InfVC: An Inference-Enhanced Local Search Algorithm for the Minimum Vertex Cover Problem in Massive Graphs

Rui Sun[1,2], **Peiyan Liu**[1], **Yiyuan Wang**[1,2*], **Zhaohui Liu**[1], **Liping Du**[1], **Jian Gao**[1*]

[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, China
{ruisun, liupeiyan, wangyy912, liuch, duliping, gaojian}@nenu.edu.cn

Abstract

The minimum vertex cover (MVC) problem is a classic NP-hard combinatorial optimization problem with extensive real-world applications. In this paper, we propose an efficient local search algorithm, InfVC, to solve the MVC in massive graphs, which comprises three ideas. First, we introduce an inference-driven optimization strategy that explores better feasible solutions through inference rules. Second, we develop a structural-determined perturbation strategy that is motivated by the structure features of high-quality solutions, prioritizing high-degree vertices into the candidate solution to guide the search process to some potential high-quality search area. Third, we design a self-adaptive local search framework that dynamically balances exploration and exploitation through a perturbation management mechanism. Extensive experiments demonstrate that InfVC outperforms all the state-of-the-art algorithms on almost massive instances.

1 Introduction

Given an undirected graph $G = (V, E)$, a vertex subset $D \subseteq V$ is a vertex cover of G if every edge in G has at least one endpoint in D. The minimum vertex cover (MVC) problem is to obtain a vertex cover with the smallest size. As one of the Karp's 21 classic NP-complete problems [Karp, 1972], the MVC is pivotal in the field of combinatorial optimization. In practical applications, the MVC has been widely applied in various fields, such as cyber security [Javad-Kalbasi *et al.*, 2019] and sensor networks[Yigit *et al.*, 2021]. Because the MVC is equivalent to the maximum independent set (MIS) problem and the maximum clique (MC) problem [Jin and Hao, 2015], solving the MVC inherently addresses the challenges associated with MIS and MC problems. Thus, the MVC holds substantial significance in terms of combinatorial optimization problems.

The MVC algorithms primarily include exact, approximation, and heuristic algorithms. Exact algorithms for the MVC guarantee to obtain optimal solution, but their exponential time complexity renders them impractical for massive hard graphs due to the NP-harness of the MVC. The current best exact algorithm for solving the MVC is based on the branch-and-reduce method [Akiba and Iwata, 2016]. Since exact methods can hardly handle massive hard graphs, researchers have turned to approximation and heuristic algorithms as practical alternatives for approximating optimal solutions. Approximation algorithms provide a guaranteed approximation ratio that quantifies the gap between the solution generated by the algorithm and the optimal solution. These algorithms have been widely applied to solve the MVC [Feo *et al.*, 1994; Bouamama *et al.*, 2012]. However, previous studies highlight the inherent challenges of approximating the MVC, as it has been proven NP-hard to achieve an approximation ratio smaller than 1.3606 [Dinur and Safra, 2005]. In practice, approximation algorithms for the MVC usually generate low-quality approximation solutions.

Heuristic algorithms usually employ various ways to improve the solution produced by specific approximation algorithm, typically achieving high-quality results within a reasonable time frame. Among all the heuristic algorithms for the MVC, local search plays a particularly significant role, as it serves as a fundamental component across all the state-of-the-art heuristic algorithms [Cai *et al.*, 2017; Gao *et al.*, 2017; Weise *et al.*, 2019; Chen and Hao, 2019; Luo *et al.*, 2019; Quan and Guo, 2021; Gu and Guo, 2021; Zhou *et al.*, 2022; Zhang *et al.*, 2023; Liao *et al.*, 2023]. To efficiently address the MVC in massive hard graphs, these studies have proposed a range of notable strategies, such as the best from multiple selections (BMS) heuristic [Cai *et al.*, 2017; Gu and Guo, 2021; Zhang *et al.*, 2023], graph reduction [Cai *et al.*, 2017; Gu and Guo, 2021], parallel component search [Gu and Guo, 2021; Liao *et al.*, 2023], and perturbation strategy [Luo *et al.*, 2019; Liao *et al.*, 2023]. According to the literature, the best-performing heuristic algorithms for the MVC are QMeaMetaVC [Liao *et al.*, 2023] and PEAVC [Gu and Guo, 2021].

Despite various notable strategies that have been developed for the MVC, some key aspects remain under-explored. Specifically, existing algorithms have not considered the potential of using inference heuristics to directly improve the solution quality. Besides, the key characteristics for high-quality solutions for the MVC is not considered by previous

*Corresponding author

studies to develop strategies. Moreover, existing studies do limited work on the local search framework for the MVC. Based on the above considerations, we propose an inference-enhanced local search algorithm called InfVC to solve the MVC in massive graphs, which comprises three ideas.

First, we propose an inference-driven optimization strategy. Our experiments reveal that feasible solutions can often be optimized by making minor adjustments. This suggests improvements can be made without search procedure. Motivated by this, we conduct an in-depth exploration of the characteristics of the MVC and propose an inference-driven optimization approach, which directly improves the quality of a feasible solution. Furthermore, we extend this approach to the set cover problem, a generalized version of the MVC. Our experimental results reveal that the proposed approach is not only effective for MVC but also exhibits strong scalability.

Second, we develop a novel structural-determined perturbation strategy for the MVC. Unlike existing methods, which primarily rely on random vertex selection or high-score criteria, our perturbation approach incorporates the structural information of a given graph. It is inspired by a key observation: high-quality solutions tend to include vertices with higher degrees, while vertices with lower degrees are more likely to be excluded. Building on this insight, we propose two new vertex selection rules and develop a structural-determined perturbation strategy. The proposed strategy helps the search escape from local optima and directs it toward promising search regions.

Lastly, we introduce a self-adaptive local search framework, which organizes the search procedure into three phases: stochastic search phase, perturbation phase, and convergence phase. This framework incorporates a perturbation management mechanism that continuously monitors and evaluates the impact of perturbations throughout the search process and the current search state. Based on these evaluations, the mechanism adaptively decides when to apply the perturbation and transitions the algorithm to a specified search phase. Furthermore, in each distinct search phase, the proposed algorithm employs a specific search model, ensuring an effective search manner in a specified search scenario.

Building upon the above strategies, we develop the inference-enhanced local search algorithm InfVC [1]. Experimental results show that the proposed algorithm achieves the best results in almost all the instances, and all the proposed strategies play a critical role in InfVC.

