Weighting-based Variable Neighborhood Search for Optimal Camera Placement*

Zhouxing Su, [1] **Qingyun Zhang,** [1] **Zhipeng Lü,** [1†] **Chumin Li,** [2] **Weibo Lin,** [3] **Fuda Ma**[3]

[1]SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]Université de Picardie Jules Verne, France
[3]Huawei Technologies Co., Ltd., China
zhipeng.lv@hust.edu.cn

Abstract

The optimal camera placement problem (OCP) aims to accomplish surveillance tasks with the minimum number of cameras, which is one of the topics in the GECCO 2020 Competition and can be modeled as the unicost set covering problem (USCP). This paper presents a weighting-based variable neighborhood search (WVNS) algorithm for solving OCP. First, it simplifies the problem instances with four reduction rules based on dominance and independence. Then, WVNS converts the simplified OCP into a series of decision unicost set covering subproblems and tackles them with a fast local search procedure featured by a swap-based neighborhood structure. WVNS employs an efficient incremental evaluation technique and further boosts the neighborhood evaluation by exploiting the dominance and independence features among neighborhood moves. Computational experiments on the 69 benchmark instances introduced in the GECCO 2020 Competition on OCP and USCP show that WVNS is extremely competitive comparing to the state-of-the-art methods. It outperforms or matches several best performing competitors on all instances in both the OCP and USCP tracks of the competition, and its advantage on 15 large-scale instances are over 10%. In addition, WVNS improves the previous best known results for 12 classical benchmark instances in the literature.

1 Introduction

The optimal camera placement problem (OCP) aims to minimize the cost for performing surveillance tasks, which can be modeled as the unicost set covering problem (USCP). Each sample point (element) in the target area (universal set) is covered by several candidate camera positions (subsets) and we need to cover all samples (elements) by picking the minimum number of candidates (subsets) to place cameras.

The optimal camera placement problem is one of the topics of the GECCO 2020 Competition, which consists of two tracks: OCP track and USCP track. The difference between them lies in the fact that the geometric information of samples and candidates are available in the OCP track, while it is modeled as a general unicost set covering problem in the USCP track. Although the early researches on OCP start from the surveillance of continuous space (Kritter et al. 2019), which received much attention from the computational geometry community, recent advances usually focus on the combinatorial models (Liu, Sridharan, and Fookes 2016). By properly discretizing the target space into a set of sample points, we can represent the sight of each candidate camera position by a subset of sample points. Therefore, the problem of filling the target space with geometric shapes becomes finding a union of subsets to cover all sample points.

Due to the close relation between the optimal camera placement and the set covering problem (SCP), the computational experiments for OCP were usually performed on the SCP instances in the OR-Library (Beasley 1990). However, most of these instances are randomly generated and do not respect the specific geometric properties in the context of OCP, which are not very objective for evaluating OCP solution methods. Recently, Brévilliers et al. (2018) proposed a dedicated benchmark dataset for OCP and tested their hybrid differential evolution algorithm on these instances. Their experimental results show that their algorithm outperforms the state-of-the-art general USCP solvers in the literature, indicating that there is huge room for improvement to the dedicated OCP solvers. Furthermore, the GECCO 2020 Competition on OCP and USCP introduces more real-world datasets and provides an opportunity to make fair comparison among various approaches.

The unicost set covering problem is one of the most classical NP-hard problems (Gao et al. 2015). It can be used to formulate many real-world optimization problems, such as logical function minimization (Seda 2007; Steinbach and Posthoff 2012), sensor placement (Rui et al. 2017), software test suite reduction (Chi et al. 2017), team formation (Demirović et al. 2018), and facility location (Zhang et al. 2020). In addition, several well-known NP-hard problems can be reduced to USCP, including dominating set (Cai et al. 2020) and vertex cover (Cai, Su, and Sattar 2011). As a specialization of the set covering problem, USCP can be tackled by any algorithm for SCP. Caprara, Toth, and Fischetti (2000) presented a comprehensive benchmark on different approaches for SCP. Their computational results indicate that solving the mixed integer programming model by general solvers like CPLEX is highly competitive comparing to the dedicated exact methods. Moreover, the state-of-the-art heuristic algorithms are able to effectively tackle the instances which are too massive for the exact methods.

*Research partially supported by Alkaid lab of Huawei Cloud.
†Corresponding author

Apart from the solution approaches for SCP, many dedicated algorithms for USCP were proposed and competitive results were obtained by exploiting the specific properties of USCP (Grossman and Wool 1997). In detail, the early algorithms usually adopt local search to solve the Lagrangian relaxation of USCP (Musliu 2006; Yagiura, Kishida, and Ibaraki 2006). Lan, DePuy, and Whitehouse (2007) propose a large neighborhood search (LNS) which destroys and repairs the current solution repeatedly. Naji-Azimi, Toth, and Galli (2010) propose an electromagnetism metaheuristic based on the LNS algorithm proposed by Lan, DePuy, and Whitehouse (2007), which iteratively drops a set and solves the partial USCP to cover the resulting uncovered elements. More recently, Gao et al. (2015) propose an efficient local search with row weighting (RWLS) which outperforms the aforementioned algorithms on most benchmark instances. Although the row weighting technique and Lagrangian relaxation share some common ideas, they are very different. Instead of considering the original objective function (the number of picked sets) and the constraint violation (the number of uncovered elements) together, RWLS forbids worsening the original objective value, and only improves the original objective value when all constraints are satisfied.

The weighting technique is widely used in local search algorithms to solve challenging combinatorial optimization problems, by guiding the search to escape from the local optima. Its basic idea is similar to the Lagrangian relaxation in that it permits constraint violation and adds corresponding penalties to the objective function. The penalty of violating each constraint is associated with a specific weight which is adjusted according to the constraint violation of the explored solutions. The weighting technique is very successful on the satisfiability problem (Selman and Kautz 1993; Luo, Su, and Cai 2012), constraint satisfaction problem (Morris 1993), unicost set covering problem (Gao et al. 2015), and minimum vertex cover problem (Cai, Su, and Sattar 2011), because it diversifies the search in an adaptive way.

