An Efficient Local Search for the Minimum Independent Dominating Set Problem

Kazuya Haraguchi
Otaru University of Commerce, Midori 3-5-21/Otaru, Hokkaido, Japan
haraguchi@res.otaru-uc.ac.jp

Abstract

In the present paper, we propose an efficient local search for the minimum independent dominating set problem. We consider a local search that uses *k-swap* as the neighborhood operation. Given a feasible solution S, it is the operation of obtaining another feasible solution by dropping exactly k vertices from S and then by adding any number of vertices to it. We show that, when $k = 2$, (resp., $k = 3$ and a given solution is minimal with respect to 2-swap), we can find an improved solution in the neighborhood or conclude that no such solution exists in $O(n\Delta)$ (resp., $O(n\Delta^3)$) time, where n denotes the number of vertices and Δ denotes the maximum degree. We develop a metaheuristic algorithm that repeats the proposed local search and the plateau search iteratively, where the plateau search examines solutions of the same size as the current solution that are obtainable by exchanging a solution vertex and a non-solution vertex. The algorithm is so effective that, among 80 DIMACS graphs, it updates the best-known solution size for five graphs and performs as well as existing methods for the remaining graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Minimum independent dominating set problem, local search, plateau search, metaheuristics

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.13

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.06478`.

Supplement Material The source code of the proposed algorithm is written in C++ and available at `http://puzzle.haraguchi-s.otaru-uc.ac.jp/minids/`.

1 Introduction

Let $G = (V, E)$ be a graph such that V is the vertex set and E is the edge set. Let $n = |V|$ and $m = |E|$. A vertex subset S ($S \subseteq V$) is *independent* if no two vertices in S are adjacent, and *dominating* if every vertex in $V \setminus S$ is adjacent to at least one vertex in S. Given a graph, the *minimum independent dominating set* (*MinIDS*) problem asks for a smallest vertex subset that is dominating as well as independent. The MinIDS problem has many practical applications in data communication and networks [13].

There is much literature on the MinIDS problem in the field of discrete mathematics [8]. The problem is NP-hard [6] and also hard even to approximate; there is no constant $\varepsilon > 0$ such that the problem can be approximated within a factor of $n^{1-\varepsilon}$ in polynomial time, unless P=NP [11].

For algorithmic perspective, Liu and Song [15] and Bourgeois et al. [4] proposed exact algorithms with polynomial space. The running times of Liu and Song's algorithms are bounded by $O^*(2^{0.465n})$ and $O^*(2^{0.620n})$, and the running time of Bourgeois et al.'s algorithm is bounded by $O^*(2^{0.417n})$, where $O^*(\cdot)$ is introduced to ignore polynomial factors. Laforest

and Phan [14] proposed an exact algorithm based on clique partition, and made empirical comparison with one of the Liu and Song's algorithms, in terms of the computation time. Davidson et al. [5] proposed an integer linear optimization model for the weighted version of the MinIDS problem (i.e., weights are given to edges as well as vertices, and the weight of an edge vx is counted as cost if the edge vx is used to assign a non-solution vertex v to a solution vertex x; every non-solution vertex v is automatically assigned to an adjacent solution vertex x such that the weight of vx is the minimum) and performed experimental validation for random graphs. Recently, Wang et al. [20] proposed a tabu-search based memetic algorithm and Wang et al. [21] proposed a metaheuristic algorithm based on GRASP (greedy randomized adaptive search procedure). They showed their effectiveness on DIMACS instances, in comparison with CPLEX12.6 and LocalSolver5.5.

A vertex subset S is an IDS iff it is a maximal independent set with respect to set-inclusion [2]. Then one can readily see that the MinIDS problem is equivalent to the maximum minimal vertex cover (MMVC) problem and the minimum maximal clique problem. Zehavi [22] studied the MMVC problem, which has applications to wireless ad hoc networks, from the viewpoint of fixed-parameter-tractability.

For a combinatorially hard problem like the MinIDS problem, it is practically meaningful to develop a heuristic algorithm to obtain a nearly-optimal solution in reasonable time. In the present paper, we propose an efficient local search for the MinIDS problem. By the term "efficient", we mean that the proposed local search has a better time bound than one naïvely implemented. The local search can serve as a key tool of local improvement in a metaheuristic algorithm, or can be used in an initial solution generator for an exact algorithm. We may also expect that it is extended to the weighted version of the MinIDS problem in the future work.

Our strategy is to search for a smallest maximal independent set. Hereafter, we may call a maximal independent set simply a *solution*. In the proposed local search, we use *k-swap* for the neighborhood operation. Given a solution S, k-swap refers to the operation of obtaining another solution by dropping exactly k vertices from S and then by adding any number of vertices to it. The *k-neighborhood of S* is the set of all solutions that can be obtained by performing k-swap on S. We call S *k-minimal* if its k-neighborhood contains no S' such that $|S'| < |S|$.

To speed up the local search, one should search the neighborhood for an improved solution as efficiently as possible. For this, we propose k-neighborhood search algorithms for $k = 2$ and 3. When $k = 2$ (resp., $k = 3$ and a given solution is 2-minimal), the algorithm finds an improved solution or decides that no such solution exists in $O(n\Delta)$ (resp., $O(n\Delta^3)$) time, where Δ denotes the maximum degree in the input graph.

Furthermore, we develop a metaheuristic algorithm named *ILPS* (*Iterated Local & Plateau Search*) that repeats the proposed local search and the plateau search iteratively. ILPS is so effective that, among 80 DIMACS graphs, it updates the best-known solution size for five graphs and performs as well as existing methods for the remaining graphs.