In Section 2, we provide the necessary preliminaries for the MVC. Section 3 introduces our inference-driven optimization strategy. In Section 4, we present the structure-determined perturbation strategy. Following that, Section 5 details the self-adaptive local search framework. Section 6 describes the proposed InfVC algorithm. The experimental results are reported in Section 7. Finally, Section 8 concludes the paper.

2 Preliminaries

Given an undirected graph $G = (V, E)$, V and E represent the set of vertices and edges, respectively. For an edge $e = \{u, v\}$, u and v are referred to as its endpoints. If two vertices share an edge, they are considered neighbors. The set of all neighbors of a vertex $v \in V$ constitutes its neighborhood, denoted by $N(v)$. The closed neighborhood is defined as $N[v] = N(v) \cup \{v\}$. The degree of a vertex v is defined as $|N(v)|$. Given a vertex set $D \subseteq V$, an edge e is covered by D if at least one of its endpoints belongs to D; otherwise, e is uncovered. D is a vertex cover of the graph G if D covers all edges in E. The MVC aims to identify a vertex cover with the smallest size. Given a vertex set $S \subseteq V$, the induced subgraph $G[S] = (V_S, E_S)$ is a subgraph of G where $V_S = S$ and the edge set E_S consists of all edges in E whose endpoints are both contained in S. A clique is an induced subgraph of a graph such that all the vertices in this subgraph are connected with each other.

Local search algorithms for the MVC maintain a candidate solution $D \subseteq V$ during the search procedure. If D is a vertex cover of G, D is called a feasible solution. Given the candidate solution D, the number of uncovered edges for D is denoted by $uncov(D)$, and the scoring function of a vertex v is defined as follows:

$$\text{score}(v) = \begin{cases} |C_1|, & v \notin D \\ -|C_2|, & v \in D \end{cases}$$

where C_1 represents the set of edges that would switch from uncovered to covered after the vertex v is added and C_2 represents the set of edges that would switch from covered to uncovered after the vertex v is removed. Additionally, age(v) represents the number of search steps since v was last operated. During the local search, two sets are maintained for the candidate solution D, including $Set_0(D) = \{v \in D \mid score(v) = 0\}$ and $Set_{-1}(D) = \{v \in D \mid score(v) = -1\}$. For a vertex $v \in V$, $Relate(v) = \{u \in N(v) \mid u \in Set_{-1}(D)\}$ is defined as the set of vertices in $Set_{-1}(D)$ that are adjacent with v.

3 Inference-Driven Optimization Strategy

During the local search, feasible solutions often have potential for further improvement through minor adjustments. This suggests that even without local search, improving the solution quality may be possible. Motivated by the above consideration, we propose an inference-driven optimization strategy, which is seen as a direct way to improve the quality of a feasible solution.

3.1 Foundations of Inference-Driven Optimization

First, we define two types of vertices, i.e., potential addition vertex and critical addition vertex.

Definition 1 (Potential Addition Vertex)**.** *Given a graph $G = (V, E)$ and a candidate solution D, a vertex $v \in V \setminus D$ is a potential addition vertex if $|Relate(v)| \geq 2$.*

Definition 2 (Critical Addition Vertex)**.** *Given a graph $G = (V, E)$ and a candidate solution D, a vertex $v \in V \setminus D$ is a critical addition vertex if v is a potential addition vertex and $G[Relate(v)]$ is not a clique.*

Obviously, for a critical addition vertex v, since $G[Relate(v)]$ is not a clique, there must exist an unadjacent

[1] Source code and supplementary materials are available at https://github.com/yiyuanwang1988/InfVC

vertex pair in $Relate(v)$. Then, we present the following proposition.

Proposition 1. *Given a graph $G = (V, E)$ and a vertex cover D, if v is a critical addition vertex and two distinct vertices $u, w \in Relate(v)$ satisfy $(u, w) \notin E$, then $D \cup \{v\} \setminus \{u, w\}$ is a vertex cover.*

Proof. Given score$(u) = -1$ and score$(w) = -1$, it is trivially to prove that there must exist exactly one respective vertex in $N(u)$ and $N(w)$ that is outside from D. Given that $u, w \in$ Relate(v) and v is a critical addition vertex, it follows that v is the unique vertex in both $N(u)$ and $N(w)$ that is excluded from D. Because u and w are unadjacent, removing u and w from D results in two uncovered edges, i.e., (u, v) and (w, v). By adding v to D, these two edges are covered, ensuring that the updated solution $D \cup \{v\} \setminus \{u, w\}$ becomes a better feasible solution. □

3.2 Details of the Inference-Driven Optimization

Based on the above foundations, we present the inference-driven optimization algorithm InfOpt in Algorithm 1. First, we introduce the detailed information of this algorithm, followed by a discussion of its time complexity.

The InfOpt Algorithm

We utilize $CriAdd(D)$ to denote the set of critical addition vertices of D. After obtaining a feasible solution D, a random vertex $v \in CriAdd(D)$ is added and two random unadjacent vertices $\{u, w\} \subseteq Relate(v)$ are removed until $CriAdd$ is empty (Lines 1–3). Finally, the improved solution D is returned. Notably, the proposed inference-driven optimization strategy can directly enhance solution quality, without relying on any local search procedure. In practice, it typically reduces the size of feasible solutions to some extent.

Additionally, we extend the proposed inference-driven optimization strategy to the set cover problem. The details of this extension are provided in the supplementary material.

Time Complexity of the Inference-Driven Optimization Strategy

In the InfOpt algorithm, we continuously maintain the set of potential addition vertices and evaluate whether each potential addition vertex is a critical addition vertex. If so, we add a critical addition vertex v and randomly remove two unadjacent vertices from $Relate(v)$ until there is no critical addition vertex. But if a potential addition vertex v is determined to be not a critical addition vertex, we will no longer check this vertex unless $Relate(v)$ changes. Thus, the time complexity comprises three components: 1) initially, distinguishing the potential addition vertices; 2) checking whether the potential addition vertices are critical addition vertices; 3) swapping a critical addition vertex and the corresponding two vertices in Set_{-1} and maintaining potential addition vertices.

By using an array to store the vertices in $Set_{-1}(D)$ and another array to track the index of each vertex within $Set_{-1}(D)$, the time complexity for maintaining $Set_{-1}(D)$ is $O(1)$. Next, let $degree_{max}$ denote the maximum degree in the graph G. According to the definition of potential addition vertices, distinguishing these vertices incurs a time complexity of $O(|Set_{-1}(D)| \cdot degree_{max})$.