In this paper, we present a weighting-based variable neighborhood search (WVNS) algorithm for solving OCP and USCP. First, WVNS reduces the problem instances by rules of dominance and independence. Then, WVNS converts USCP into a series of decision subproblems and tackles them with a fast local search procedure which employs a swap-based neighborhood structure and an efficient incremental evaluation technique. Furthermore, by exploiting the dominance and independence among neighborhood moves, we propose a neighborhood reduction technique which dramatically boosts the neighborhood evaluation. The computational results on the 69 benchmark instances used in GECCO 2020 Competition on OCP and USCP demonstrate the effectiveness of our proposed algorithm. It outperforms or matches several best performing competitors on all the instances in both the OCP and USCP tracks of the competition, and its advantage on 15 large instances is over 10%. In addition, WVNS improves the best known results for 12 classical benchmark instances in the literature. Finally, we analyze the impact of the proposed reduction techniques and justify their contribution to the effectiveness of WVNS.

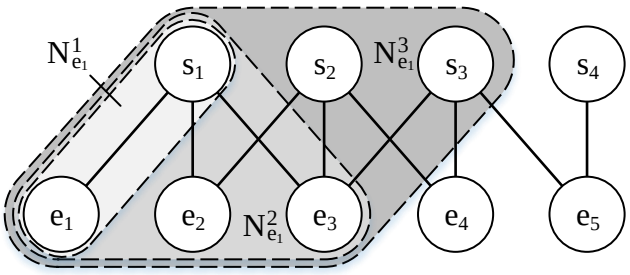


Figure 1: Bipartite graph representation of an USCP instance where $E = \{e_1, e_2, e_3, e_4, e_5\}$ and $S = \{s_1, s_2, s_3, s_4\}$. Set s_i and element e_j are adjacent if set s_i covers element e_j.

2 Preliminaries

Given a set of elements E and a set of subsets S, each subset $s \in S$ covers a set of elements $C_s \subset E$ and each element $e \in E$ is covered by a set of subsets $B_e \subset S$. The unicost set covering problem aims to cover all elements in E with the minimum number of subsets in S. In other words, if we denote the selected subsets with $X \subset S$, we need to minimize $|X|$ while satisfying the constraints $\bigcup_{s \in X} C_s = E$, i.e., $B_e \cap X \neq \varnothing, \forall e \in E$. In the sequel, we will mention the "subset" as "set" for short.

Generally, we can represent USCP instances as bipartite graphs. Each set or element corresponds to a node in the bipartite graph. If set s covers element e, i.e., $e \in C_s$, there is an edge between the corresponding pair of nodes. Figure 1 gives an example of representing a small USCP instance by a bipartite graph. With this graph representation, we can define the neighborhood N^k_e of each element $e \in E$ as the elements or sets that element e can reach within k hops. For example, in Figure 1, $N^1_{e_1} = B_{e_1} = \{s_1\}$, $N^2_{e_1} = N^1_{e_1} \cup \{e_2, e_3\}$, $N^3_{e_1} = N^2_{e_1} \cup \{s_2, s_3\}$.

3 Solution Approach

We propose a weighting-based variable neighborhood search (WVNS) for solving OCP and USCP as presented in Algorithm 1. First, it adopts reduction rules to simplify the problem instances. Then, the proposed algorithm employs a PageRank-like constructive heuristic to generate an initial feasible solution. Next, WVNS transforms USCP into a series of k-set covering decision subproblems, which aims to cover all elements using exact k sets. Starting from the initial feasible solution, once the current solution covers all elements by $k+1$ sets, WVNS randomly drops a set and turns to tackle the resulting k-set covering problem. Finally, we tackle each subproblem with a fast local search procedure.

3.1 Reduction Rules

We adopt four reduction rules to eliminate some elements and sets, and deduce that some sets must be included in the optimal solution. We repeatedly apply them in turn until no more elements or sets can be eliminated or fixed. Note that rules RD, CD, and UC are classical reduction rules proposed by Beasley (1987), while rule CC is first proposed in this study to the best of our knowledge.

Algorithm 1 The main framework of the WVNS algorithm

Input: A set of elements E, a set of subsets S
Output: The best solution found so far X^*

```
Reduce instance size                                   /* (Section 3.1) */
Initial solution X ← Init(E, S)                        /* (Section 3.2) */
X* ← X
Element weights w_e ← 1, ∀e ∈ E                        /* (Section 3.3) */
while Time limit is not reached do
    Randomly select a set q from X, X ← X \ {q}
    while Time limit is not reached do
        Find best swap move (p, q)                     /* (Section 3.4) */
        X ← X ∪ {p} \ {q}                              /* Make move */
        /* U(X) = E \ ⋃_{s∈X} C_s is a function which */
        /* returns the set of uncovered elements by X */
        if |U(X)| = 0 then
            X* ← X
            break
        else if |U(X)| ≥ |U(X ∪ {q} \ {p})| then
            w_e ← w_e + 1, ∀e ∈ U(X)
        end if
        if max_{e∈E} w_e > 2^16 then
            w_e ← w_e/2, ∀e ∈ E                        /* Smooth weights */
        end if
    end while
end while
```

- **Row dominance (RD)**. For each pair of elements $e, e' \in E$, if all the sets covering element e also cover element e', i.e., $B_e \subseteq B_{e'}$, then covering element e by any set $s \in B_e$ will also make element e' be covered. We say that element e dominates element e' in such case, and we can safely eliminate element e'.
- **Column dominance (CD)**. For each pair of distinct sets $s, s' \in S$, if all elements covered by set s are also covered by set s', i.e., $C_s \subseteq C_{s'}$, then we say that set s' dominates set s, and we can safely eliminate set s.
- **Unit clause (UC)**. If an element e is only covered by a single set s, then we must pick set s in order to cover element e. Apparently, all elements covered by set s can be eliminated consequently.
- **Connected component (CC)**. In the bipartite graph representation, we may discover that an instance G can be divided into several connected components $G'_1, G'_2, ..., G'_n$ by breadth first search. Let X'_i be the optimal solution to instance G'_i, the optimum for the original instance G can be composed by $\bigcup_{i=1}^{n} X'_i$. If the size of a subgraph G'_i is small enough, we can solve it with exact algorithms, fix the sets in X'_i, and eliminate all the elements in G'_i.

