *C-metric*, which indicates MONSD^{WDscore} was much or slightly better than the other versions on most instances. For the instance *VC*-200-2000, MONSD^{WDscore} was still a little worse than MONSD^{Wscore}, similar to the I_H. In the mixed score function, the *Dscore* is numerator,

which may reduce the effect of *Wscore* and lead to worse results. However, when we analyzed the distribution (listed in next section) of the final sets obtained by different versions, we found that even on the *VC*-200-2000 MONSD^{WDscore} was also competitive compared with others.

Table 3. Experimental results on small-scale and moderate-scale instances.

Instances	I_H			*C-Metric*			
	WDsocre	Wscore	Dscore	(WDscore, Wscore)		(WDscore, Dscore)	
				C(A,B)	C(B,A)	C(A,B)	C(B,A)
VC-100-100	**1.22**	1.19	1.04	**28.57**	4.17	**85.71**	4.17
VC-100-250	**1.21**	1.21	0.71	**11.11**	9.52	**100.00**	0.00
VC-100-500	**1.21**	1.19	0.75	**20.00**	18.75	**100.00**	0.00
VC-100-750	**1.08**	1.03	0.79	**45.45**	28.57	**100.00**	0.00
VC-100-1000	**1.16**	1.09	0.83	**55.56**	0.00	**100.00**	0.00
VC-100-2000	**0.96**	0.72	0.68	**0.60**	0.00	**60.00**	0.00
VC-150-150	**1.09**	1.07	0.62	**38.46**	6.25	**100.00**	0.00
VC-150-250	**1.21**	1.20	0.90	**50.00**	36.36	**100.00**	0.00
VC-150-500	**1.12**	1.02	0.55	**63.64**	25.00	**100.00**	0.00
VC-150-750	**1.22**	1.18	0.62	**80.00**	11.76	**100.00**	0.00
VC-150-1000	**1.18**	1.17	0.59	**61.11**	40.00	**100.00**	0.00
VC-150-2000	**1.08**	1.06	0.69	**50.00**	25.00	**100.00**	0.00
VC-150-3000	**1.17**	1.10	0.96	**50.00**	9.09	**71.00**	0.00
VC-200-250	**1.21**	1.14	0.83	**50.00**	18.18	**100.00**	0.00
VC-200-500	**1.22**	1.04	0.39	**78.57**	14.29	**100.00**	0.00
VC-200-750	**1.21**	1.15	0.44	**50.00**	23.08	**100.00**	0.00
VC-200-1000	**1.16**	1.10	0.40	**66.67**	16.67	**100.00**	0.00
VC-200-2000	1.23	**1.25**	0.71	33.33	**57.89**	**100.00**	0.00
VC-200-3000	**1.17**	1.11	0.77	**41.67**	8.33	**100.00**	0.00
VC-250-250	**1.20**	1.12	0.83	**81.25**	12.50	**100.00**	0.00
VC-250-500	**1.19**	1.17	0.60	**50.00**	15.38	**100.00**	0.00
VC-250-750	**1.22**	1.18	0.59	**60.87**	35.00	**100.00**	0.00
VC-250-1000	**1.16**	1.14	0.69	**45.45**	33.33	**100.00**	0.00
VC-250-2000	**1.30**	1.12	0.47	**88.89**	0.00	**100.00**	0.00
VC-250-3000	**1.23**	1.19	0.47	**61.11**	57.89	**100.00**	0.00
VC-250-5000	**1.26**	1.17	0.74	**83.33**	0.00	**100.00**	0.00
VC-300-300	**1.14**	0.90	0.66	**100.00**	0.00	**100.00**	0.00
VC-300-500	**1.20**	1.13	0.66	**85.00**	11.11	**100.00**	0.00
VC-300-750	**1.27**	1.15	0.45	**90.48**	4.35	**100.00**	0.00
VC-300-1000	**1.21**	1.07	0.45	**86.67**	17.65	**100.00**	0.00
VC-300-2000	**1.29**	1.13	0.43	**100.00**	0.00	**100.00**	0.00
VC-300-3000	**1.29**	1.22	0.56	**83.33**	16.67	**100.00**	0.00
VC-300-5000	**1.14**	1.01	0.32	**85.71**	5.88	**100.00**	0.00
Average	**1.19**	1.11	0.64	**59.90**	17.05	**97.00**	0.13