Algorithm 1: InfOpt

Input: A feasible solution D
Output: An improved solution D
1 **while** $CriAdd(D) \neq \emptyset$ **do**
2 select a random vertex $v \in CriAdd(D)$ and two random unadjacent vertices $u, w \in Relate(v)$;
3 $D := D \cup \{v\} \setminus \{u, w\}$;
4 return D;

For a potential addition vertex v, determining whether $Relate(v)$ forms a clique requires a time complexity of $O(degree_{max} \cdot |Relate(v)|) = O(degree^2_{max})$, because $|Relate(v)|$ is smaller than $degree_{max}$. Since the total number of critical addition vertices is smaller than $|Set_{-1}(D)|$, the time complexity to determine whether all potential addition vertices are critical addition vertices is $O(|Set_{-1}(D)| \cdot degree^2_{max})$.

Furthermore, adding a critical addition vertex v and removing two unadjacent vertices u and w from $Relate(v)$ requires a time complexity of $O(degree^2_{max})$ to maintain the set of potential addition vertices. This is because the score of v's neighbors changes to -1, which may necessitate finding a vertex outside D in $N(v)$ to maintain the set of potential addition vertices.

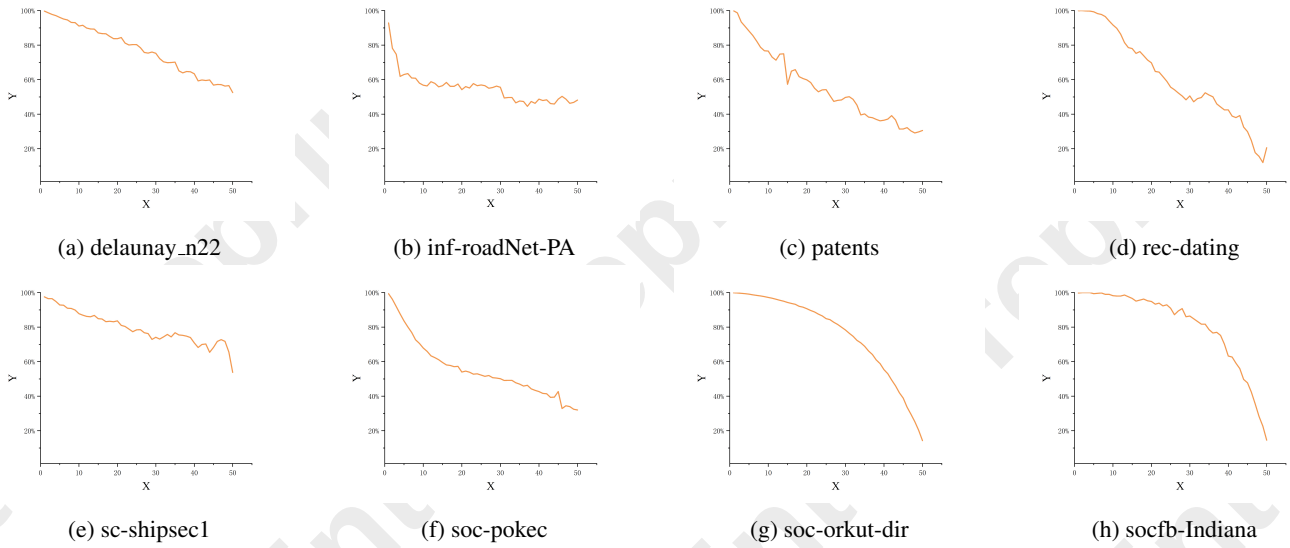
Based on the above analysis, the overall time complexity of InfOpt is $O(|Set_{-1}(D)| \cdot degree^2_{max})$.

4 Structural-Determined Perturbation Strategy

During the process of local search, perturbation procedures are commonly used to escape from local optima [Wang *et al.*, 2020; Jin *et al.*, 2021; Chen *et al.*, 2023]. In this section, we introduce the structural-determined perturbation strategy for MVC. Initially, we review existing perturbation strategies for the MVC. Subsequently, we present an observation from our experiments. Finally, we present the structural-determined perturbation strategy.

4.1 Previous Perturbation Strategies for MVC

Previous heuristic algorithms for the MVC have periodically invoked perturbation strategies to escape from local optima. In detail, two automatic configuration algorithms, MetaVC and MetaVC2 [Luo *et al.*, 2019], employ a simple perturbation strategy when dealing with massive graphs. Additionally, Q-MeaMetaVC [Liao *et al.*, 2023] utilizes three types of perturbation strategies to alter three different kinds of solutions, with one of these strategies being identical to that used by MetaVC and MetaVC2. However, these strategies either select vertices randomly or prioritize those with the highest $score$ values, but they overlook the structural distribution characteristics of high-quality solutions, potentially limiting their ability to guide the search toward more promising regions of the search space.



(a) delaunay_n22 (b) inf-roadNet-PA (c) patents (d) rec-dating

(e) sc-shipsec1 (f) soc-pokec (g) soc-orkut-dir (h) socfb-Indiana

Figure 1: An intuitive presentation of the characteristics of high-quality solutions on eight representative instances.

4.2 An Observation of Structural Characteristics of High-Quality Solutions

In our preliminary experiments, we observe a key structural characteristic in most graphs: vertices with high degrees are more likely to be included in high-quality solutions, whereas vertices with low degrees tend to be excluded. To illustrate this observation more intuitively, we select eight representative instances for detailed analysis. Specifically, we employ the state-of-the-art algorithm QMetaMeaVC [Liao *et al.*, 2023] to generate high-quality solutions for the eight instances, using a time limit of 10,000 seconds and a fixed seed of 1, and record the best solution achieved within this period. After obtaining the solutions, we rank the vertices of the graph in descending order by degree, dividing them into 50 equal segments, and present the proportion of the vertices belonging to the achieved high-quality solution in each segment. The results are presented in Figure 1, where the X-axis represents the segments arranged in descending order of degree, and the Y-axis shows the proportion of high-quality solutions within each segment. According to Figure 1, high-degree vertices tend to appear in the high-quality solutions achieved.

4.3 Details of Structural-Determined Perturbation Strategy

Based on the above observation, the structural-determined perturbation strategy is proposed. We adopt the best from multiple selections (BMS) heuristic [Cai, 2015] to select vertex, i.e., randomly sampling k vertices and then choosing the best one among them according to a specified criterion. The proposed perturbation selection rules are presented below.