3.2 Initial Solution Generation

The proposed algorithm employs a constructive heuristic to generate an initial feasible solution. This procedure iteratively picks the most important set until all elements are covered. The importance of each set is evaluated by a PageRank-like strategy (Page et al. 1999). If an element is covered by few sets, i.e., $|B_e|$ is small, it may be hard to be covered. If a set covers many hard-to-cover elements, it could be very important. Based on these assumptions, we calculate the importance α_s of each set s by Equation (1), where we denote the importance of element e by α_e. Since each element is equally important in USCP, we adopt $\alpha_e = 1$ for the uncovered elements and $\alpha_e = 0$ for the already covered ones during the construction.

$$\alpha_s = \sum_{e \in C_s} \frac{\alpha_e}{|B_e|}, \quad \forall s \in S \tag{1}$$

The above general-purpose heuristic is applied to the instances in the USCP track. Moreover, for the OCP track, in addition to this heuristic, we adopt a tiling method which utilizes the geometric information for massive regular instances. This method is especially effective for cuboid space such as warehouses or public squares. In such cases, the monitored space is discretized by a regular grid of sample points (Brévilliers et al. 2018). Thus, we can generate high-quality solutions by exploiting the subgraph isomorphism in the bipartite graph of the monitored space, since the isomorphic substructures usually induce repeating patterns in the solution vectors. Without loss of generality, we will discuss the two-dimensional scenario in the sequel in order to simplify the description and illustration.

The proposed tiling method first builds a pool of rectangular tiles with distinct widths or heights, then it tries to fill the target area with these tiles. For each tile with a specific size, it covers the sample points in the corresponding rectangular area, and the camera placement for this limited area can be regarded as the decorative pattern of the tile. We call the procedure of determining the pattern of a tile as tile-building. One thing to notice is that, this tiling method is different from the block-building technique for the packing problems or the agglomeration approaches for the clustering problems. Each tile is not required to fully cover its containing sample points, instead, the adjacent tiles cooperate with each other to achieve seamless tiling.

In detail, the main idea of the tile-building procedure is illustrated in Figure 2. Given a tile whose width is w and height is h, if we pick a candidate camera at (x, y), the candidate cameras located at $(x, y \pm h)$, $(x \pm w, y)$, and $(x \pm w, y \pm h)$ and heading to the same direction should be picked together. This requirement guarantees that the pattern of the cooperating tiles is the same as the must-cover tile. Furthermore, if we combine the tied sets into a single larger set, the tile-building procedure itself becomes a unicost set covering problem which minimizes the number of picked combined sets to cover all the elements in the must-cover tile. However, comparing to the original USCP instance, the ones for the tile-building procedure are much smaller.

For each built tile with specific size, we try to tile the target area with it starting from different positions (x, y) and only consider placing tiles at $(x \pm i \times w, y \pm j \times h), \forall i, j \in \mathbb{Z}$. Since the edge of the target area is irregular and the tiles may go beyond the boundary, we need to remove the non-existing sets and use the general-purpose heuristic to repair the incomplete solution. Finally, we randomly choose one of the top 64 solutions over the ones produced by different size and offset configurations as the initial solution.

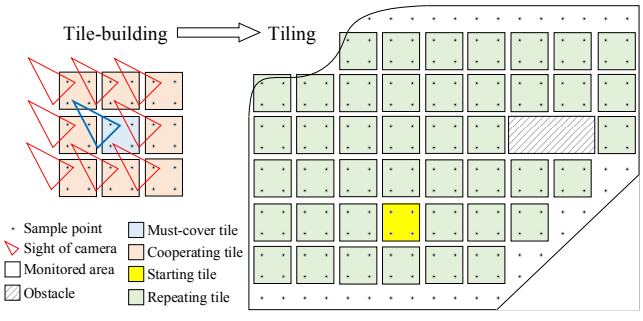


Figure 2: Key idea of the tile-building and tiling procedures.

3.3 Reformulation and Weighting Technique

It is a common technique to convert an optimization problem into a series of decision sub-problems and conquer them in sequence or even concurrently (Lü and Hao 2010; Gao et al. 2015; Zhang et al. 2020). Regarding OCP, it can be transformed into a series of k-set covering problems, denoted by OCP$_k$, which determines whether all elements can be covered by exact k sets. Once the proposed algorithm obtains a solution which covers all elements by $k+1$ sets, it randomly drops a set and moves on to OCP$_k$.

In order to effectively solve each subproblem, we relax the coverage constraints and impose penalty for each uncovered element to the objective function. Specifically, let W_e be the weight of element e, the objective function $f(X)$ can be defined as Equation (2), which is the sum of the weights of each uncovered element.

$$\min f(X) = \sum_{e \in E, B_e \cap X = \varnothing} W_e \tag{2}$$

Initially, the weight of each element is $W_e = 1$. The objective is equivalent to minimizing the number of uncovered elements. When the algorithm encounters stagnation, i.e., it failed to improve the objective value, we increase the weight of each uncovered element by one.

As the search progresses, the weights of some elements will become very large. It may misguide the search or cause integer overflow. Therefore, we introduce a smoothing procedure to overcome these issues. If the weight of any element exceeds a given threshold, the proposed algorithm will divide the weight of each element by two. Specifically, the threshold is set to 2^{16} in WVNS.

3.4 Neighborhood Structure and Evaluation

The neighborhood structure describes the adjacency relation between solutions in the solution space. Given the incumbent solution X, performing neighborhood move m produces a neighboring solution $X \oplus m$. The improvement of move m, denoted by $\Delta(m)$, is the difference of objective values between the two solutions, so that $f(X \oplus m) = f(X) + \Delta(m)$.