4.5.2. Experiments on Large-Scale Instances

Among the weighted vertex cover problem instances, there are many large-scale instances. Therefore, to better show the performance of our proposed algorithm, we conducted the experiments on large scale instances and the results are listed in Table 4 in terms of I_H and *C-metric*. From the results shown in Table 4, we found that MONSD^{WDscore} performed much better than MONSD^{Wscore} and MONSD^{Dscore} on all instances. The average I_H of MONSD^{WDscore} was 1.21 while for MONSD^{Wscore} and MONSD^{Dscore} it was 1.04 and 0.37, respectively. Meanwhile, on some instances, the *C-metric* of MONSD^{WDscore} reached 100% compared with MONSD^{Wscore}. On all instances, the *C-metric* of MONSD^{WDscore} reached 100% compared with MONSD^{Dscore}. Hence, we could conclude that MONSD^{WDscore} performed better than the other two versions considering the objective of solutions and the number of non-dominated solutions.

Table 4. Experimental results on large-scale instances.

Instances	I_H			*C-Metric*			
	WDsocre	Wscore	Dscore	(WDscore, Wscore)		(WDscore, Dscore)	
				C(A,B)	C(B,A)	C(A,B)	C(B,A)
VC-500-500	**1.07**	1.02	0.43	**86.11**	17.86	**100.00**	0.00
VC-500 -1000	**1.26**	1.04	0.42	**100.00**	0.00	**100.00**	0.00
VC-500-2000	**1.02**	0.78	0.19	**80.00**	9.09	**100.00**	0.00
VC-500-5000	**1.29**	1.19	0.36	**80.00**	31.58	**100.00**	0.00
VC-500-10000	**1.36**	1.16	0.46	**100. 00**	0.00	**100.00**	0.00
VC-800-500	**1.22**	0.98	0.69	**100.00**	0.00	**100.00**	0.00
VC-800-1000	**1.20**	1.11	0.56	**71.43**	5.26	**100.00**	0.00
VC-800-2000	**1.24**	0.94	0.53	**100.00**	0.00	**100.00**	0.00
VC-800-5000	**1.17**	1.10	0.32	**82.35**	10.71	**100.00**	0.00
VC-800-10000	**1.28**	1.07	0.24	**81.25**	0.00	**100.00**	0.00
VC-1000-1000	**1.22**	1.00	0.55	**100.00**	0.00	**100.00**	0.00
VC-1000-5000	**0.98**	0.75	0.25	**100.00**	0.00	**100.00**	0.00
VC-1000-10000	**1.27**	1.11	0.15	**66.67**	15.79	**100.00**	0.00
VC-1000-15000	**1.39**	1.26	0.28	**100.00**	0.00	**100.00**	0.00
VC-1000-20000	**1.23**	1.05	0.09	**50.00**	7.69	**100.00**	0.00
Average	**1.21**	1.04	0.37	**86.52**	6.53	**100.00**	0.00

4.5.3. Discussion

- Why is the new score function more effective than the other two functions?
  For the sub-problem objective, we aim to minimize the sum of the associated weights of the vertexes. In general, there are two approaches to obtain the goal. The first is to minimize the number of vertexes in the solution, that is, *Dscore*. *Dscore* is used to evaluate how many edges would change the state if a vertex were flipped. The purpose of *Dscore* is to keep fewer vertexes in the solution but cover more edges. The second is to minimize total weights directly. In the implementation, we prefer the vertexes with smaller *Wscore*. *Wscore* can compute how much benefit a vertex can bring for the objective directly. However, there are drawbacks if we only use *Dscore* or *Wscore*, which may lead the vertex with bigger (smaller) *Dscore* but also bigger (smaller) *Wscore* in the solution. Actually, we should prefer the vertex with the bigger *Dscore* but smaller *Wscore*. Therefore, we combine the *Dscore* and *Wscore*, that is, *WDscore*. *Dscore* is the numerator and *Wscore* is the denominator. Thus, we can optimize the number of vertex and the sum of weights simultaneously. *WDscore* takes two factors into account, thus it is more effective.
- How is the convergence of the neighborhood search?
  In Figure 4, we present how the neighborhood search works. Firstly, it explores the neighborhood $Delete(S)$ and $Swap(S)$ until there is no neighbor solution better than the current solution. If there are two feasible operators, we choose the operator which can bring greater benefit for the objective. No better neighbor solution in $Delete(S)$ and $Swap(S)$ neighborhood indicates that the search is convergent to some extent. Then, the $Add(S)$ neighborhood is explored. Actually, the search cannot get any improvement in $Add(S)$ neighborhood and it aims to explore the bigger solution space. After the $Add(S)$ neighborhood, the search returns back to perform the $Delete(S)$ and $Swap(S)$ search. Therefore, the search finally converges.