Perturbation Addition Rule. sample k random vertices from the $V \setminus D$, and select the one with the highest degree, breaking ties with the highest age value.

Perturbation Removal Rule. select a random vertex in D as the best initial candidate vertex, and then iteratively samples $k-1$ extra random vertices. For each sampled vertex, if its degree is smaller than the current best vertex and its age is higher than the current best vertex, the current best vertex is updated to the sampled vertex.

Because high-degree vertices are more likely to appear in high-quality solutions, the sets of high-degree vertices across different high-quality solutions tend to be similar. Thus, in the proposed perturbation addition rule, we primarily select vertices with high degrees and then add them to the candidate solution. As for the operation of low-degree vertices, since these vertices are less frequently included in high-quality solutions, the sets of low-degree vertices across different high-quality solutions tend to be varied. Thus, we operate these vertices in a more diversified manner. The proposed perturbation removal rule equally takes both the degree and age of vertices into account.

The structural-determined perturbation strategy is depicted in Algorithm 2. It iteratively removes N vertices and then adds N vertices according to two perturbation rules, where N is a parameter (Lines 1–6). Finally, the perturbed solution is returned (Line 7). This perturbation method generates a solution that contains more high-degree vertices and fewer low-degree vertices, while ensuring a certain level of diversity.

5 Self-Adaptive Local Search Framework

In this section, we propose a self-adaptive local search framework. It organizes the local search procedure with three distinct phases: the stochastic search phase, the perturbation phase, and the convergence search phase. Initially, the algorithm goes into a stochastic search phase to explore the neighboring search space extensively. Subsequently, a perturbation phase is periodically triggered to facilitate escape from local optima. After the perturbation phase, the algorithm transitions to the convergence search phase, which aims to restore a desired level of convergence. Once this is achieved, the al-

Algorithm 2: StruPerturb

Input: A graph $G = (V, E)$, current solution D
Output: The solution D after perturbed

for $i = 1$ **to** N **do**
  select a vertex v_{rem} based on **Perturbation Removal Rule**;
  $D := D \setminus \{v_{rem}\}$;
for $i = 1$ **to** N **do**
  select a vertex v_{add} based on **Perturbation Addition Rule**;
  $D := D \cup \{v_{add}\}$;
return D;

gorithm transitions back to the stochastic search phase, continuing until the next perturbation phase is triggered.

The transitions between these phases are governed by a perturbation management mechanism, which continuously monitors the current search state and evaluates the impact of perturbations. Building up on this, it dynamically determines the appropriate search phase, enabling the framework to adapt to varying search conditions.

The proposed framework employs distinct search models for each phase. In the following, we begin by describing the vertex selection rules, which form the foundation of the proposed local search framework. Then, we provide a detailed explanation of the self-adaptive local search and the perturbation management mechanism.

5.1 Vertex Selection Rules

The vertex selection rules adopt the configuration checking (CC) strategy to forbid the cycling problem, where the CC is a technique specifically designed to prevent the cycling during the local search. The CC strategy associates each vertex with a binary indicator $conf$. For a vertex outside D, it can only be added back to D if its $conf$ value is 1. Detailed process of CC can be referred to [Cai *et al.*, 2011]. It has been proven that, under the CC method for the MVC, at least one endpoint of any uncovered edge must have a $conf$ value of 1 [Cai *et al.*, 2013].

During the stochastic search phase, introducing a certain degree of randomness can enhance exploration and help to escape from local optima. Based on this consideration, we propose two general vertex selection rules as follows:

General Removal Rule. With a small probability p, select a random vertex in D. Otherwise, sample k random vertices from D, and among these vertices select the one with the highest $score$ value, breaking ties by prioritizing the vertex with the highest age value.

General Addition Rule. Choose a random uncovered edge e. If one endpoint of e has a $conf$ value of 0, the other endpoint is selected. Otherwise, the endpoint with the higher $score$ is selected, breaking ties by prioritizing the vertex with the higher age value.

However, at the start of the convergence search phase, there often exists a large number of uncovered edges, requiring a quick recovery to restore convergence. To address this issue, we introduce the two greedy vertex selection rules as below:

Algorithm 3: SelfAdaptiveLS

Input: A graph $G = (V, E)$, the cutoff time *cutoff*
Output: the best obtained solution D^*

$D := D^* := ReduceInit(G)$;
$conv\_flag := conv\_indicator := 1$;
$per\_step := conv\_count := early\_count := 0$;
while $elapsed\_time <$ *cutoff* **do**
  if $uncov(D) = 0$ **then**
    D := InfOpt(D);
    while $Set_0(D) \neq \emptyset$ **do**
      select a random vertex $v \in Set_0(D)$;
      $D := D \setminus \{v\}$;
    $D^* := D$, $conv\_flag := 1$;
    select a vertex v according to the **General Removal Rule** and $D := D \setminus \{v\}$;
    continue;
  $\langle conv\_flag, D \rangle$:= PerManage($conv\_flag$, D);
  if $conv\_flag = 1$ **then**
    /* Stochastic Search Phase */
    select a vertex v_1 according to the **General Removal Rule** and $D := D \setminus \{v_1\}$;
    select a vertex v_2 according to the **General Addition Rule** and $D := D \cup \{v_2\}$;
  else
    /* Convergence Search Phase */
    select a vertex v_1 according to the **Greedy Removal Rule** and $D := D \setminus \{v_1\}$;
    select a vertex v_2 according to the **Greedy Addition Rule** and $D := D \cup \{v_2\}$;
return D^*;

Greedy Removal Rule. If $Set_0(D)$ or $Set_{-1}(D)$ is not empty, select the first non-empty set in the order of $Set_0(D)$ followed by $Set_{-1}(D)$. Sample k random vertices from the selected set and select the one with the highest age value, breaking ties randomly. Otherwise, select a removal vertex according to the general removal rule.

Greedy Addition Rule. Sample $\lfloor \sqrt{uncov(D)} \rfloor$ uncovered edges, and identify the endpoints of these edges with a $conf$ value of 1. Among these satisfied vertices, select the one with the highest $score$ value, breaking ties by prioritizing the vertex with the highest age value.

By leveraging these two kinds of selection rule in specific search phases, the local search framework dynamically adjusts its behavior to balance rapid convergence and effective exploration, thus improving search efficiency and avoiding stagnation in local optima.