The paper is organized as follows. Making preparations in Section 2, we present k-neighborhood search algorithms for $k = 2$ and 3 in Section 3 and describe ILPS in Section 4. We show computational results in Section 5 and then give concluding remark in Section 6.

2 Preliminaries

2.1 Notation and Terminologies

For a vertex $v \in V$, we denote by $\deg(v)$ the degree of v, and by $N(v)$ the set of neighbors of v, i.e., $N(v) = \{u \mid vu \in E\}$. For $S \subseteq V$, we define $N(S) = (\bigcup_{v \in S} N(v)) \setminus S$. We denote by $G[S]$ the subgraph induced by S. The S is called a *k-subset* if $|S| = k$.

Suppose that S is an independent set. The *tightness of $v \notin S$* is the number of neighbors of v that belong to S, i.e., $|N(v) \cap S|$. We call the v *t-tight* if its tightness is t. In particular, a 0-tight vertex is called *free*. We denote by T_t the set of t-tight vertices. Then V is partitioned into $V = S \cup T_0 \cup \cdots \cup T_{n-1}$, where T_t may be empty. Let $T_{\geq t}$ denote the set of vertices that have the tightness no less than t, that is, $T_{\geq t} = T_t \cup T_{t+1} \cup \cdots \cup T_{n-1}$.

An independent set S is a solution (i.e., a maximal independent set) iff $T_0 = \emptyset$. We call $x \in S$ a *solution vertex* and $v \notin S$ a *non-solution vertex*. When a solution vertex $x \in S$ and a t-tight vertex $v \notin S$ are adjacent, x is a *solution neighbor of v*, or equivalently, v is a *t-tight neighbor of x*.

A k-swap on a solution S is the operation of obtaining another solution $(S \setminus D) \cup A$ such that D is a k-subset of S and that A is a non-empty subset of $V \setminus S$. We call D a *dropped subset* and A an *added subset*. The k-neighborhood of S is the set of all solutions obtained by performing a k-swap on S. A solution S is k-minimal if the k-neighborhood contains no improved solution S' such that $|S'| < |S|$. Note that every solution is 1-minimal.

If a k-subset D is dropped from S, then trivially, the k solution vertices in D become free, and some non-solution vertices may also become free. Observe that a non-solution vertex becomes free if the solution neighbors are completely contained in D. We denote by $F(D)$ the set of such vertices and it is defined as $F(D) = \{v \in V \setminus S \mid N(v) \cap S \subseteq D\}$. Clearly the added subset A should be a maximal independent set in $G[D \cup F(D)]$. We have $F(D) \subseteq N(D)$, and the tightness of any vertex in $F(D)$ is at most k (at the time before dropping D from S).

2.2 Data Structure

We store the input graph by means of the typical adjacency list. We maintain a solution based on the data structure that Andrade et al. [1] invented for the maximum independent set problem. For the current solution S, we have an ordering $\pi : V \to \{1, \ldots, n\}$ on all vertices in V such that;

- $\pi(x) < \pi(v)$ whenever $x \in S$ and $v \notin S$;
- $\pi(v) < \pi(v')$ whenever $v \in T_0$ and $v' \in T_{\geq 1}$;
- $\pi(v') < \pi(v'')$ whenever $v' \in T_1$ and $v'' \in T_{\geq 2}$;
- $\pi(v'') < \pi(v''')$ whenever $v'' \in T_2$ and $v''' \in T_{\geq 3}$.

Note that the ordering is partitioned into five sections; S, T_0, T_1, T_2 and $T_{\geq 3}$. In each section, the vertices are arranged arbitrarily. We also maintain the number of vertices in each section and the tightness $\tau(v)$ for every non-solution vertex $v \notin S$.

Let us describe the time complexities of some elementary operations. We can scan each vertex section in linear time. We can pick up a free vertex (if exists) in $O(1)$ time. We can drop (resp., add) a vertex v from (resp., to) the solution in $O(\deg(v))$ time. See [1] for details.

Before closing this preparatory section, we mention the time complexities of two essential operations.

▶ **Proposition 1.** *Let D be a k-subset of S. We can list all vertices in $F(D)$ in $O(k\Delta)$ time.*

Proof. We let every $v \in V$ have an integral counter, which we denote by $c(v)$. It suffices to scan vertices in $N(D)$ twice. In the first scan, we initialize the counter value as $c(u) \leftarrow 0$ for every neighbor $u \in N(x)$ of every solution vertex $x \in D$. In the second, we increase the counter of u by one (i.e., $c(u) \leftarrow c(u) + 1$) when u is searched in the adjacency list of $x \in D$. Then, if $c(u) = \tau(u)$ holds, we output u as a member of $F(D)$ since the equality represents that every solution neighbor of u is contained in D. Obviously the time bound is $O(k\Delta)$. ◀

▶ **Proposition 2.** *Let D be a k-subset of S. For any non-solution vertex $v \in F(D)$, we can decide whether v is adjacent to all vertices in $F(D) \setminus \{v\}$ in $O(k\Delta)$ time.*

Proof. We use the algorithm of Proposition 1. As preprocessing of the algorithm, we set the counter $c(u)$ of each $u \in N(v)$ to 0, i.e., $c(u) \leftarrow 0$, which can be done in $O(\deg(v))$ time. After we acquire $F(D)$ by running the algorithm of Proposition 1, we can see if v is adjacent to all other vertices in $F(D)$ in $O(\deg(v))$ time by counting the number of vertices $u \in N(v)$ such that $\tau(u) \in \{1, \ldots, k\}$ and $c(u) = \tau(u)$. If the number equals to (resp., does not equal to) $|F(D)| - 1$, then we can conclude that it is true (resp., false). ◀

3 Local Search

Assume that, for some $k \geq 2$, a given solution S is k'-minimal for every $k' \in \{1, \ldots, k-1\}$. Such k always exists, e.g., $k = 2$. In this section, we consider how we find an improved solution in the k-neighborhood of S or conclude that S is k-minimal efficiently.