In order to tackle each k-set covering sub-problem, the proposed local search procedure adopts a swap-based neighborhood structure. At each iteration, it tries to improve the current solution X by picking a set $p \in S \setminus X$ and dropping a set $q \in X$ to obtain a neighboring solution $X \cup \{p\} \setminus \{q\}$. The proposed algorithm follows the best-improvement policy, i.e., it evaluates all neighborhood moves and performs the one which leads to the best neighboring solution. In order to avoid re-evaluating the just-visited solutions, we employ a simple tabu strategy to diversify the search. Specifically, the sets which are just dropped or picked will not be taken into account at the next iteration.

Instead of naively evaluating each neighboring solution by Equation (2), WVNS adopts an incremental evaluation technique inspired by Zhang et al. (2020). In other words, we maintain the improvement $\Delta(p)$ of picking a set $p \in S \setminus X$ so that we can evaluate the neighboring solution by $f(X \oplus p) = f(X) + \Delta(p)$. The corresponding notations for dropping a set $q \in X$ can fit into the above ones without ambiguity. In the current solution, the elements can be partitioned into three groups, which are uncovered (G_0), covered by exact one set (G_1), and covered by more than one sets (G_2), respectively. Apparently, picking a set p will make all elements in $C_p \cap G_0$ covered, and dropping a set q will make all elements in $C_q \cap G_1$ uncovered. Therefore, we can calculate the incremental values as Equations (3) and (4). Note that performing a neighborhood move will change the belonging groups of some elements, so we need to track these elements and update the incremental values accordingly.

$$\Delta(p) = -\sum_{e \in C_p \cap G_0} W_e, \quad \forall p \in S \setminus X \tag{3}$$

$$\Delta(q) = \sum_{e \in C_q \cap G_1} W_e, \quad \forall q \in X \tag{4}$$

3.5 Neighborhood Reduction

Empirically, larger neighborhood leads to better solution quality. Unfortunately, it usually results in poor performance because larger neighborhood requires more time-consuming evaluation. Therefore, a balanced neighborhood structure is critical to the performance of local search algorithms.

In the classical solution methods for USCP like RWLS (Gao et al. 2015), the picking and dropping moves are evaluated separately. That is, they perform the best picking move out of $O(|S \setminus X|)$ ones and the best dropping move out of $O(|X|)$ ones in turn. As a result, there are totally $O(|S \setminus X| + |X|)$ evaluated moves at each iteration. However, WVNS considers the pick-drop pair as a whole, which dramatically expand the size of the neighborhood to $O(|S \setminus X| \times |X|)$. Regarding the large scale instances in real-world applications where there are millions of sets and thousands of picked sets, evaluating all valid moves is impractical. Thus, we need to reduce the neighborhood to reach a trade-off between the computational efficiency and solution quality. For convenience, we denote the set of swap moves which consists of picking a set in P and dropping a set in Q by $M(P, Q)$. For example, we can represent the moves in the complete swap neighborhood by $M(S \setminus X, X)$.

On the one hand, we restrict the candidate sets to pick by focusing on a single uncovered element at a time. In detail, at each iteration, WVNS selects an uncovered element e^* uniformly at random, and only considers picking each set

$p \in B_{e^*}$ which covers element e^*. This technique reduces the complete neighborhood to $M(B_{e^*}, X)$, which improves both the efficiency and diversification of the search.

On the other hand, we require that the sets to drop are closely related to element e^*. This idea is based on the conjecture that, it is usually hard to improve the current solution by swapping a pair of distant sets, especially when the current solution is almost a feasible cover and the picked sets rarely overlap. Specifically, the proposed algorithm only considers dropping set $s \in X \cap N^3_{e^*}$, i.e., the neighborhood becomes $M(B_{e^*}, X \cap N^3_{e^*})$. Recall that $N^3_{e^*}$ contains all the sets which are at most three hops to element e^* in the bipartite graph representation, which means $\forall s \in N^3_{e^*}, \exists s' \in B_{e^*}, C_s \cap C_{s'} \neq \varnothing$. The performance gain brought by this reduction technique is especially significant when the coverage of each set is much smaller than the universal set, at the price of neglecting some potential promising moves.

The above reduction technique improves the performance by ignoring the moves in $M(B_{e^*}, X \setminus N^3_{e^*})$ comparing to $M(B_{e^*}, X)$. However, we are able to efficiently find the best move among $M(B_{e^*}, X \setminus N^3_{e^*})$ by utilizing the concept of independent moves.

Definition 1. *A pair of neighborhood moves m_1 and m_2 to be performed on the incumbent solution X are independent if $f(X \oplus m_1 \oplus m_2) = f(X) + \Delta(m_1) + \Delta(m_2)$.*

Definition 1 describes a common phenomenon in local search algorithms that, after performing a move, the improvements of some other moves may not change.

Proposition 1. *In USCP, a pair of moves is independent if the involved sets do not cover any common elements.*

Proof. Let the involved sets be p and q. According to Equations (3) and (4), the value of $\Delta(p)$ only depends on the covering states of elements in C_p. Since $X \oplus q$ only affects the covering states of elements in C_q and $C_p \cap C_q = \varnothing$, $\Delta(p)$ is the same for both X and $X \oplus q$. Thus, $f(X \oplus p \oplus q) = f(X \oplus q) + \Delta(p) = f(X) + \Delta(p) + \Delta(q)$. □

Proposition 2. *In USCP, picking a set $p \in B_{e^*}$ is independent of dropping a set $q \in X \setminus N^3_{e^*}$.*

Proof. Assume that sets p and q cover some common elements, i.e., $C_p \cap C_q \neq \varnothing$. Then, these two sets can reach each other within two hops in the bipartite graph representation. Meanwhile, set $p \in B_{e^*}$ means set p is adjacent to element e^*, so element e^* can reach set q within three hops by $e^* \rightarrow p$ (one hop) and $p \rightarrow q$ (two hops). That is to say, $q \in N^3_{e^*}$ is true, which contradicts $q \in X \setminus N^3_{e^*}$. So, sets p and q do not cover any common elements. According to Proposition 1, these two moves are independent. □

Recall the toy instance illustrated in Figure 1, according to Propositions 1 or 2, the objective value increments of picking/dropping set s_1 is independent of picking/dropping set s_4, no matter what the picking states of other sets are.