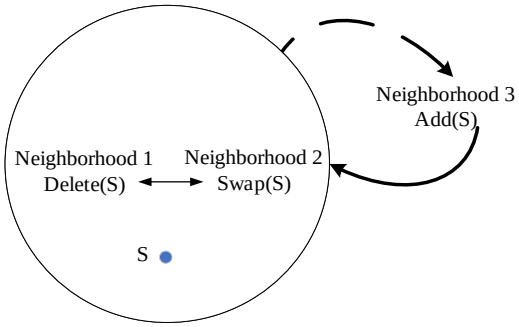

Figure 4. The illustration of neighborhood search.

4.6. The Analysis of Distribution

We analyzed the distribution of the final non-dominated sets obtained by MONSD^{WDscore}, MONSD^{Wscore}, MONSD^{Dscore} and NSGA-II. Due to the limitation of space, we selected some representative instances that vary from small-scale instances to large-scale instances and also contain sparse and dense instances. Because the multi-objective minimum weighted vertex cover problem is discrete, the curve is not too smooth. However, the curve can describe the tendency of the problem. In Figure 5, we can see that MONSD could find a better spread and more solutions in the entire Pareto optimal region compared with NSGA-II. On the VC-100-100 instance, NSGA-II obtained a better *C-metric* value than MONSD. However, it did not have a good spread and the solutions were clustered in a small region. We also found that MONSD^{WDscore} performed better than MONSD^{Wscore} and MONSD^{Dscore}, although, on the instance *VC*-200-2000 mentioned above, MONSD^{WDscore} obtained a relatively diverse and smooth curve.

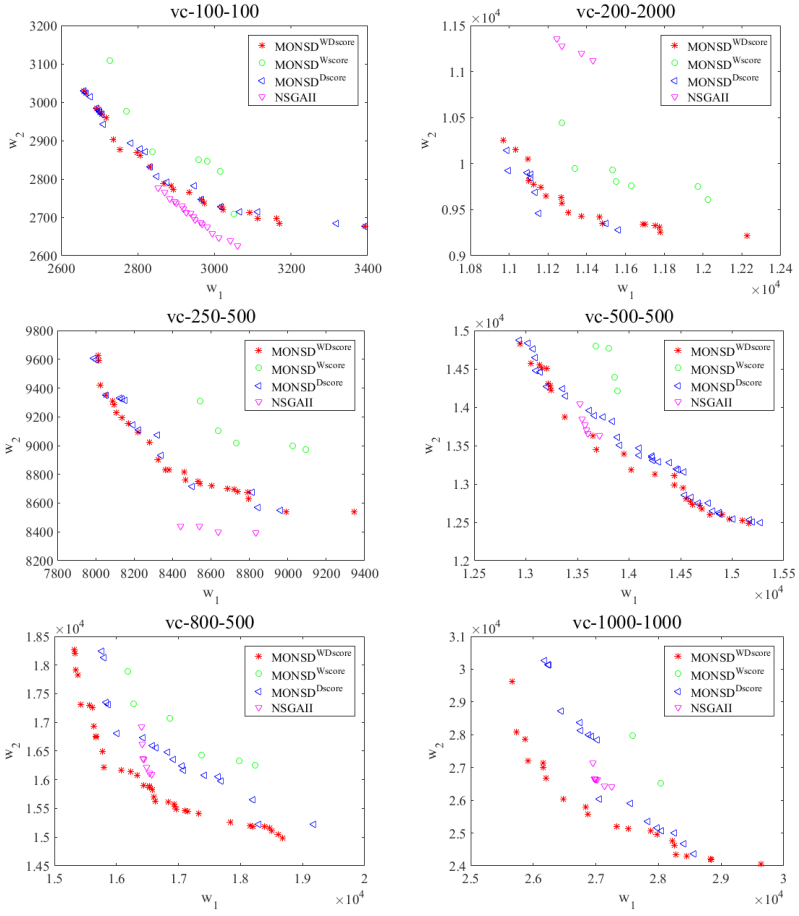

Figure 5. Distribution of the final sets obtained by three versions on some instances.