5.2 The Self-Adaptive Local Search Algorithm

The details of the self-adaptive local search algorithm is presented in Algorithm 3. During the local search procedure, D and D^* are the candidate solution and the best solution during the local search procedure, whereas $conv\_flag$ represents the current search phase where $conv\_flag = 0$ represents the convergence phase and $conv\_flag = 1$ indicates the stochastic phase. In addition, four global variables need to be maintained, which play a critical rule in the perturbation management mechanism. They will be described in the next

subsection.

Initially, D and D^* are initialized by reduction based initialization strategy, which can be referred to [Fan *et al.*, 2015] (Line 1), and other variables are initialized accordingly (Lines 2–3). During the local search, the algorithm iteratively swaps a pair of vertices to improve the solution (Lines 4–20). In each iteration, the algorithm checks if $uncov(D)$ is zero. If so, the algorithm uses the proposed inference-driven optimization strategy to improve D (Line 6). Several redundant vertices are removed until $Set_0(D)$ becomes empty (Lines 7–9). Lastly, the D^* is updated by D, and the search phase is switched to the stochastic phase by setting $conv\_flag$ to 1, and a vertex is selected by the general removal rule to be removed (Lines 10–11).

Subsequently, the algorithm employs a perturbation management mechanism to assess the current search phase and determine whether to perturb the current solution (Line 13). If a perturbation phase is executed, D will be updated by the perturbed solution. This process will be described in the next subsection. Subsequently, if the current search phase is the stochastic phase, the algorithm would utilize the general selection rules to explore neighboring search space (Lines 14–16). Otherwise, the current search phase is the convergence search phase, and the algorithm utilizes the greedy selection rules to converge the solution quickly (Lines 17–19). By combining these distinct search phases, the framework effectively balances exploration and exploitation. Finally, D^* is returned (Line 20).

5.3 Perturbation Management Mechanism

In this subsection, we introduce the perturbation management mechanism. It continuously monitors and evaluates the effects of perturbations during the search process. Based on the evaluation, it dynamically determines the current search phase. We depict this procedure in Algorithm 4. It initially decides whether to transition from the convergence search phase to the stochastic search phase (Lines 1–4). Subsequently, it determines whether to shift from the stochastic search phase to the perturbation phase and then back to the convergence search phase (Lines 5–10).

In this procedure, T is a parameter that controls the execution of perturbations. We introduce four global variables: 1) $conv\_count$ denotes the total number of times the convergence phase has taken place; 2) per_step records the steps since the last perturbation; 3) $conv\_indicator$ represents the number of uncovered edges before the most recent perturbation; 4) $early\_count$ denotes the total number of times the early convergence phase has taken place; If the current search phase is the convergence search phase and $uncov(D)$ doesn't exceed $conv\_indicator$, it indicates the $uncov(D)$ have recovered to the level before the most recent perturbation. In this case, the search process is considered converged. if this convergence occurs within $\frac{T}{2}$ steps since the latest perturbation, we consider this convergence phase is an early convergence phase. If $early\_count$ is not larger than $\frac{conv\_count}{2}$, it indicates at least half of the convergence search phases are early convergence phase, indicating that the graph is easy to converge. Otherwise, the graph is considered challenging to converge.

Algorithm 4: PerManage

Input: Candidate solution D, search indicator $conv\_flag$
Output: Candidate solution D, search indicator $conv\_flag$

1 **if** $conv\_flag = 0 \wedge uncov(D) \leq conv\_indicator$ **then**
2 $conv\_flag := 1$;
3 **if** $per\_step \leq T/2$ **then**
4 $early\_count := early\_count + 1$;
5 **if** $per\_step \geq T$ **then**
6 **if** $conv\_flag = 1 \vee early\_count \leq \frac{conv\_count}{2}$ **then**
  /* Perturbation Phase */
7 $D :=$StruPerturb(G, D) and $conv\_flag := 0$;
8 $per\_step := 0$;
9 $conv\_count := conv\_count + 1$;
10 $conv\_indicator := uncov(D)$;
11 $per\_step := per\_step + 1$;
12 return $\langle conv\_flag, D \rangle$;

Initially, if the current search phase is convergence phase (i.e., $conv\_flag = 0$) and the convergence level has returned to the state before the latest perturbation (i.e., $uncov(D) \leq conv\_indicator$), then the algorithm shifts to the stochastic phase (Lines 1–2), i.e., setting $conv\_flag$ to 1. Moreover, if this convergence phase is an early convergence phase (i.e., $per\_step \leq \frac{T}{2}$), then $early\_count$ is incremented by one (Lines 3–4).

Next, the algorithm periodically determines whether the the perturbation procedure needs to be called (Line 5). If needed, the algorithm further evaluates whether the current search phase is the stochastic search phase (i.e., $conv\_flag = 1$) or the current graph is easy to converge (i.e., $early\_count \leq \frac{conv\_count}{2}$). If satisfied, the algorithm calls the *StruPerturb* and then shifts the search phase to the convergence search phase (Lines 7–8). After shifting to the convergence search phase, three global variables are updated accordingly (Lines 7–10). Finally, per_step is incremented by 1 (Line 11), and $\langle conv\_flag, D \rangle$ is returned.

6 The InfVC Algorithm

In this section, we introduce the framework of the InfVC algorithm in Algorithm 5.

Initially, the algorithm employs a state-of-the-art preprocessing strategy *PreP* to simplify the input graph G, generating a smaller graph G^*. The detailed preprocessing method can be referred to [Gu and Guo, 2021]. G^* comprises one or numerous connected components, and we utilize D_i to represents the solution of the ith component, and the union of the solutions of all components is stored in D_{all}.

