Learning driven three-phase search for the maximum independent union of cliques problem

Zhi Lu[a], Jian Gao[b], Jin-Kao Hao[c,*], Pingle Yang[a], Lixin Zhou[a]

[a] *Business School, University of Shanghai for Science & Technology, Shanghai, China*

[b] *School of CS & IT, Northeast Normal University, Changchun, China*

[c] *LERIA, Université d'Angers, Angers, France*

Computers & Operations Research, Jan. 2024,
https://doi.org/10.1016/j.cor.2024.106549

Abstract

Given a simple and undirected graph, the maximum independent union of cliques (IUC) problem aims to identify a subset of vertices with maximum cardinality, such that each connected component of the induced subgraph is a complete graph. As a generalization of the popular NP-hard maximum clique problem, the maximum IUC problem is of great practical importance for social network analysis and network-based data mining. In this work, we present the first learning driven three-phase search algorithm for this relevant problem. The proposed algorithm incorporates a constrained swap-based tabu search to effectively examine candidate solutions and a frequency-based perturbation to diversify the search. It additionally integrates a probability learning mechanism to learn useful information during the search, which helps to build promising new starting solutions. Computational results on 83 benchmark graphs from the well-known 2nd DIMACS Challenge indicate that the algorithm competes very favorably with the current best-performing algorithms. We also present a practical application of the algorithm to social network analysis. Key algorithmic components are analyzed to understand their influences on the algorithm.

Keywords: Combinatorial optimization, heuristics, metaheuristics, reinforcement learning, social network analysis.

[*]Corresponding author.
Email addresses: `zhilusix@gmail.com` (Zhi Lu), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao)

1. Introduction

A clique of an undirected graph is a subset of vertices that induces a complete subgraph, i.e., the vertices are pairwise adjacent. The NP-hard maximum clique problem is one of the most popular and general clique models, which involves finding in a graph the clique with maximum cardinality (i.e., the clique number) [35]. The maximum clique problem has many practical applications in areas like telecommunications [1], bioinformatics [26], economics [5], social network analysis [27], etc. However, the rigorous pairwise connectivity requirement of a clique becomes too rigid to formulate some real-world problems. As a result, a number of generalized or relaxed clique models have been proposed in the literature.

In this work, we are interested in the following relaxed clique model. Given a simple and undirected graph $G = (V, E)$ with a vertex set $V = \{1, \ldots, n\}$ and an edge set $E \subset V \times V$. An independent union of cliques (IUC) is a subset of vertices $S \subseteq V$ such that each connected component of the induced subgraph is a complete graph. The maximum IUC problem is then to find the maximum IUC with the largest cardinality (called the IUC number and denoted by $\omega^{\alpha}(G)$) [6].

Let $S \subseteq V$ be an IUC of G, we define a binary variable x_i where $x_i = 1$ if vertex i belongs to S, and $x_i = 0$ otherwise. Let $A[n, n]$ be the adjacency matrix of G such that $a_{ij} = 1$ if $\{i, j\} \in E$, and $a_{ij} = 0$ otherwise. Then, the maximum IUC problem can be formally stated by the following linear integer programming model [6],

$$\alpha^{\omega}(G) = \max \sum_{i \in V} x_i \tag{1}$$

$$\textbf{s.t.} \quad x_i + x_j + x_l \leq 2 \quad \forall i,\ j,\ l \in V \ \textit{such that} \ a_{ij} + a_{jl} + a_{il} = 2 \tag{2}$$

$$x_i \in \{0, 1\},\ i \in V. \tag{3}$$

where constraint (2) ensures that a subset of vertices $S \subseteq V$ is an IUC if and only if it contains no set of three vertices inducing an open triangle (an open triangle is a graph with three vertices and two edges). Constraint (3) indicates the binary nature of the variables x_i. This model is based on the property that $S \subseteq V$ is an IUC if and only if no set of three vertices from S induces an open triangle in G [6].

Similar to the maximum clique problem, the maximum IUC problem arises naturally in several real-world applications. For example, it is a useful tool for analyzing cohesive subgraphs and community structures in network-based data mining [6, 15]. In fact, solving the maximum IUC problem results in a structure with multiple connected components, each of which is a clique. These cliques, also known as cohesive subgraphs, help us understand social communication within the networks. In addition, these cliques can also serve as seeds for network clustering algorithms that partition the network nodes into different clusters. We will discuss one such application of this work to social network analysis in Section 3.5.