Corollary 1. *In USCP, if $p^* = \operatorname{argmin}_{p \in B_{e^*}} \Delta(p)$ and $q^* = \operatorname{argmin}_{q \in X \setminus N^3_{e^*}} \Delta(q)$, then $\Delta(p^* \oplus q^*) = \Delta(p^*) + \Delta(q^*) \leq \Delta(p) + \Delta(q) \leq \Delta(p \oplus q), \forall p \in B_{e^*}, \forall q \in X \setminus N^3_{e^*}$.*

Corollary 1 is a natural extension of Proposition 2. It points out that we do not need to enumerate all $O(|B_{e^*}| \times |X \setminus N^3_{e^*}|)$ pick-drop pairs in $M(B_{e^*}, X \setminus N^3_{e^*})$ to find the best swap move. Instead, it can be done in $O(|B_{e^*}| + |X \setminus N^3_{e^*}|)$ time by directly combining the best picking move and the best dropping move. Since $M(B_{e^*}, X) = M(B_{e^*}, X \cap N^3_{e^*}) \cup M(B_{e^*}, X \setminus N^3_{e^*})$, we can find out the best move in $M(B_{e^*}, X)$ in $O(|B_{e^*}| \times |X \cap N^3_{e^*}| + |B_{e^*}| + |X \setminus N^3_{e^*}|)$ time if evaluating each move takes constant time.

Although the aforementioned acceleration technique significantly reduces the time complexity for evaluating $M(B_{e^*}, X)$, it is still slower than $M(B_{e^*}, X \cap N^3_{e^*})$. In order to balance between the effectiveness and efficiency, WVNS evaluates $M(B_{e^*}, X \setminus N^3_{e^*})$ only if $M(B_{e^*}, X \cap N^3_{e^*})$ contains no improving move to the current solution. That is to say, our algorithm actually explores two neighborhoods $M(B_{e^*}, X \cap N^3_{e^*})$ and $M(B_{e^*}, X)$ alternatively. This is the reason we call our algorithm weighting-based variable neighborhood search.

4 Experiments and Analysis

4.1 Experimental Protocol

In order to assess the effectiveness of the proposed WVNS algorithm, we conduct extensive experiments on 69 instances used in GECCO 2020 Competition on OCP and USCP, and compare our outcomes with the state-of-the-art algorithms in the literature and the participating solvers in the competition. There are 32 academic instances (AC) and 37 real-world ones (RW) in the GECCO dataset[1]. In addition, 18 out of the 32 AC instances were tested by Brévilliers et al. (2018), and the RW instances are similar to those tackled in Kritter et al. (2019). There are up to four million sets and one million elements in each instance. Moreover, there are two tracks in the competition. In the USCP track, the optimal camera placement problem is modeled into pure unicost set covering problem. In the OCP track, additional geometric information about the sample points and candidate camera positions is provided.

The experiments are run on a server equipped with 2.6GHz CPU and 128GB RAM. We perform four independent runs under 1,000-second time limit on a single CPU core, which is the same as adopted in Brévilliers et al. (2018). The connected component reduction rule is implemented by running exact mixed-integer linear programming (MILP) solver Gurobi 9.0.1 (Gurobi Optimization, LLC 2019) on each subproblem under one-second time limit and fixing the solutions of the subproblems which are solved to optimal. In addition, since the instances used in the GECCO 2020 Competition on OCP and USCP are really large and hard, there is no limit on hardware and computational time in the competition, in order to allow the competitors to fully exploit their potentials. So, we also run WVNS with the time limit of six hours for each RW instance, and with the time limit of one day and up to 16 cores along with GPU to accelerate the neighborhood evaluation for each AC instance. The experimental results obtained by WVNS under the extended

[1] http://www.mage.fst.uha.fr/brevilliers/gecco-2020-ocp-uscp-competition/

Algorithm	Characteristic
DEsim	Hybrid differential evolution algorithm (Brévilliers et al. 2018)
MILP+LNS	Hybridization of a MILP solver and a large neighborhood search
MSEC	Local search
SSHH	Sequence-based selection hyper-heuristic
ACO	Ant colony optimization

Table 1: Characteristics of reference algorithms.