Let us describe how time-consuming naïve implementation is. In naïve implementation, we search all k-subsets of S as candidates of the dropped subset D, where the number of them is $O(n^k)$. Furthermore, for each D, there are $O(n^{k-1})$ candidates of the added subset A. The number of possible pairs (D, A) is up to $O(n^{2k-1})$.

In the proposed neighborhood search algorithm, we do not search dropped subsets but added subsets; we generate a dropped subset from each added subset. When $k \in \{2, 3\}$, the added subsets can be searched more efficiently than the dropped subsets. This search strategy stems from Proposition 3, a necessary condition of a k-subset D that the improvement is possible by a k-swap that drops D. We introduce the condition in Section 3.1.

Then in Section 3.2 (resp., 3.3), we present a k-neighborhood search algorithm that finds an improved solution or decides that no such solution exists for $k = 2$ (resp., 3), which runs in $O(n\Delta)$ (resp., $O(n\Delta^3)$) time.

3.1 A Necessary Condition for Improvement

Let D be a k-subset of S. If there is a subset $A \subseteq F(D)$ such that A is maximal independent in $G[D \cup F(D)]$ and $|A| < |D|$, then we have an improved solution $(S \setminus D) \cup A$. The connectivity of $G[D \cup F(D)]$ is necessary for the existence of such A, as stated in the following proposition.

▶ **Proposition 3.** *Suppose that a solution S is k'-minimal for every $k' \in \{1, \ldots, k-1\}$ for some integer $k \geq 2$. Let D be a k-subset of S. There is a maximal independent set A in $G[D \cup F(D)]$ such that $A \subseteq F(D)$ and $|A| < |D|$ only when the subgraph is connected.*

Proof. Suppose that $G[D \cup F(D)]$ is not connected. Let q be the number of connected components and $D^{(p)} \cup F^{(p)}(D)$ be the subset of vertices in the p-th component ($q \geq 2$, $p = 1, \ldots, q$, $D^{(p)} \subseteq D$, $F^{(p)}(D) \subseteq F(D)$). Each $D^{(p)}$ is not empty since otherwise there would be an isolated vertex in $F^{(p)}(D)$. It is a free vertex with respect to S, which contradicts that S is a solution. Then we have $1 \leq |D^{(p)}| < k$.

The maximal independent set A is a subset of $F(D)$. We partition A into $A = A^{(1)} \cup \cdots \cup A^{(q)}$, where $A^{(p)} = A \cap F^{(p)}(D)$. Each $A^{(p)}$ is maximal independent for the p-th component. As $|A| < |D|$, $|A^{(p)}| < |D^{(p)}|$ holds for some p. Then we can construct an improved solution $(S \setminus D^{(p)}) \cup A^{(p)}$, which contradicts the k'-minimality of S. ◀

3.2 2-Neighborhood Search

Applying Proposition 3 to the case of $k = 2$, we have the following proposition.

▶ **Proposition 4.** *Let D be a 2-subset of S. There is a non-solution vertex v in $F(D)$ such that $(S \setminus D) \cup \{v\}$ is a solution only when there is a 2-tight vertex in $F(D)$.*

We can say more on Proposition 4. The vertex v should be 2-tight since, if not so (i.e., v is 1-tight), $\{v\}$ would not be maximal independent for $G[D \cup F(D)]$; v is adjacent to only one of $D = \{x, y\}$ from the definition of 1-tightness.

In summary, if there is an improved solution $(S \setminus D) \cup \{v\}$, then v is 2-tight and has x and y as the solution neighbors. Instead of searching all 2-subsets of S, we scan all 2-tight vertices, and for each 2-tight vertex v, we take $D = \{x, y\}$ as the candidate of the dropped set. We have the following theorem.

▶ **Theorem 5.** *Given a solution S, we can find an improved solution in the 2-neighborhood or conclude that S is 2-maximal in $O(n\Delta)$ time.*

Proof. Since we maintain the solution by means of the vertex ordering, we can scan all the 2-tight vertices in $O(|T_2|)$ time. For each 2-tight v, we can detect the two solution neighbors, say x and y, in $O(\deg(v))$ time.

Let $D = \{x, y\}$. The singleton $\{v\}$ is maximal independent for $G[D \cup F(D)]$ and thus we have an improved solution $(S \setminus D) \cup \{v\}$ iff v is adjacent to all other vertices in $F(D)$. Whether v is adjacent to all other vertices in $F(D)$ is decided in $O(\Delta)$ time, as we stated in Proposition 2. If it is the case, then we can construct an improved solution $(S \setminus D) \cup \{v\}$ in $O(\deg(x) + \deg(y) + \deg(v)) = O(\Delta)$ time as the vertex ordering takes $O(\deg(x))$ time to drop x from S and $O(\deg(v))$ time to add v to it [1]. Otherwise, we can conclude that $(S \setminus D) \cup \{v\}$ is not a solution because some vertices in $F(D)$ are not dominated.