The maximum IUC problem is strongly related to the maximum α-cluster problem [7, 33], which is a relevant model for community detection in social network analysis. The maximum α-cluster problem is to find the maximum number of vertices inducing a subgraph in which each vertex has a local clustering coefficient of at least α. The maximum IUC problem is equivalent to the maximum 1-cluster problem when $\alpha = 1$ and the connectivity constraint of the α-cluster is relaxed to allow multiple connected components. In [7], several mathematical models (cubic and quadratic models, and triangle model) were proposed, which were solved by using the FICO Xpress-IVE solver to compute the maximum α-clusters of social networks with up to 115 vertices. A simple network clustering algorithm was also presented to compute disjoint unions of α-clusters. Based on their computational results, the authors of [7] recognized that heuristic approaches need to be developed to deal with large social networks. The maximum IUC problem is also closely related to several other clustering problems, such as the cluster vertex deletion problem [16], the s-plex cluster vertex deletion problem (when $s = 1$) [30], and the maximum induced cluster subgraph problem (also known as the maximum induced P_3-free subgraph problem) [8].

The maximum IUC problem is NP-hard in general, and this remains true even for some special cases such as planar graphs, claw-free graphs, and bipartite graphs [6]. A number of theoretical studies have been performed to uncover the properties of the problem. Recent theoretical analyses mainly concern the upper (or lower) bounds and time complexity of the IUC and related problems. Given that the IUC problem is a generalization of clique, cliques are feasible solutions for the maximum IUC problem and the clique number provides a lower bound of the maximum IUC [29, 6]. Besides, parametrized algorithms with guaranteed run-time bounds were studied for the related cluster vertex deletion problem [16] and the related maximum induced cluster subgraph problem [8].

Although interesting progresses have been made in theoretical studies on the maximum IUC problem, research on practical IUC algorithms is still in its early stages. Unlike the maximum clique problem for which many exact algorithms and heuristic algorithms exist [35, 34, 20, 19], we are aware of only two exact algorithms for the maximum IUC problem in the literature [6]. The first is based on a linear and integer programming formulation of the problem where the open triangle property (a subgraph with three vertices and two edges) is formulated by linear constraints. Another one is a branch-and-bound (B&B) approach following the Russian doll search (RDS) method [31]. Given the computational challenge of the maximum IUC problem, these exact algorithms can only be used to solve problem instances of limited sizes with up to 1,024 vertices. For large instances, heuristic and metaheuristic algorithms provide a relevant alternative approach for finding good-enough solutions within an acceptable time frame. Unfortunately, such practical algorithms for the maximum IUC problem are still missing in the literature.

We fill the gap by proposing an effective heuristic algorithm to enrich the toolkit for solving the maximum IUC problem. The main contributions of this work are listed as follows.

- We propose a learning driven three-phase search (LDTPS) algorithm for the maximum IUC problem. Specifically, the algorithm uses a descent-based local search and a constrained swap-based tabu search to effectively examine the search space. It applies a frequency-based perturbation mechanism to diversify the search. It additionally relies on a probability learning strategy to collect helpful information during the search and guide solution construction.

- We carry out extensive experiments to evaluate LDTPS on the 83 2nd DIMACS benchmark graphs in the literature, and compare our results with two representative heuristic algorithms, a heuristic-based commercial solver (LocalSolver), and two best-performing exact algorithms. Computational results indicate that LDTPS performs significantly better than the reference algorithms on the benchmark instances. Following that, we present the applications of the LDTPS algorithm to the analysis of social networks. We also perform additional studies to investigate the important search strategies used in our algorithm.

- The code of the algorithm will be made publicly available, thus allowing researchers and practitioners to solve real-world problems that can be formulated by the maximum IUC model.

The rest of the paper is organized as follows. In the next section, we introduce the proposed LDTPS algorithm. Computational experiments and analyses are presented in Section 3 and Section 4, respectively. Conclusions and future works are discussed in the last section.

2. Learning driven three-phase search for the maximum IUC

In this section, we introduce our learning driven three-phase search (LDTPS) algorithm for the maximum IUC problem, which follows the general three-phase search framework [23, 10, 39, 14] enhanced with a probability learning strategy from reinforcement learning [38, 2, 32, 28].

2.1. Search space and evaluation function

The maximum IUC problem can be approximated by finding a series of k-IUC for increasing values of k (a k-IUC is an IUC of size k), just like for the maximum clique problem (in fact, the equivalent maximum independent set problem) [9]. Our LDTPS algorithm is designed to determine a k-IUC with a given size of k. Each time such a k-IUC is found, k is incremented by 1, and a new (larger) k-IUC is sought. This process is repeated until no k-IUC can be found. The final k-IUC is an approximation of the maximum IUC of the input graph G.