5. Conclusions and Future Work

In this paper, an effective multi-objective algorithm integrating iterated neighborhood search with decomposition technique is proposed to solve multi-objective minimum weighted vertex cover problem. To start with, a greedy initializing procedure combined with restricted candidate list mechanism is used to generate the population. Subsequently, according to the features of the problem, we design a score function to explore the search space effectively. Then, we apply the neighborhood search including three operators to improve the solution in the population. Finally, we utilize perturbation to help the search to jump out of the local optima. Computational results on instances verified the performance of our algorithm.

To demonstrate the performance of the proposed algorithm, extensive experiments were carried out on small, moderate and large graphs. We compared our experimental results with the classical multi-objective algorithm non-dominated sorting genetic algorithm II. It was obviously shown that our algorithm can provide much better results than the comparative algorithm on several metrics. In the future, the proposed algorithm is expected to solve other multi-objective problems such as the multi-objective minimum weighed dominating set problem. Meanwhile, we plan to study the theoretical aspects as well as the performance of the method.

Author Contributions: Software, S.H.; Methodology, R.L. and X.W.; Writing—original draft preparation, S.H. and H.L.; and Writing—review and editing, M.Y. and Y.W.

Funding: This work was supported by Jilin education department 13th five-year science and technology project under Grant Nos. JJKH20190726KJ, JJKH20190756SK, and JJKH20180465KJ and the National Natural Science Foundation of China (NSFC) under Grant Nos. 61502464, 61503074 and 61806082.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MOP	Multi-objective Optimization Problem
COP	Constraint Optimization Problem
BB	Branch and Bound
MBE	Mini-Bucket Elimination
MO-MBE	Multi-objective Optimization Problem with Mini-Bucket Elimination
MOEA/D	Multi-objective Evolutionary Algorithm Based on Decomposition
MVCP	Minimum Vertex Cover Problem
MWVCP	Minimum Weighted Vertex Cover Problem
MMWVCP	Multi-objective Minimum Weighted Vertex Cover Problem
BMWVCP	Bi-objective Minimum Weighted Vertex Cover Problem
MONSD	Multi-objective Iterated Neighborhood Search based on Decomposition Algorithm
RCL	Restricted Candidate List
NSGA-II	Non-dominated Sorting Genetic Algorithm II

References

1. Hillermeier, C. *Nonlinear Multiobjective Optimization: A Generalized Homotopy Approach*; Springer Science & Business Media: Berlin, Germany, 2001.
2. Hwang, C.L.; Masud, A.S.M. *Multiple Objective Decision Making—Methods and Applications: A State-of-the-Art Survey*; Springer: Berlin, Germany, 1994.
3. Marler, R.T.; Arora, J.S. Survey of multi-objective optimization methods for engineering. *Struct. Multidiscip. Optim.* **2004**, *26*, 369–395. [CrossRef]
4. Huang, H.Z.; Qu, J.; Zuo, M.J. A new method of system reliability multi-objective optimization using genetic algorithms. In Proceedings of the RAMS'06—Annual Reliability and Maintainability Symposium, Newport Beach, CA, USA, 23–26 January 2006; pp. 278–283.
5. Coello, C.A. An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surv.* **2000**, *32*, 109–143. [CrossRef]