We have seen that, for each 2-tight vertex v, it takes $O(\Delta)$ time to find an improved solution $(S \setminus D) \cup \{v\}$ or to conclude that it is not a solution. Therefore, the overall running time is bounded by $O(|T_2|\Delta) = O(n\Delta)$. ◀

3.3 3-Neighborhood Search

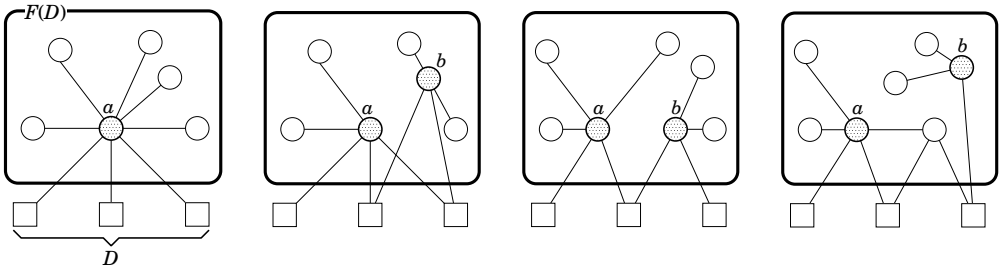
We have the following proposition by applying Proposition 3 to the case of $k = 3$.

▶ **Proposition 6.** *Suppose that S is a 2-minimal solution and that $D = \{x, y, z\}$ is a 3-subset of S. There is a subset A of $F(D)$ such that A is maximal independent in $G[D \cup F(D)]$ and $|A| < |D|$ only when either of the followings holds:*

(a) *there is a 3-tight vertex in $F(D)$ that has x, y and z as the solution neighbors;*

(b) *there are two 2-tight vertices in $F(D)$ such that one has x and y as the solution neighbors and the other has x and z as the solution neighbors.*

Let us make observation on the added subset. Suppose that, for an arbitrary 3-subset $D \subseteq S$, there is $A \subseteq F(D)$ such that A is maximal independent in $G[D \cup F(D)]$ and $|A| < |D|$. When $|A| = 1$, the only vertex in A is 3-tight since otherwise some vertex in D would not be dominated. When $|A| = 2$, at least one of the two vertices in A is either 2-tight or 3-tight; if



(i) $A = \{a\}$ **(ii)** $A = \{a, b\}$ **(iii)** $A = \{a, b\}$ **(iv)** $A = \{a, b\}$

Figure 1 Illustration of a dropped set D and an added set A for (i) to (iv) in Section 3.3: For clarity of the figure, we draw only edges that are incident to the vertices a and b. Note that every vertex in $F(D)$ is adjacent to at least one vertex in D.

both are 1-tight, one vertex of D would not be dominated. Concerning the tightness, the following four situations are possible:

- **(i)** $A = \{a\}$ and a is 3-tight;
- **(ii)** $A = \{a, b\}$, a is 3-tight, and b is t-tight such that $t \in \{1, 2, 3\}$;
- **(iii)** $A = \{a, b\}$, a is 2-tight, and b is 2-tight;
- **(iv)** $A = \{a, b\}$, a is 2-tight, and b is 1-tight.

From (ii) to (iv), the vertices a and b are not adjacent. We illustrate (i) to (iv) in Figure 1.

Based on the above, we summarize the search strategy as follows. In order to generate all 3-subsets D of S such that $F(D)$ satisfies either (a) or (b) of Proposition 6, we scan all 3-tight vertices u (Proposition 6 (a)) and all pairs of 2-tight vertices, say v and w, such that $|(N(v) \cup N(w)) \cap S| = 3$ (Proposition 6 (b)). For (a), we take $D = N(u) \cap S$ and search $F(D)$ for a 1- or 2-subset A that is maximal independent in $G[D \cup F(D)]$, regarding the 3-tight vertex u as the vertex a in (i) and (ii). Similarly, for (b), we take $D = (N(v) \cup N(w)) \cap S$ and search $F(D)$ for a 2-subset A that is maximal independent in $G[D \cup F(D)]$, regarding the 2-tight vertex v as the vertex a in (iii) and (iv).

We have the following theorem on the time complexity of 3-neighborhood search. We omit the proof due to space limitation.

▶ **Theorem 7.** *Given a 2-minimal solution S, we can find an improved solution in the 3-neighborhood or conclude that S is 3-maximal in $O(n\Delta^3)$ time.*

4 Iterated Local & Plateau Search

In this section, we present a metaheuristic algorithm named ILPS (Iterated Local & Plateau Search) that repeats the proposed local search and the plateau search iteratively.

We show the pseudo code of ILPS in Algorithm 1. The ILPS has four parameters, that is S, k, δ and ν, where S is an initial solution, k is the order of the local search (i.e., a k-minimal solution is searched by LocalSearch(S, k) in Line 6), and δ and ν are integers. The roles of the last two parameters are mentioned in Section 4.2.

The LocalSearch(S, k) in Line 6 is the subroutine that returns a k-minimal solution from an initial solution S, where k is set to either two or three. When $k = 2$, it determines a 2-minimal solution by moving to an improved solution repeatedly as long as the 2-neighborhood search algorithm delivers one. When $k = 3$, it first finds a 2-minimal solution, and then runs the 3-neighborhood search algorithm. If an improved solution is delivered, then the local search moves to the improved solution and seeks a 2-minimal one again since the solution is not necessarily 2-minimal. Otherwise, the current solution is 3-minimal.