Given a graph $G = (V, E)$, the search space Ω explored by our LDTPS algorithm is composed of all subsets $S \subseteq V$ of size k (k-IUC) including both legal and illegal IUC as follows,

$$\Omega = \{S \subseteq V : |S| = k\} \tag{4}$$

To assess the quality of any candidate solution $S \in \Omega$, we use the following evaluation (fitness) function $f(S)$ (to be minimized) that counts the number of open triangles induced by S,

$$f(S) = \sum_{i,j,l \in S} t_{ijl} \tag{5}$$

where $t_{ijl} = 1$ if vertices i, j, l form an open triangle, and $t_{ijl} = 0$ otherwise.

Since $S \subseteq V$ is an IUC if and only if no set of three vertices from S induces an open triangle in G [6], we use Eq. (5) to compare candidate solutions. For two candidate solutions S_1 and S_2, S_1 is better than S_2 if $f(S_1) < f(S_2)$ (S_1 contains fewer open triangles than S_2). If $f(S) = 0$, there are no open triangles in S, and the candidate solution S is a legal k-IUC. If $f(S) > 0$, there are at least one open triangle in S; consequently, S is not a legal k-IUC. The LDTPS algorithm uses the evaluation function f to guide its search to find a legal k-IUC for a given k value.

2.2. Main scheme

The proposed LDTPS algorithm is presented in Algorithm 1, which includes two main features: 1) a three-phase search consisting of a descent-based local search, a constrained swap-based tabu search, and a frequency-based perturbation, and 2) a probability learning strategy consisting of a hybrid subset selection and a probability updating rule. Probability learning is based on two probability vectors P_0 and P_1, which indicate the probability for a vertex $i \in V$ to be part of the candidate solution S or to stay in the subset $V \setminus S$, respectively.

The algorithm first determines an initial IUC number k by a greedy clique procedure (line 2) and initializes the probability vectors P_0 and P_1. Based on P_0 and P_1 (lines 5,6), a starting IUC solution S is built by the hybrid subset selection strategy (line 7), which is then improved during the main 'while' loop. For each 'while' loop, the input IUC S, which is not necessarily a legal IUC, is first improved by the descent-based local search (line 10), and the new solution is used to update P_0 and P_1 (line 11). If S is not a legal k-IUC (i.e., $f(S) > 0$), S is further improved by the constrained swap-based tabu search (line 14, Algorithm 2) followed by the update of P_0 and P_1 (line 15). If S is a legal k-IUC (i.e., $f(S) = 0$), the best IUC S^* found so far and the IUC number k^* are updated (line 17). Then a larger (and not necessarily legal) IUC S size of $k+1$ is created, which serves as the new starting solution of the next round of LDTPS. Otherwise, the frequency-based perturbation procedure (line 21, Algorithm 3 and Section 2.6) is activated to create a new starting solution of the next LDTPS search.

2.3. Initialization and hybrid subset selection

Initialization. Our LDTPS algorithm requires an initial IUC number k, and the clique number provides a lower bound on the cardinality of a maximum IUC [6]. In general, the initial IUC number can be generated

Algorithm 1: Learning driven three-phase search for the maximum IUC problem (LDTPS).

```
Input: Graph $G = (V, E)$, cutoff time $t_{max}$, parametric constrained
       neighborhood coefficient $\tau$, maximum non-improving iterations of
       TS $td$, tabu tenure $tt$, reward factor for the correct group $\alpha$,
       penalization factor for the incorrect group $\beta$, compensation factor
       for the expected group $\gamma$
Output: Best $k^*$-IUC found during the search $S^*$, $k^*$
1  begin
2      $(S,\ k) \leftarrow$ Greedy_initialization$(S,\ k)$            /* Section 2.3 */
3      $S^* \leftarrow S,\ k^* \leftarrow k$
4      $k \leftarrow k + 1$
5      for $i = 1$ to $n$ do
6          $p_0(i) = p_1(i) = 1/2$ /* Init. probability vectors $P_0$, $P_1$ */
7      $S \leftarrow$ Hybrid_subset_select$(S,\ P_0,\ P_1)$            /* Section 2.3 */
8      while $time() \leq t_{max}$ do
9          $S' \leftarrow S$
10         $S \leftarrow$ Descent-based_local_search$(S)$            /* Section 2.4 */
11         $(P_0,\ P_1) \leftarrow$ Probability_update$(S,\ S',\ \alpha,\ \beta,\ \gamma)$ /* Section 2.7 */
12         if $S$ is not a legal $k$-IUC then
13             $S' \leftarrow S$
14             $S \leftarrow$ Constrained_swap-based_tabu_search$(S,\ \tau,\ td,\ tt)$
                /* Algorithm 2, Section 2.5 */
15             $(P_0,\ P_1) \leftarrow$ Probability_update$(S,\ S',\ \alpha,\ \beta,\ \gamma)$
16         if $S$ is a legal $k$-IUC then
17             $S^* \leftarrow S,\ k^* \leftarrow k$
18             $k \leftarrow k + 1$
19             $S \leftarrow$ Hybrid_subset_select$(S,\ P_0,\ P_1)$
20         else
21             $S \leftarrow$ Frequency-based_perturb$(S)$   /* Algorithm 3, Section
                2.6 */
22 return $S^*,\ k^*$
```