For each connected component G_i of G, if $|G_i|$ is smaller than a certain threshold V_{thre}, a complete MVC solver *CPLEX* is utilized to solve it within a time limit of $\frac{|V_i|}{|V^*|} * \frac{cutoff}{2}$ where $\frac{|V_i|}{|V^*|}$ is the ratio of the vertices in the ith component to the vertices in the G^* (Line 4). We employ CPLEX 22.10 as the complete MVC solver to solve these small components. If the complete solver returns an empty solution, it means that the solution cannot be solved in the given time limit. Otherwise, it indicates the component is successfully solved. If so,

Algorithm 5: InfVC

Input: A graph $G = (V, E)$, the cutoff time *cutoff*
Output: The obtained best solution D_{final}

1 $G^* := PreP(G)$, $D_{all} := \emptyset$;
2 **foreach** *component* $G_i \in G^*$ **do**
3 **if** $|V_i| < T_{thre}$ **then**
4 $D_i := CPLEX(G_i, \frac{|V_i|}{|V^*|} * \frac{cutoff}{2})$;
5 **if** $D_i \neq \emptyset$ **then**
6 $D_{all} := D_{all} \cup D_i$;
7 remove G_i from G^*;
8 re-calculate the remaining time limit *cutoff*;
9 **foreach** *component* $G_i \in G^*$ **do**
10 $D_{all} := D_{all} \cup SelfAdaptiveLS(G_i, \frac{|V_i|}{|V^*|} * cutoff)$;
11 $D_{final} := InvPreP(D_{all})$;
12 **return** D_{final};

D_{all} and G^* are updated accordingly (Lines 5–7). According to our preliminary experiments, we set T_{thre} to 300.

After the above procedure, the algorithm needs to update the *cutoff* by excluding the the time consumption of calling the *Prep* and *CPLEX* (Line 8). After that, the algorithm conducts *SelfAdaptiveLS* on each component G_i, with the corresponding time limit (Lines 9–10). Finally, the algorithm reconstructs the solution using *InvPreP*, which is a dual technology of *PreP*, and it restores the vertex cover of G based on the solutions obtained from G^* (Line 11). The details of this process can be referred to [Gu and Guo, 2021]. Finally, the best solution D_{final} is returned (Line 12).

7 Experimental Evaluation

In this section, we evaluate the performance of InfVC and the proposed strategies.

We compare InfVC with the state-of-the-art heuristic algorithms for MVC, including FastVC2+p [Cai *et al.*, 2017], MetaVC [Luo *et al.*, 2019], MetaVC2 [Luo *et al.*, 2019], EAVC3+p [Quan and Guo, 2021], TIVC [Zhang *et al.*, 2023], PEAVC [Gu and Guo, 2021], and QMeaMetaVC [Liao *et al.*, 2023]. The source codes of FastVC2+p, MetaVC, MetaVC2, EAVC3+p are kindly provided by the authors. However, the source codes of TIVC, PEAVC, and QMeaMetaVC are not available for us, so we have to re-implement them. In addition to these comparisons, InfVC is also compared with the best-record results. Following the most recent and best-performing heuristic algorithm [Liao *et al.*, 2023] for the MVC, we set the time limit of all algorithms to 3600 seconds, with seeds ranging from 1 to 10. All the algorithms are implemented in C++ and compiled by g++ with '-O3' option. For all competitors, we utilize the same parameters as described in the corresponding literature and optimize these parameters for newly added instances. All experiments are run on an Intel Xeon Gold 6238 CPU @ 2.10GHz CPU with 512GB RAM under Ubuntu 22.04.4 LTS.

We utilize all massive graphs that were previously evaluated in previous studies. Specifically, 102 instances from the Network Data Repository [Rossi and Ahmed, 2015] have been utilized in numerous MVC studies [Cai *et al.*, 2017; Chen and Hao, 2019; Luo *et al.*, 2019; Quan and Guo, 2021; Gu and Guo, 2021; Liao *et al.*, 2023], as well as algorithms for other combinatorial optimization problems [Sun *et al.*, 2024; Luo *et al.*, 2024; Sun *et al.*, 2025]. In 2023, Zhang et al. [2023] selected 72 additional instances from the same repository. We select all the above instances. Note that there are 18 duplicate instances across the two benchmarks. In total, we selected 156 instances.

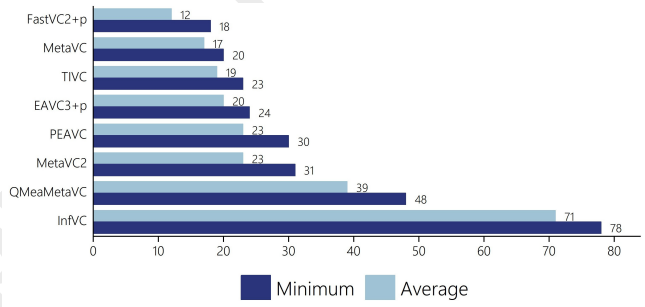


Figure 2: An overall comparison between InfVC and all comparative algorithms.

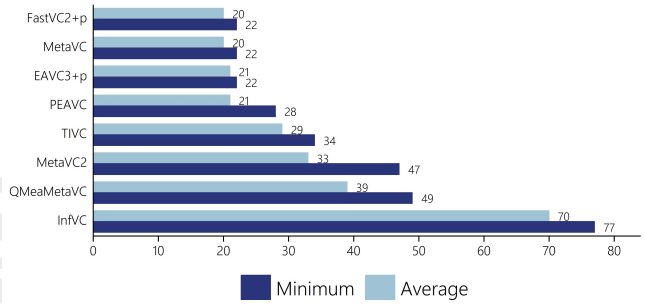


Figure 3: An overall comparison between InfVC and all comparative algorithms under the state-of-the-art preprocessing strategy.

By utilizing the automatic configuration tool irace [López-Ibáñez *et al.*, 2016], we tune the parameters of InfVC, and the results are presented in Table 1. Specifically, we construct a training set by randomly selecting 20 instances. The tuning process was allotted a budget of 1000 runs for this training set, with each run having a time limit of 3600 seconds. In addition, we conduct experiments to examine the parameter sensitivity of the InfVC algorithm. The results show that our algorithm exhibits low sensitivity to parameter variations. The best solution achieved using optimal parameter settings, is on average only 0.008% smaller than those obtained through other parameter combinations.

Parameter	Parameter Range	Final value
T	{30000, 34000, 38000, 42000}	38000
N	{65, 75, 85, 95}	85
p	{0.00045, 0.0005, 0.00055, 0.0006}	0.0005
k	{50, 250, 500, 750, 1000}	750

Table 1: Tuned parameters of our proposed algorithm.

#inst.	vs. InfVC1 #bet(#wor)	vs. InfVC2 #bet(#wor)	vs. InfVC3 #bet(#wor)	vs. InfVC4 #bet(#wor)	vs. InfVC5 #bet(#wor)	vs. InfVC6 #bet(#wor)	vs. InfVC7 #bet(#wor)	vs. InfVC8 #bet(#wor)
82	**17**(6)	**24**(6)	**23**(7)	**25**(8)	**12**(7)	**7**(2)	**21**(5)	**15**(7)

Table 2: Comparing InfVC with 8 modified versions. #bet and #wor represent respectively the number of instances where InfVC achieves better and worse minimal solutions.