Algorithm 1 Iterated Local & Plateau Search (ILPS).

```
1: function ILPS(S, k, δ, ν)
2:     S* ← S                                   ▷ S* is used to store the incumbent solution
3:     ρ ← a penalty function such that ρ(v) = 0 for all v ∈ V
4:     ρ ← UpdatePenalty(S, ρ, δ)
5:     while termination condition is not satisfied do
6:         S ← LocalSearch(S, k)                ▷ The local search returns a k-minimal solution
7:         S ← PlateauSearch(S, k)              ▷ The plateau search returns a k-minimal solution
8:         if |S| ≤ |S*| then
9:             S* ← S
10:        end if
11:        S ← Kick(S*, ρ, ν)                   ▷ The initial solution of the next iteration is generated
12:        ρ ← UpdatePenalty(S, ρ, δ)           ▷ The penalty function is updated
13:    end while
14:    return S*
15: end function
```

Below we explain two key ingredients: the plateau search and the vertex penalty. We describe these in Sections 4.1 and 4.2 respectively. We remark that they are inspired by *Dynamic Local Search* for the maximum clique problem [19] and *Phased Local Search* for the unweighted/weighted maximum independent set and minimum vertex cover [18].

4.1 Plateau Search

In the plateau search (referred to as PlateauSearch(S, k) in Line 7), we search solutions of the size $|S|$ that can be obtained by swapping a solution vertex $x \in S$ and a non-solution vertex $v \notin S$. Let $\mathcal{P}(S)$ be the collection of all solutions that are obtainable in this way. The size of any solution in $\mathcal{P}(S)$ is $|S|$. We execute LocalSearch(S', k) for every solution $S' \in \mathcal{P}(S)$, and if we find an improved solution S'' such that $|S''| < |S'| = |S|$, then we do the same for S'', i.e., we execute LocalSearch(P, k) for every solution $P \in \mathcal{P}(S'')$. We repeat this until no improved solution is found and employ a best solution among those searched as the output of the plateau search.

We emphasize the efficiency of the plateau search; all solutions in $\mathcal{P}(S)$ can be listed in $O(|T_1|\Delta)$ time. Observe that $(S \setminus \{x\}) \cup \{v\}$ is a solution iff v is 1-tight such that x is the only solution neighbor of v, and v is adjacent to all vertices in $F(\{x\})$ other than v. We can scan all 1-tight vertices in $O(|T_1|)$ time. For each 1-tight vertex v, the solution neighbor x is detected in $O(\deg(v))$ time, and whether the last condition is satisfied or not is identified in $O(\Delta)$ time from Proposition 2. Dropping x from S and adding v to $S \setminus \{x\}$ can be done in $O(\Delta)$ time.

4.2 Vertex Penalty

In order to avoid the search stagnation, one possible approach is to apply a variety of initial solutions. To realize this, we introduce a penalty function $\rho : V \to \mathbb{Z}^+ \cup \{0\}$ on the vertices. The penalty function ρ is initialized so that $\rho(v) = 0$ for all $v \in V$ (Line 3). During the algorithm, ρ is managed by the subroutine UpdatePenalty (Lines 4 and 12). When the initial solution S of the next local search is determined, it increases the penalty $\rho(v)$ of every vertex $v \in S$ by one, i.e., $\rho(v) \leftarrow \rho(v) + 1$. Furthermore, to "forget" the search history long ago, it reduces $\rho(v)$ to $\lfloor \min\{\rho(v), \delta\}/2 \rfloor$ for all $v \in V$ in every δ iterations. This δ is the third parameter of ILPS and called the *penalty delay*.

The ρ is used in the subroutine KICK (Line 11), the initial solution generator, so that vertices with fewer penalties are more likely to be included in the initial solution. KICK generates an initial solution by adding non-solution vertices (with respect to the incumbent solution S^*) "forcibly" to S^*. The added vertices are chosen one by one as follows; in one trial, KICK picks up one non-solution vertex. It then goes on to the next trial with the probability $(\nu - 1)/\nu$ or stops the selection with the probability $1/\nu$, where ν is the fourth parameter of ILPS. Observe that ν specifies the expected number of added vertices. In the first trial, KICK randomly picks up a non-solution vertex that has the fewest penalty. In a subsequent r-th trial ($r = 2, 3, \dots$), let $R = \{v_1, \dots, v_{r-1}\}$ be the set of vertices chosen so far. KICK samples three vertices randomly from $V \setminus (S^* \cup R \cup N(R))$, and picks up the one that has the fewest penalty among the three. Suppose that $R = \{v_1, \dots, v_r\}$ has been picked up as the result of r trials. Then we construct an independent set $S = (S^* \setminus N(R)) \cup R$. The S may not be a solution as there may remain free vertices. If so, we repeatedly pick up free vertices by the maximum-degree greedy method until S becomes a solution. We use the acquired S as the initial solution of the next local search.

5 Computational Results

We report some experimental results in this section. In Section 5.1, to gain insights into what kind of instance is difficult, we examine the phase transition of difficulty with respect to the edge density. The next two subsections are devoted to observation on the behavior of the proposed method. In Section 5.2, we show how a single run of LOCALSEARCH(S, k) improves a given initial solution. In Section 5.3, we show how the penalty delay δ affects the search. Finally in Section 5.4, we compare ILPS with the memetic algorithm [20], GRASP+PC [21], CPLEX12.6 [12] and LocalSolver5.5 [16] in terms of the solution size, using DIMACS graphs.

All the experiments are conducted on a workstation that carries an Intel Core i7-4770 Processor (up to 3.90GHz by means of Turbo Boost Technology) and 8GB main memory. The installed OS is Ubuntu 16.04. Under this environment, it takes 0.25 s, 1.54 s and 5.90 s approximately to execute `dmclique` (`http://dimacs.rutgers.edu/pub/dsj/clique/`) for instances `r300.5.b`, `r400.5.b` and `r500.5.b`, respectively. The ILPS algorithm is implemented in C++ and compiled by the g++ compiler (ver. 5.4.0) with `-O2` option.

5.1 Phase Transition of Difficulty

The phase transition has been observed for many combinatorial problems [7, 9, 10]. Roughly, it is said that over-constrained and under-constrained instances are relatively easy, and that intermediately constrained ones tend to be more difficult.