Inst.	LB	MILP +LNS	MSEC	SSHH	DEsim	WVNS			WVNS-e			WVNS-g		
						UB	CPU	Hit	UB	CPU	Hit	UB	CPU	Hit
AC01	**7**	**7**	**7**	**7**	**7**	**7**	1	4/4	**7**	1	4/4	**7**	1	64/64
AC02	**4**	**4**	**4**	**4**	**4**	**4**	1	4/4	**4**	1	4/4	**4**	1	64/64
AC03	**3**	**3**	**3**	**3**	**3**	**3**	1	4/4	**3**	1	4/4	**3**	1	64/64
AC04	**5**	**5**	**5**	**5**	**5**	**5**	1	4/4	**5**	1	4/4	**5**	1	64/64
AC05	**7**	**7**	**7**	**7**	**7**	**7**	1	4/4	**7**	1	4/4	**7**	1	64/64
AC06	6	**10**	**10**	11	**10**	**10**	1	4/4	**10**	1	4/4	**10**	1	64/64
AC07	9	**17**	**17**	**17**	19	**17**	1	4/4	**17**	1	4/4	**17**	1	64/64
AC08	13	26	26	27	30	**25**	106	4/4	**25**	109	4/4	**25**	111	64/64
AC09	-	38	38	41	-	**36**	123	4/4	**36**	133	4/4	**36**	128	64/64
AC10	18	**20**	**20**	**20**	21	**20**	1	4/4	**20**	1	4/4	**20**	1	64/64
AC11	54	66	65	66	70	**64**	338	4/4	**64**	392	4/4	**64**	484	64/64
AC12	111	145	138	142	149	138	432	4/4	**136**	68169	2/4	**136**	13648	58/64
AC13	-	249	233	249	262	241	866	2/4	**232**	6539	4/4	**232**	3002	64/64
AC14	-	379	357	409	414	375	693	3/4	354	83490	1/4	**353**	6636	53/64
AC15	-	542	505	598	600	540	998	1/4	502	78655	1/4	**501**	2409	49/64
AC16	-	948	875	1059	1043	966	989	1/4	869	82296	1/4	**868**	5793	45/64
AC17	-	1486	1348	1651	1601	1523	999	1/4	1334	80392	1/4	**1334**	5361	52/64
AC18	-	2122	2055	2373	2277	2228	974	1/4	1907	82042	1/4	**1906**	14546	37/64
AC19	-	2862	2869	3178	3104	3097	990	1/4	2573	83025	1/4	**2571**	377	23/64
AC20	-	3745	3772	4109	-	4061	977	1/4	3336	84176	1/4	**3335**	4302	34/64
AC21	-	4718	4801	5263	-	5103	966	1/4	4212	85292	1/4	**4211**	2682	16/64
AC22	-	5828	5986	6397	-	6300	974	1/4	5176	77848	1/4	**5175**	5106	11/64
AC23	-	7022	7303	7757	-	7636	965	1/4	6239	85307	1/4	**6237**	6841	8/64
AC24	-	8346	8783	9216	-	9109	972	1/4	7421	79020	1/4	**7415**	4294	3/64
AC25	-	9820	10385	10775	-	10706	968	1/4	**8686**	79032	1/4	**8686**	5223	1/64
AC26	-	11372	12125	12465	-	12441	928	1/4	10041	83915	1/4	**10039**	3235	2/64
AC27	-	13041	13943	14378	-	14316	940	1/4	11533	77154	1/4	**11520**	8893	3/64
AC28	-	14803	16019	16277	-	16283	983	1/4	13106	77266	1/4	**13096**	11818	1/64
AC29	-	16734	18150	18407	-	18396	972	1/4	14752	82248	1/4	**14740**	6460	1/64
AC30	-	18768	20356	20649	-	20678	912	1/4	16534	80042	1/4	**16524**	17121	1/64
AC31	-	20917	22790	22892	-	23053	821	1/4	18405	77228	1/4	**18400**	16537	1/64
AC32	-	23162	25239	25243	-	25532	903	1/4	20358	85231	1/4	**20342**	6057	2/64
#best	-	8	8	7	6	11	-	-	14	-	-	32	-	-
Gap (%)	-	7.96	9.14	15.37	10.34	12.15	-	-	0.04	-	-	0.00	-	-

Table 2: Experimental results on AC instances.

Inst.	LB	MILP +LNS	MSEC	SSHH	ACO	WVNS			WVNS-e		
						UB	CPU	Hit	UB	CPU	Hit
RW01	660	665	665	676	928	**664**	764	2/4	**664**	11634	3/4
RW02	728	738	738	814	1031	737	133	4/4	**736**	2943	4/4
RW03	744	**745**	**745**	751	1021	**745**	8	4/4	**745**	3	4/4
RW04	824	833	833	919	1170	**832**	35	4/4	**832**	90	4/4
RW05	933	**934**	**934**	953	1240	**934**	12	4/4	**934**	33	4/4
RW06	926	**938**	**938**	1062	1321	**938**	95	4/4	**938**	111	4/4
RW07	985	**989**	**989**	1043	1365	**989**	20	4/4	**989**	5	4/4
RW08	1015	1026	1027	1182	1462	**1025**	276	4/4	**1025**	378	4/4
RW09	963	966	966	1009	1316	**965**	101	4/4	**965**	28	4/4
RW10	1025	1036	1036	1163	1413	**1034**	262	4/4	**1034**	205	4/4
RW11	314	**316**	**316**	321	438	**316**	4	4/4	**316**	5	4/4
RW12	315	322	321	328	455	**320**	289	4/4	**320**	2075	4/4
RW13	1292	1301	1302	1433	1828	1300	148	4/4	**1299**	11734	1/4
RW14	**337**	**337**	**337**	340	455	**337**	8	4/4	**337**	27	4/4
RW15	**341**	**341**	342	354	481	**341**	40	4/4	**341**	45	4/4
RW16	506	**508**	**508**	511	682	**508**	3	4/4	**508**	4	4/4
RW17	515	**516**	517	534	730	**516**	31	4/4	**516**	134	4/4
RW18	**338**	**338**	**338**	342	461	**338**	3	4/4	**338**	2	4/4
RW19	346	348	348	355	485	**347**	25	4/4	**347**	331	4/4
RW20	1458	**1466**	**1466**	1582	1971	**1466**	39	4/4	**1466**	29	4/4
RW21	1565	1581	1582	1839	-	1581	628	2/4	**1580**	4673	3/4
RW22	**398**	**398**	399	404	531	**398**	93	4/4	**398**	281	4/4
RW23	415	**417**	**417**	425	588	**417**	10	4/4	**417**	19	4/4
RW24	880	**886**	**886**	897	1205	**886**	25	4/4	**886**	17	4/4
RW25	945	951	952	1064	1328	951	87	4/4	**950**	3786	3/4
RW26	**464**	**464**	**464**	470	609	**464**	2	4/4	**464**	1	4/4
RW27	485	**489**	**489**	497	694	**489**	14	4/4	**489**	7	4/4
RW28	645	**648**	**648**	656	881	**648**	9	4/4	**648**	1	4/4
RW29	736	**741**	743	799	1021	**741**	185	4/4	**741**	233	4/4
RW30	1085	1096	1096	1177	1534	**1095**	92	4/4	**1095**	375	4/4
RW31	1176	1202	1201	1401	-	1199	426	4/4	**1198**	6503	3/4
RW32	648	**651**	**651**	657	876	**651**	10	4/4	**651**	7	4/4
RW33	709	720	721	791	1019	**719**	280	4/4	**719**	405	4/4
RW34	606	**609**	611	618	841	**609**	45	4/4	**609**	435	4/4
RW35	656	669	670	710	931	**668**	85	4/4	**668**	134	4/4
RW36	**609**	**609**	**609**	615	845	**609**	16	4/4	**609**	16	4/4
RW37	633	**644**	645	702	911	**644**	406	4/4	**644**	299	4/4
#best	-	22	16	0	0	32	-	-	37	-	-
Gap (%)	-	0.08	0.12	5.66	38.28	0.01	-	-	0.00	-	-

Table 3: Experimental results on RW instances.