7.1 Results on All Instances

Note that all the algorithms incorporate efficient preprocessing mechanisms from [Cai *et al.*, 2017] or its advanced version [Gu and Guo, 2021] to reduce the graph. After this reduction, many instances become trivially solvable. Specifically, 74 instances are reduced to fewer than 1,000 vertices, for which all algorithms produce the same solution across all seeds. We used CPLEX to verify that the solutions are optimal. Thus, we concentrate only on the remaining 82 instances.

We summarize the results of all algorithms in Figure 2, where Minimal and Average denote the number of instances where the corresponding algorithm finds the best minimal and average solutions among all algorithms. As shown in Figure 2, InfVC significantly outperforms the other heuristic algorithms. Specifically, it outperforms FastVC2+p, MetaVC, MetaVC2, EAVC3+p, TIVC, PEAVC, and QMeaMetaVC on 63, 59, 49, 57, 58, 50, and 34 instances, while only being outperformed by these seven algorithms on only 4 instances. Additionally, InfVC surpasses the best-record solutions on 31 instances, but is outperformed on 4 only instances. Detailed comparisons are provided in the supplementary material.

Note that PEAVC, QMeaMetaVC, and InfVC employ a more advanced preprocessing procedure [Gu and Guo, 2021] compared to other algorithms. To ensure a fair comparison of the local search effectiveness of InfVC, we apply the same preprocessing procedure to FastVC2+p, EAVC3+p, TIVC, MeataVC, and MetaVC2, and reevaluate the performance of all algorithms. The results are shown in Figure 3. After integrating the state-of-the-art preprocessing, the performance of InfVC remains largely unchanged. Specifically, the number of best minimal solutions and best average solutions decreases slightly, from 78 to 77 and 71 to 70, respectively.

7.2 Further Results with Exact MVC Algorithms

Additionally, we compare InfVC with the state-of-the-art exact algorithms for MVC. To the best of our knowledge, the best-performing exact algorithms for MVC are the algorithms proposed in [Akiba and Iwata, 2016]. In this work, the authors propose several state-of-the-art reduction methods for MVC and develop a branch-and-reduce algorithm B&R for solving MVC. Additionally, they utilize these reduction methods to preprocess graphs and employ the CPLEX to directly solve the MVC problem. Following their methodology, we compare InfVC with B&R and the CPLEX 22.10 [2], where the CPLEX is carried out on the graphs that are reduced based on the methods proposed in the study.

The time limit for all algorithms is set to 3600 seconds, and we compare the two exact algorithms with InfVC under the specified seed value 1. It is worth noting that even when CPLEX does not find an optimal solution, it still returns a feasible solution. In contrast, B&R does not output any solution if it fails to find the optimal one. Therefore, we present the best solutions found by InfVC and CPLEX within the 3600-second time limit. If B&R fails to find an optimal solution for a given instance, we mark it as "−". The detailed results are presented in the supplementary material. The results show that InfVC outperforms CPLEX and B&R in 41 and 48 instances, respectively, while being surpassed in only 5 and 3 instances. The results clearly highlight the effectiveness of InfVC.

7.3 Effectiveness of the Proposed Strategies

To evaluate the effectiveness of our proposed strategies, we generated several alternative versions: InfVC1 disables the inference-driven optimization strategy; InfVC2, InfVC3, and InfVC4, each substituting the proposed perturbation strategy with one of three perturbation strategies used in Q-

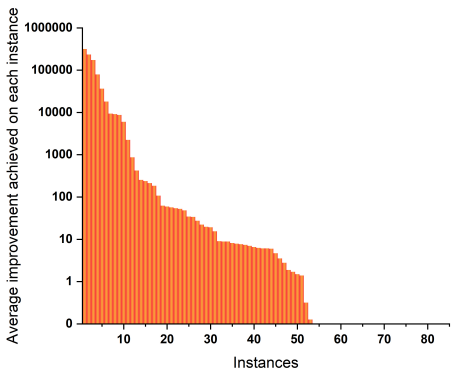


Figure 4: Average improvement obtained by the InfOpt.

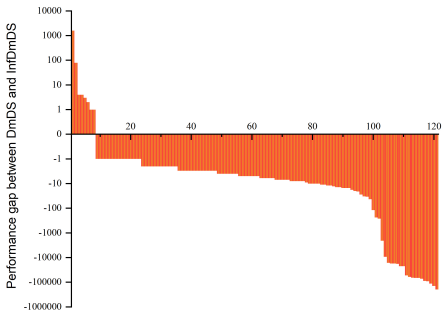


Figure 5: Comparison between InfDmDS and DmDS.

[2]https://www.ibm.com/products/software

MeaMetaVC, respectively; InfVC5 deletes the structural-determined perturbation strategy; InfVC6 directly executes the structure-determined perturbation strategy every T steps, that is, omitting Line 6 in Algorithm 4; InfVC7 ignores the convergence search phase; InfVC8 ignores the stochastic search. The results are presented in Table 2, which clearly indicates that all the proposed strategies are crucial in InfVC.

7.4 Further Analysis of the Inference-Driven Optimization Strategy

Figure 4 presents the average improvement achieved by the inference-driven optimization strategy for each instance, with each column representing the corresponding average improvement over 10 runs. The results clearly show that the proposed strategy delivers significant improvements across many instances.

Furthermore, we integrate the inference-driven optimization strategy into the minimum dominating set(MDS), which can be seen as a specific variant of the set cover problem. Specifically, we apply the proposed strategy to DmDS [Zhu *et al.*, 2024] that is the state-of-the-art heuristic algorithm for the MDS problem, resulting in a new algorithm, InfDmDS. We conducted experiments on 260 massive graphs used in [Zhu *et al.*, 2024]. Following the experimental settings of DmDS, we set the time limit to 1000 seconds and use random seeds ranging from 1 to 10. We present the results in Figure 5, where each column is the best solution obtained by DmDS minus the best solution obtained by InfDmDS for a specified instance. The scenarios in which the two algorithms achieve the same best solution are excluded. As depicted in Figure 5, InfDmDS significantly outperforms DmDS.