In the MinIDS problem, the amount of constraints is proportional to the edge density p. We examine the change of difficulty with respect to p. We estimate the difficulty of an instance by how long CPLEX12.8 takes to solve it.

For each $(n, p) \in \{100, 150, 200\} \times \{0.00, 0.05, \dots, 1.00\}$, we generate 100 random graphs (Erdös-Rényi model) with n vertices and the edge density p, i.e., an edge is drawn between two vertices with probability p. We solve the 100 instances by CPLEX12.8 and take the averaged computation time. We set the time limit of each run to 60 s. If CPLEX12.8 terminates by the time limit, then we regard the computation time as 60 s.

Figure 2 shows the result. We may say that instances with the edge densities from 0.1 to 0.4 are likely to be more difficult than others. In fact, the experiments in [5, 14] mainly deal with random graphs with the edge densities in this range.

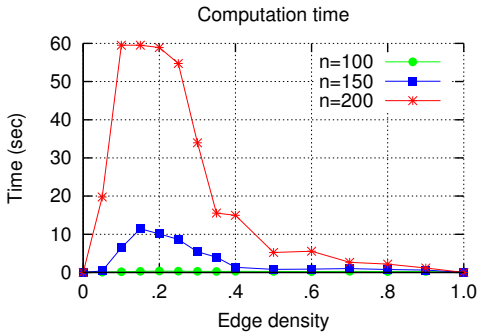


Figure 2 Computation time of CPLEX12.8 for random graphs.

Table 1 Averaged sizes of random, 2-minimal and 3-minimal solutions in random graphs with 10^3 vertices.

	$p = .1$	.2	.3	.4	.5	.6	.7	.8	.9	.95	.99
random	44.57	24.42	16.70	12.50	9.66	7.70	6.12	4.84	3.62	3.00	2.12
2-minimal	37.37	20.36	13.84	10.18	7.86	6.12	4.95	3.95	2.99	2.00	1.95
3-minimal	35.44	19.04	12.74	9.28	7.01	5.64	4.06	3.02	2.15	2.00	1.95

5.2 A Single Run of Local Search

We show how a single run of LocalSearch(S, k) improves an initial solution S. Again we take a random graph. We fix the number n of vertices to 10^3. For every $p \in \{0.1, \ldots, 0.9, 0.95, 0.99\}$, we generate 100 random graphs. Then for each graph, we run LocalSearch(S, k) five times, where we use different random seeds in each time and construct the initial solution S randomly.

We show the averaged sizes of random, 2-minimal and 3-minimal solutions in Table 1. We see that, the larger the edge density p is, the fewer the solution size becomes. The local search improves a random solution to some extent. LocalSearch$(S, 3)$ improves the solution more than LocalSearch$(S, 2)$. The difference between the two local searches is the largest when $p = 0.1$, that is $37.37 - 35.44 = 1.93$. The difference gets smaller when p gets larger. In particular, when $p > 0.9$, we see no difference.

Let us discuss computation time. In the left of Figure 3, we show how the averaged computation time changes with respect to p. We see that the computation time of LocalSearch$(S, 3)$ is tens to thousands of times the computation time of LocalSearch$(S, 2)$. However, it does not necessarily diminish the value of the 3-neighborhood search. As will be shown in Section 5.4, when $k = 3$, ILPS can find such a good solution that is not obtained by $k = 2$.

In general, for a fixed k, it takes more computation time when p is larger. Recall Theorem 5 (resp., 7); when $k = 2$ (resp., 3), the k-neighborhood search algorithm finds an improved solution for the current solution S or concludes that S is k-minimal in $O(n\Delta)$ (resp., $O(n\Delta^3)$) time. Roughly, Δ is increasing as p gets larger.

For $k = 3$, we attribute the peak at $p = 0.8$ to the number of 3-tight vertices. In the right of Figure 3, We show the averaged numbers of 2- and 3-tight vertices with respect to 3-minimal solutions. The 3-neighborhood search algorithm searches 2- and 3-tight vertices. The numbers of both vertices are generally non-decreasing from $p = 0.1$ to 0.8, but when $p > 0.8$, the number of 3-tight vertices decreases dramatically. This is due to the solution

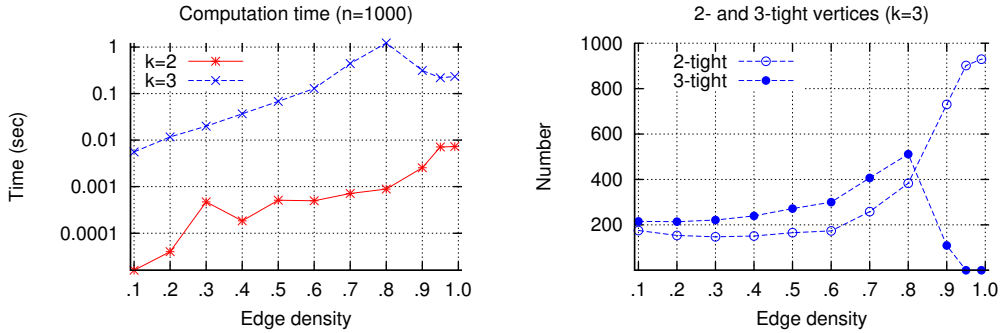


Figure 3 (Left) averaged computation time that LocalSearch(S, k) takes to decide a k-minimal solution (Right) numbers of 2- and 3-tight vertices with respect to 3-minimal solutions.