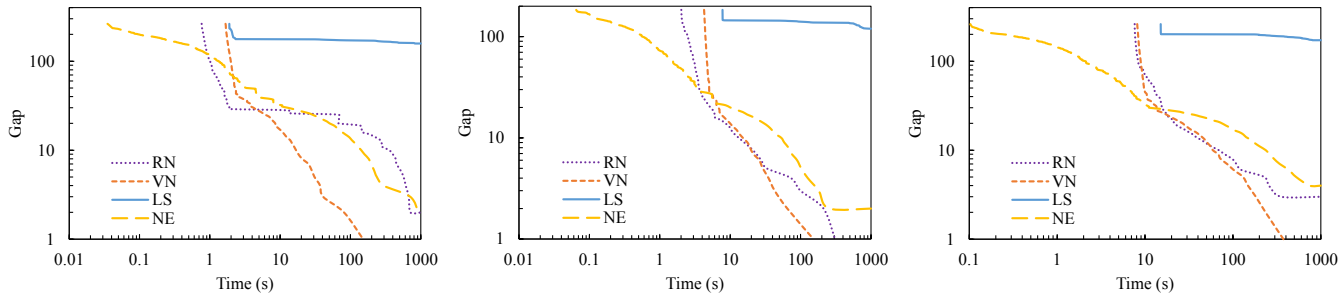
resource conditions are denoted by WVNS-e. Furthermore, we also report the computational results of WVNS on the AC instances when the geometric information is available. These results are denoted by WVNS-g and obtained by performing 64 independent runs under six-hour time limit.

The characteristics of the reference algorithms are listed in Table 1. DEsim in Brévilliers et al. (2018) is the best algorithm for OCP in the literature to the best of our knowledge. MILP+LNS, MSEC, SSHH, and ACO are the the most competitive participants in the GECCO 2020 Competition on OCP and USCP. From Table 1 we can observe that, state-of-the-art algorithms usually use metaheuristics or the hybridation of mathematical programming and metaheuristics to solve OCP and USCP.

4.2 Computational Results

Tables 2 and 3 report the computational results obtained by the proposed algorithm. Columns LB and UB present the objective values, i.e., the number of sets to pick to cover all elements. In detail, column LB gives the lower bound of each instance calculated by the MILP solver Gurobi 9.0.1 (Gurobi Optimization, LLC 2019) in two hours, and the numbers in bold indicate the optimal objective value, i.e., the lower bound matches the best known upper bound. Column DEsim reports the results obtained by Brévilliers et al. (2018) executed with a time limit of 1,000 seconds on Intel Core i5-3330 3.00GHz CPU and 4GB RAM. Columns MILP+LNS, MSEC, SSHH, and ACO present the results of the corresponding algorithms reported in the GECCO 2020 Competition on OCP and USCP, respectively. As there is no time limit in the competition, the competitors are able to submit the best solutions they can obtain ever. That is to say, they are free to keep running their algorithms until they think more CPU time will not help improving the results. Column UB represents the best upper bound obtained by WVNS, and the numbers in bold stand for the best known results. Column CPU reports the shortest computational time in seconds to hit the best upper bounds. Column Hit shows the hit rate of the presented results by corresponding algorithms. Row #best gives the number of instances for which the corresponding algorithm obtains the best results among all algorithms. Row Gap shows the average gap between the best results obtained by each algorithm and the best results.

From Table 2 we can observe that, the MILP solver only proves the optimality for the five smallest instances, and fails to solve the linear relaxation to obtain the initial lower bounds on 21 instances. These results indicate that the AC



(a) RW13. (b) RW25. (c) RW31.

Figure 3: Evolution of the objective value gaps by integrating different neighborhood reduction policies.

instances are very challenging. The proposed WVNS algorithm matches or outperforms DEsim on every public instance in the literature within 1,000 seconds and the improvements can be 10% on some instances. Moreover, when the time limit is extended, the advantage of WVNS expands to 17% on the largest instance tested by DEsim. Regarding the participants in the competition, although most of them also outperform the best results in the literature, WVNS-e still keeps over 7% advantage on average. In addition, the gaps become larger as the scale of the instances grows. These results indicate that WVNS is not only efficient to obtain good solutions quickly, but also very effective and scalable to search for high-quality solutions. Moreover, when the geometric information is available, WVNS is able to further improve the best known upper bounds on 18 instances, which justifies the effectiveness of the tiling method.

Table 3 reports the results on RW instances. There are six instances whose best known solutions are proven to be optimal, and the proposed algorithm is able to obtain such results within 300 seconds. For the remaining instances, the lower bounds are very close to the best known upper bounds, which means that the room for improvement is relatively small, and even a tiny advance may require huge efforts. As we can see from the table, WVNS keeps its advantage on these real-world datasets. In detail, WVNS obtains the best known results on every RW instance, and it still dominates the reference algorithms even if the time limit is restricted to 1,000 seconds. Again, these results justify the efficiency and effectiveness of WVNS.

4.3 Analysis on Neighborhood Evaluation

In this section, we investigate how different neighborhood evaluation strategies influence the performance of WVNS. We include four configurations in this comparison.

- **RN**: Only evaluate $M(B_{e^*}, X \cap N^3_{e^*})$.
- **VN**: Evaluate $M(B_{e^*}, X \setminus N^3_{e^*})$ when $M(B_{e^*}, X \cap N^3_{e^*})$ contains no improving move to the current solution.
- **LS**: Same as VN but the weighting technique is disabled.
- **NE**: Evaluate $M(B_{e^*}, X)$ in the naive way.