8 Conclusion

In this paper, we propose an inference-enhanced local search algorithm to tackle the MVC. It comprises three strategies: an inference-driven optimization strategy, a structural-determined perturbation strategy, and a self-adaptive local search framework. InfVC effectively balances exploration and exploitation, leveraging structural insights and adaptive mechanisms to improve the solution quality and escape from local optima. Experimental results demonstrate that the proposed algorithm significantly outperforms previous algorithms in massive graphs.

In future work, we plan to develop a pseudo-reduction based heuristic algorithm that integrates reduction techniques into the heuristic search process.

Acknowledgments

This work is supported by National Cryptologic Science Fund of China 2025NCSF02046, NSFC under Grant No. 61806050 and 61972063, the Fundamental Research Funds for the Central Universities 2412023YQ003, Jilin Science and Technology Department YDZJ202201ZYTS412.

References

[Akiba and Iwata, 2016] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

[Bouamama *et al.*, 2012] Salim Bouamama, Christian Blum, and Abdellah Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632–1639, 2012.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10):1672–1696, 2011.

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

[Cai *et al.*, 2017] Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In Qiang Yang and Michael J. Wooldridge, editors, *IJCAI*, pages 747–753, 2015.

[Chen and Hao, 2019] Yuning Chen and Jin-Kao Hao. Dynamic thresholding search for minimum vertex cover in massive sparse graphs. *Engineering Applications of Artificial Intelligence*, 82:76–84, 2019.

[Chen *et al.*, 2023] Jiejiang Chen, Shaowei Cai, Yiyuan Wang, Wenhao Xu, Jia Ji, and Minghao Yin. Improved local search for the minimum weight dominating set problem in massive graphs by using a deep optimization mechanism. *Artificial Intelligence*, 314:103819, 2023.

[Dinur and Safra, 2005] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[Fan *et al.*, 2015] Yi Fan, Chengqian Li, Zongjie Ma, LjiLjana Brankovic, Vladimir Estivill-Castro, and Abdul Sattar. Exploiting reduction rules and data structures: Local search for minimum vertex cover in massive graphs. *arXiv preprint arXiv:1509.05870*, 2015.

[Feo *et al.*, 1994] Thomas A Feo, Mauricio GC Resende, and Stuart H Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878, 1994.

[Gao *et al.*, 2017] Wanru Gao, Tobias Friedrich, Timo Kötzing, and Frank Neumann. Scaling up local search for minimum vertex cover in large graphs by parallel kernelization. In *AI 2017*, pages 131–143. Springer, 2017.

[Gu and Guo, 2021] Jiaqi Gu and Ping Guo. Peavc: An improved minimum vertex cover solver for massive sparse graphs. *Engineering Applications of Artificial Intelligence*, 104:104344, 2021.

[Javad-Kalbasi *et al.*, 2019] Mohammad Javad-Kalbasi, Keivan Dabiri, Shahrokh Valaee, and Ali Sheikholeslami.

Digitally annealed solution for the vertex cover problem with application in cyber security. In *ICASSP*, pages 2642–2646. IEEE, 2019.

[Jin and Hao, 2015] Yan Jin and Jin-Kao Hao. General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Engineering Applications of Artificial Intelligence*, 37:20–33, 2015.

[Jin *et al.*, 2021] Yan Jin, Bowen Xiong, Kun He, Jin-Kao Hao, Chu-Min Li, and Zhang-Hua Fu. Clustering driven iterated hybrid search for vertex bisection minimization. *IEEE Transactions on Computers*, 71(10):2370–2380, 2021.

[Karp, 1972] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Boston, MA, 1972.

[Liao *et al.*, 2023] Chunmei Liao, Ping Guo, Jiaqi Gu, and Qiuju Deng. Q-meametavc: An mvc solver of a large-scale graph based on membrane evolutionary algorithms. *Applied Sciences*, 13(14):8021, 2023.

[López-Ibáñez *et al.*, 2016] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Luo *et al.*, 2019] Chuan Luo, Holger H Hoos, Shaowei Cai, Qingwei Lin, Hongyu Zhang, and Dongmei Zhang. Local search with efficient automatic configuration for minimum vertex cover. In *IJCAI*, pages 1297–1304, 2019.

[Luo *et al.*, 2024] Chunyu Luo, Yi Zhou, Zhengren Wang, and Mingyu Xiao. A faster branching algorithm for the maximum k-defective clique problem. In *ECAI 2024*, pages 4132–4139. IOS Press, 2024.

[Quan and Guo, 2021] Changsheng Quan and Ping Guo. A local search method based on edge age strategy for minimum vertex cover problem in massive graphs. *Expert Systems with Applications*, 182:115185, 2021.

[Rossi and Ahmed, 2015] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, volume 29, 2015.

[Sun *et al.*, 2024] Rui Sun, Yiyuan Wang, HL Shimao Wang, Hui Li, Ximing Li, and Minghao Yin. Nukplex: An efficient local search algorithm for maximum k-plex problem. In *IJCAI*, pages 7029–7037, 2024.

[Sun *et al.*, 2025] Rui Sun, Yiyuan Wang, and Minghao Yin. Improving local search algorithms for clique relaxation problems via group driven initialization. *Frontiers of Computer Science*, 19(6):1–12, 2025.

[Wang *et al.*, 2020] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. Sccwalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280:103230, 2020.

[Weise *et al.*, 2019] Thomas Weise, Zijun Wu, and Markus Wagner. An improved generic bet-and-run strategy with performance prediction for stochastic local search. In *AAAI*, volume 33, pages 2395–2402, 2019.

[Yigit *et al.*, 2021] Yasin Yigit, Vahid Khalilpour Akram, and Orhan Dagdeviren. Breadth-first search tree integrated vertex cover algorithms for link monitoring and routing in wireless sensor networks. *Computer Networks*, 194:108144, 2021.

[Zhang *et al.*, 2023] Yu Zhang, Shengzhi Wang, Chanjuan Liu, and Enqiang Zhu. Tivc: An efficient local search algorithm for minimum vertex cover in large graphs. *Sensors*, 23(18):7831, 2023.

[Zhou *et al.*, 2022] Qian Zhou, Xiaojun Xie, Hua Dai, and Weizhi Meng. A novel rough set-based approach for minimum vertex cover of hypergraphs. *Neural Computing and Applications*, 34(24):21793–21808, 2022.

[Zhu *et al.*, 2024] Enqiang Zhu, Yu Zhang, Shengzhi Wang, Darren Strash, and Chanjuan Liu. A dual-mode local search algorithm for solving the minimum dominating set problem. *Knowledge-Based Systems*, page 111950, 2024.