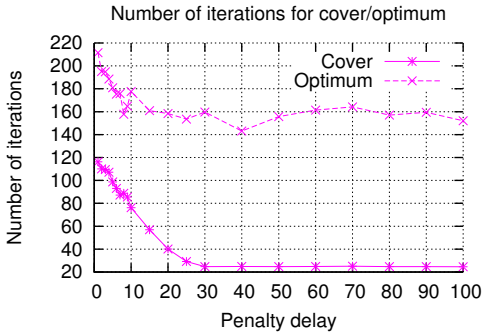


Figure 4 Averaged numbers of iterations to cover all vertices (solid line) and to find the optimum (dashed line).

size. The solution size gives an upper bound on the tightness of any non-solution vertex, and when $p > 0.8$, the averaged size of a 3-minimal solution is less than three; see Table 1. Since most of the non-solution vertices are either 1- or 2-tight, we hardly handle the situations (i) and (ii) in Section 3.3.

5.3 Penalty Delay

We introduced the notion of vertex penalty to control the search diversification. When the penalty delay δ is larger, more varieties of initial solutions are expected to be tested in ILPS.

To illustrate the expectation, we evaluate how many iterations ILPS takes until all vertices are covered by the initial solutions, that is, used in the initial solutions at least once. The solid line in Figure 4 shows the number of iterations taken to cover all vertices. The graph we employ here is a 10×10 grid graph such that each vertex is associated with a 2D integral point $(i, j) \in \{1, \ldots, 10\}^2$, and that two vertices (i, j) and (i', j') are adjacent iff $|i - i'| + |j - j'| = 1$. For each δ, the number of iterations is averaged over 500 runs of ILPS with different random seeds, where we fix $(k, \nu) = (2, 1)$ and construct the first initial solution S by the maximum-degree greedy algorithm.

The observed phenomenon meets our expectation; The number is non-increasing with respect to δ and saturated for $\delta \geq 30$. In other words, when δ is larger, more varieties of initial solutions are generated in a given number of iterations.

However, setting δ to a large value does not necessarily lead to discovery of better solutions.

The dashed line in Figure 4 shows the averaged number of iterations that ILPS takes to find an optimal solution; we know that the optimal size is 24 since we solve the instance optimally by CPLEX. When $\delta \leq 40$, the number is approximately decreasing and takes the minimum at $\delta = 40$, but a larger δ does not make any improvement. Hence, given an instance, we need to choose an appropriate value of δ carefully.

5.4 Performance Validation

We run ILPS algorithm for 80 DIMACS instances that are downloadable from [17]. We generate the first initial solution S by the maximum-degree greedy method, and fix the parameter ν to three. For (k, δ), all pairs in $\{2, 3\} \times \{2^0, \ldots, 2^6\}$ are tested. For each instance and each (k, δ), we run ILPS algorithm 10 times, using different random seeds. We terminate the algorithm by the time limit. The time limit is set to 200 s. When $k = 3$, we modify Algorithm 1 so that PlateauSearch(S, k) in Line 7 is called only when $|S| \leq |S^*| + 2$ as the plateau search is rather time-consuming.

We take four competitors from [20] and [21]. The first is MEM, a tabu-search based memetic algorithm in [20]. The second is GP, the GRASP+PC algorithm in [21]. The third is CP, which stands for CPLEX12.6 [12] that solves an integer optimization model of the MinIDS problem. The fourth is LS, which stands for LocalSolver5.5 [16], a general discrete optimization solver based on local search. MEM is run on a computer with a 2.0GHz CPU and a 4GB memory, whereas the other competitors are run on computers with a 2.3GHz CPU and an 8GB memory. The time limit of MEM and GP is set to 200 s, and that of CP and LS is set to 3600 s.

In Table 2, we show the results on selected instances. The columns "n" and "p" indicate the number of vertices and the edge density, respectively. The edge density is between 0.1 and 0.5 in all instances except hamming8-2. In our context, the instances are expected to be difficult. For ILPS, we show the results for $(k, \delta) = (2, 2^6)$ in detail, regarding this pair as the representative. The columns "Min" and "Max" indicate the minimum/maximum solution size over 10 runs, and the column "Avg" indicates the average. The column "TTB" indicates the time to best (in seconds), that is, the average of the computation time that ILPS takes to find the solution of the size "Min". The symbol ε represents that the time is less than 0.1 s. The column "Best" indicates the minimum solution size attained over all $(k, \delta) \in \{2, 3\} \times \{2^0, \ldots, 2^6\}$. The rightmost four columns indicate the solution size attained by the competitors. The symbol $*$ before the instance name indicates that the solution size attained by CPLEX is optimal.

The table contains only results on the 13 selected instances such that the solutions sizes attained by "Best", "MEM" and "GP" are not-all-equal, except hamming8-2. We guarantee that, for the remaining 67 instances, ILPS's "Best" is as good as any competitor. The boldface indicates that the solution size is strictly smaller than those of the competitors. Then we update the best-known solution size in five graphs. These show the effectiveness of the proposed local search and the ILPS algorithm.

For hamming8-2, when $k = 2$, ILPS cannot find a solution of the optimal size 32 for any penalty delay $\delta \in \{2^0, \ldots, 2^6\}$. However, when $k = 3$, ILPS finds an optimal solution with $\delta = 2^0$, 2^1 and 2^2.

Before closing this section, let us report our preliminary results briefly.