In detail, RN only evaluates the reduced neighborhood, which is fast but shortsighted. VN is the variable neighborhood in the final version of WVNS which adaptively evaluates the larger neighborhood in an effective manner. LS modifies the objective function that the quality of a solution is evaluated by the number of uncovered elements instead of Equation (2). NE evaluates the swap neighborhood without utilizing the dominance and independence properties. In order to make the comparison clearer, we conduct experiments on three hard RW instances (RW13, RW25, and RW31) which are not solved to the best known results within 1,000 seconds. Figure 3 shows the evolution of the objective value gaps by integrating different neighborhood evaluation strategies into WVNS. Each point (x, y) on the curves represents that the gap between the set number of the current solution and the best known one is y at x seconds.

Figure 3 shows that VN is able to obtain better results in shorter time on the tested representative instances comparing to other implementations. Although initializing the data structures for the reduction takes some time, VN can overtake the naive implementation NE within 20 seconds. In addition, by comparing VN and RN we observe that, the benefit of evaluating larger neighborhood is able to compensate for the performance loss. These phenomena indicate that variable neighborhood is quite effective and the proposed acceleration technique significantly improves the performance of WVNS. Furthermore, we can observe that the weighting technique is essential to WVNS that the configuration LS quickly converges to low-quality solutions. Thus, the weighting technique is a very successful diversification strategy to guide the search to escape from the local optima.

5 Conclusion

We propose an effective weighting-based variable neighborhood search algorithm for solving the optimal camera placement and unicost set covering problem. WVNS adopts several reduction techniques which can be applied to simplify the problem instances and accelerate the neighborhood evaluation. The proposed algorithm improves the best known results for 12 classical instances in the literature and obtains highly competitive results on the 69 benchmark instances used in the GECCO 2020 Competition on OCP and USCP. In the future, we will further explore the dominance and independence properties in the local search algorithms for other combinatorial optimization problems to enhance their search efficiency and effectiveness.

References

Beasley, J. 1987. An algorithm for set covering problem. *European Journal of Operational Research* 31(1): 85–93.

Beasley, J. E. 1990. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society* 41(11): 1069–1072.

Brévilliers, M.; Lepagnot, J.; Idoumghar, L.; Rebai, M.; and Kritter, J. 2018. Hybrid differential evolution algorithms for the optimal camera placement problem. *Journal of Systems and Information Technology* 20(4): 446–467.

Cai, S.; Hou, W.; Wang, Y.; Luo, C.; and Lin, Q. 2020. Two-goal local search and inference rules for minimum dominating set. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 1467–1473. International Joint Conferences on Artificial Intelligence Organization.

Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence* 175(9-10): 1672–1696.

Caprara, A.; Toth, P.; and Fischetti, M. 2000. Algorithms for the set covering problem. *Annals of Operations Research* 98: 353–371.

Chi, Z.; Xuan, J.; Ren, Z.; Xie, X.; and Guo, H. 2017. Multi-Level Random Walk for Software Test Suite Reduction. *IEEE Computational Intelligence Magazine* 12(2): 24–33.

Demirović, E.; Schwind, N.; Okimoto, T.; and Inoue, K. 2018. Recoverable Team Formation: Building Teams Resilient to Change. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, 1362–1370. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gao, C.; Yao, X.; Weise, T.; and Li, J. 2015. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research* 246(3): 750–761.

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

Gurobi Optimization, LLC. 2019. Gurobi optimizer reference manual.

Kritter, J.; Brévilliers, M.; Lepagnot, J.; and Idoumghar, L. 2019. On the real-world applicability of state-of-the-art algorithms for the optimal camera placement problem. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, 1103–1108.

Kritter, J.; Brévilliers, M.; Lepagnot, J.; and Idoumghar, L. 2019. On the optimal placement of cameras for surveillance and the underlying set cover problem. *Applied Soft Computing* 74: 133–153.

Lan, G.; DePuy, G. W.; and Whitehouse, G. E. 2007. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research* 176(3): 1387–1403.

Liu, J.; Sridharan, S.; and Fookes, C. 2016. Recent Advances in Camera Planning for Large Area Surveillance: A Comprehensive Review. *ACM Computing Surveys* 49(1).

Lü, Z.; and Hao, J. K. 2010. A memetic algorithm for graph coloring. *European Journal of Operational Research* 203(1): 241 – 250.

Luo, C.; Su, K.; and Cai, S. 2012. Improving local search for random 3-SAT using quantitative configuration checking. In *Proceedings of the 20th European Conference on Artificial Intelligence*, 570–575. IOS Press.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI'93, 40–45. AAAI Press.

Musliu, N. 2006. Local search algorithm for unicost set covering problem. In Ali, M.; and Dapoigny, R., eds., *Advances in Applied Artificial Intelligence*, 302–311. Berlin, Heidelberg: Springer Berlin Heidelberg.

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

Page, L.; Brin, S.; Motwani, R.; and Winograd, T. 1999. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Rui, X.; Wan, X.; Sheng, H.; and Yi, J. 2017. Joint optimization of receiver placement and illuminator selection for a multiband passive radar network. *Sensors* 17(6): 1378.

Seda, M. 2007. Heuristic set-covering-based postprocessing for improving the Quine-McCluskey method. In Ardil, C, ed., *Proceedings of World Academy of Science, Engineering and Technology*, volume 23, 256–260.

Selman, B.; and Kautz, H. 1993. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, 290–295. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Steinbach, B.; and Posthoff, C. 2012. Improvements of the construction of exact minimal covers of Boolean functions. In Moreno-Díaz, R.; Pichler, F.; and Quesada-Arencibia, A., eds., *Computer Aided Systems Theory – EUROCAST 2011*, 272–279. Berlin, Heidelberg: Springer Berlin Heidelberg.

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

Zhang, Q.; Lü, Z.; Su, Z.; Li, C.; Fang, Y.; and Ma, F. 2020. Vertex weighting-based tabu search for p-center problem. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 1481–1487. International Joint Conferences on Artificial Intelligence Organization.