6. Stadler, W. A survey of multicriteria optimization or the vector maximum problem, part I: 1776–1960. *J. Optim. Theory Appl.* **1979**, *29*, 1–52. [CrossRef]
7. Lieberman, E.R. Soviet multi-objective mathematical programming methods: An overview. *Manag. Sci.* **1991**, *37*, 1147–1165. [CrossRef]
8. Fishburn, P.C. A survey of multiattribute/multicriterion evaluation theories. In *Multiple Criteria Problem Solving*; Springer: Berlin/Heidelberg, Germany, 1978; pp. 181–224.
9. Evans, G.W. An overview of techniques for solving multiobjective mathematical programs. *Manag. Sci.* **1984**, *30*, 1268–1282. [CrossRef]
10. Dechter, R. Mini-buckets: A general scheme for generating approximations in automated reasoning. In Proceedings of the International Joint Conferences on Artificial Intelligence, Nagoya, Japan, 23–29 August 1997; pp. 1297–1303.
11. Emma, R.; Javier, L. Constraint optimization techniques for exact multi-objective optimization. In *Multiobjective Programming and Goal Programming*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 89–98.
12. Kalyanmoy, D. *Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction*; John Wiley & Sons: Hoboken, NJ, USA, 2011; pp. 75–96.
13. Laumanns, M.; Thiele, L.; Deb, K.; Zitzler, E. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolut. Comput.* **2002**, *10*, 263–282. [CrossRef]
14. GarciaNajera, A.; Bullinaria, J.A. An improved multi-objective evolutionary algorithm for the vehicle routing problem with time windows. *Comput. Oper. Res.* **2011**, *38*, 287–300. [CrossRef]
15. Carvalho, R.D.; Saldanha, R.R.; Gomes, B.N.; Lisboa, A.C.; Martins, A.X. A multi-objective evolutionary algorithm based on decomposition for optimal design of Yagi-Uda antennas. *IEEE Trans. Magn.* **2012**, *48*, 803–806. [CrossRef]
16. Zhu, Z.; Zhang, G.; Li, M.; Liu, X. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *27*, 1344–1357. [CrossRef]
17. Zhang, Q.; Li, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evolut. Comput.* **2007**, *11*, 712–731. [CrossRef]
18. Ke, L.; Zhang, Q.; Battiti, R. Hybridization of decomposition and local search for multiobjective optimization. *IEEE Trans. Cybern.* **2014**, *44*, 1808–1820. [PubMed]
19. Li, X.; Li, M. Multiobjective local search algorithm-based decomposition for multiobjective permutation flow shop scheduling problem. *IEEE Trans. Eng. Manag.* **2015**, *62*, 544–557. [CrossRef]
20. Hu, Y.; Yang, B.; Wong, H.S. A weighted local view method based on observation over ground truth for community detection. *Inf. Sci.* **2016**, *355*, 37–57. [CrossRef]
21. Fernau, H.; Fomin, F.V.; Philip, G.; Saurabh, S. On the parameterized complexity of vertex cover and edge cover with connectivity constraints. *Theor. Comput. Sci.* **2015**, *565*, 1–15. [CrossRef]
22. Wang, L.; Du, W.; Zhang, Z.; Zhang, X. A PTAS for minimum weighted connected vertex cover P_3 problem in 3-dimensional wireless sensor networks. *J. Comb. Optim.* **2017**, *33*, 106–122. [CrossRef]
23. Pullan, W. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optim.* **2009**, *6*, 214–219. [CrossRef]
24. Li, R.; Hu, S.; Zhang, H.; Yin, M. An efficient local search framework for the minimum weighted vertex cover problem. *Inf. Sci.* **2016**, *372*, 428–445. [CrossRef]
25. Pourhassan, M.; Feng, S.; Frank, N. Parameterized analysis of multi-objective evolutionary algorithms and the weighted vertex cover problem. In Proceedings of the International Conference on Parallel Problem Solving from Nature, Edinburgh, UK, 17–21 September 2016.
26. Resende, M.G.C. *Greedy Randomized Adaptive Search Procedures*; Springer US: New York, NY, USA, 2001; pp. 219–249.
27. Resende, M.G.C.; Ribeiro, C.C. Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. *J. Glob. Optim.* **2010**, *6*, 109–133.
28. Shyu, S.J.; Yin, P.Y.; Lin, B.M.T. An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem. *Ann. Oper. Res.* **2004**, *131*, 283–304. [CrossRef]
29. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T.A. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolut. Comput.* **2002**, *6*, 197. [CrossRef]
30. Bekele, E.G.; Nicklow, J.W. Multi-objective automatic calibration of SWAT using NSGA-II. *J. Hydrol.* **2007**, *341*, 165–176. [CrossRef]

31. Alikar, N.; Mousavi, S.M.; Ghazilla, R.A.; Tavana, M.; Olugu, E.U. Application of the NSGA-II Algorithm to A Multi-Period Inventory-Redundancy Allocation Problem in a Series-Parallel System. *Reliab. Eng. Syst. Saf.* **2017**, *160*, 1–10. [CrossRef]
32. Vo-Duy, T.; Duong-Gia, D.; Ho-Huu, V.; Vu-Do, H.C.; Nguyen-Thoi, T. Multi-objective optimization of laminated composite beam structures using NSGA-II algorithm. *Compos. Struct.* **2017**, *168*, 498–509. [CrossRef]