- A preliminary version of ILPS happened to find a solution of the size 31 for C2000.9 and a solution of the size 15 for keller6.
- Let us consider a finer swap operation, (j, k)-*swap*, that obtains another solution by dropping exactly k vertices from the current one and then by adding exactly j vertices

Table 2 Selected results from the validation experiments on DIMACS graphs.

	n	p	ILPS ($k=2$, $\nu=2^6$)					[20]	[21]		
			Min	Avg	Max	TTB	Best	MEM	GP	CP	LS
brock400_2	400	.25	10	10.0	10	1.1	9	9	10	10	11
C1000.9	1000	.10	27	27.8	29	0.0	**26**	27	27	29	30
∗C125.9	125	.10	14	14.0	14	0.1	14	14	15	14	14
C2000.9	2000	.10	**32**	33.6	35	12.1	**32**	33	33	48	36
C4000.5	4000	.50	**7**	7.9	8	49.7	**7**	8	8	-	-
C500.9	500	.10	22	22.2	23	92.3	**21**	22	23	23	22
gen400_p0.9_55	400	.10	20	20.1	21	39.4	20	20	21	22	22
gen400_p0.9_65	400	.10	20	20.7	21	99.0	20	20	21	21	22
∗hamming8-2	256	.03	36	36.0	36	0.0	32	N/A	32	32	32
keller6	3361	.18	18	18.0	18	26.1	**16**	18	18	32	19
∗san200_0.7_1	200	.30	6	6.1	7	85.9	6	6	7	6	7
∗san200_0.9_1	200	.10	15	15.0	15	16.7	15	15	16	15	16
san400_0.7_3	400	.30	7	7.8	8	106.6	7	7	8	8	9

to it. One can prove that, given a solution S and a constant k, we can improve S by $(1,k)$-swap or conclude that it is not possible in $O(n\Delta)$ time. We implemented $(1,k)$-swap in a preliminary version of ILPS, but it does not yield significant improvement even when k is set to a constant larger than three.

- We tested Laforest and Phan's exact algorithm [14], and found that the algorithm is not suitable for a task of finding a good solution quickly. The source code is available at `http://todo.lamsade.dauphine.fr/spip.php?article42`.
- BHOSLIB [3] is another well-known collection of benchmark instances. It contains 36 instances such that n is between 450 and 4000 and that p is no less than 0.82. Hence, the BHOSLIB instances are expected to be easy in our context. The ILPS with $(k,\delta)=(2,2^6)$ finds a solution of the size three for all the instances. We also run CPLEX12.8 for 200 s, generating an initial solution by the maximum-degree greedy algorithm. CPLEX12.8 finds a solution of the size five for frb100-40, and a solution of the size three for the other instances. In addition, the solution of the size three is proved to be optimal for 15 instances whose names start with frb30, frb35 and frb40.

6 Concluding Remark

We have considered an efficient local search for the MinIDS problem. We proposed fast k-neighborhood search algorithms for $k=2$ and 3, and developed a metaheuristic algorithm named ILPS that repeats the local search and the plateau search iteratively. ILPS is so effective that it updates the best-known solution size in five DIMACS graphs.

The proposed local search is applicable to other metaheuristics such as genetic algorithms, as a key tool of local improvement. The future work includes an extension of the local search to a weighted version of the MinIDS problem.

References

1 D.V. Andrade, M.G.C. Resende, and R.F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18:525–547, 2012.

2 C. Berge. *Theory of Graphs and its Applications*. Methuen, London, 1962.

3 BHOSLIB: Benchmarks with hidden optimum solutions for graph problems. `http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`. accessed on February 1, 2018.

4 N. Bourgeois, F.D. Croce, B. Escoffier, and V.Th. Paschos. Fast algorithms for MIN independent dominating set. *Discrete Applied Mathematics*, 161(4):558–572, 2013.

5 P.P. Davidson, C. Blum, and J. Lozano. The weighted independent domination problem: ILP model and algorithmic approaches. In *Proc. EvoCOP 2017*, pages 201–214, 2017. `doi:10.1007/978-3-319-55453-2_14`.

6 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, 1979.

7 I.P. Gent and T. Walsh. The SAT phase transition. In *Proc. ECAI-94*, pages 105–109, 1994.

8 W. Goddard and M.A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313:839–854, 2013.

9 C.P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proc. AAAI-97*, pages 221–227, 1997.

10 C.P. Gomes and D.B. Shmoys. Completing quasigroups or latin squares: a structured graph coloring problem. In *Proc. Computational Symposium on Graph Coloring and Generalizations*, 2002.

11 M.M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993.

12 IBM ILOG CPLEX. `https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer`. accessed on February 1, 2018.

13 F. Kuhn, T. Nieberg, T. Moscibroda, and R. Wattenhofer. Local approximation schemes for ad hoc and sensor networks. In *Proc. the 2005 Joint Workshop on Foundations of Mobile Computing*, pages 97–103, 2005.

14 C. Laforest and R. Phan. Solving the minimum independent domination set problem in graphs by exact algorithm and greedy heuristic. *RAIRO-Operations Research*, 47(3):199–221, 2013.

15 C. Liu and Y. Song. Exact algorithms for finding the minimum independent dominating set in graphs. In *Proc. ISAAC 2006*, LNCS 4288, pages 439–448, 2006.

16 LocalSolver. `http://www.localsolver.com/`. accessed on February 1, 2018.

17 F. Mascia. dimacs benchmark set. `http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark`. accessed on February 1, 2018.

18 W. Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization*, 6(2):214–219, 2009.

19 W. Pullan and H.H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

20 Y. Wang, J. Chen, H. Sun, and M. Yin. A memetic algorithm for minimum independent dominating set problem. *Neural Computing and Applications*, in press. `doi:10.1007/s00521-016-2813-7`.

21 Y. Wang, R. Li, Y. Zhou, and M. Yin. A path cost-based grasp for minimum independent dominating set problem. *Neural Computing and Applications*, 28(1):143–151, 2017. `doi:10.1007/s00521-016-2324-6`.

22 M. Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017.