by any method that does not exceed the lower bounds. In our method, we greedily construct a clique and return its size as the initial IUC number k. Starting with an empty set S, a random vertex $i \in V$ is added to S in a step-by-step way, ensuring that vertex i is adjacent to all the vertices of S (ties are broken randomly). This process is repeated until no more such vertices can be added to S. We use the final clique S and its cardinality to initialize the starting IUC solution and the IUC number. The time complexity of the procedure is bounded by $\mathcal{O}(n)$.

Hybrid subset selection. The hybrid subset selection strategy is invoked to initialize each round of LDTPS with a new (larger) IUC. This strategy equiprobably combines the random selection and the greedy selection method. The random method selects k vertices to form S at random, regardless of the probability vectors P_0 and P_1. The greedy method allocates k vertices to the subset S such that P_1 is greater than P_0. The hybrid subset selection strategy has the advantage of flexibility by switching back and forth between greediness and randomness with a time complexity of $\mathcal{O}(k)$.

2.4. Descent-based local search

The descent-based local search (DLS) in our LDTPS algorithm aims at finding, from a given initial solution, new solutions of better quality in terms of the evaluation function $f(S)$ given by Eq. (5). DLS iteratively makes transitions from the current solution to a neighboring solution according to a given neighborhood and terminates when no better neighboring solution exists, i.e., when a local optimum is reached. At each iteration, DLS explores the complete swap-based neighborhood N_{swap} (see Eq. (6), Section 2.5.1 for details) and selects the best neighboring solution S' to replace the current solution S (i.e., $f(S') < f(S)$). This process is repeated until no better solutions can be found in the neighborhood, then DLS stops and returns the last solution (a local optimum), which is used as the starting solution for the next search procedure (CNTS, Section 2.5). DLS also terminates when a legal k-IUC is found, i.e., when $f(S') = 0$, and returns the IUC found S and the IUC number k. The time complexity of each iteration of DLS is bounded by $\mathcal{O}(n^2)$.

2.5. Constrained swap-based tabu search

The constrained swap-based tabu search (CSTS, Algorithm 2) is the key local optimization procedure used in our LDTPS algorithm. Tabu search [12] is a well-known metaheuristic that has been successfully applied to many

Algorithm 2: Constrained swap-based tabu search (CSTS).

```
Input: $G = (V, E)$, parametric constrained neighborhood coefficient $\tau$,
       maximum non-improving iterations of TS $td$, tabu tenure $tt$
Output: Best IUC found $S_{best}$
begin
    $iter \leftarrow 0$  /* Counter of consecutive non-improving iterations */
    $S_{best} \leftarrow S$                      /* Best IUC found so far */
    for $i = 1$ to $n$ do
        $tl_0(i) \leftarrow 0$, $tl_1(i) \leftarrow 0$  /* Initialize tabu lists $tl_0$ and $tl_1$ */
    for $i = 1$ to $n$ do
        Initial $\delta_0(i)$ and $\delta_1(i)$
    while $iter < td$ do
        //Construct the parametric constrained swap-based neighborhood, Section 2.5.1
        for $\forall u \in S$ do
            $d_S(u) \leftarrow \delta_0(u) + \delta_1(u)/2$
        for $\forall v \in V \setminus S$ do
            $d_{V \setminus S}(v) \leftarrow \delta_0(v) + \delta_1(v)/2$
        $d_{max} \leftarrow$ the $i$-th element in $d'_S$, where $i = (1 - \tau) \times k$ and $d'_S$
          represents $d_S$ in an ascending order
        $d_{min} \leftarrow$ the $j$-th element in $d'_{V \setminus S}$, where $j = \tau \times (n - k)$ and $d'_{V \setminus S}$
          represents $d_{V \setminus S}$ in an ascending order
        $A = \{u | u \in S, d_S(u) \geq d_{max}\}$
        $B = \{v | v \in V \setminus S, d_{V \setminus S}(v) \leq d_{min}\}$
        //Make a transition (move), Section 2.5.1
        Select the best admissible $swap(u, v)$ move from the neighborhood
          $CN_{swap}$ with the smallest move gain value $\triangle_f(u, v)$
        $S \leftarrow S \setminus \{u\} \cup \{v\}$
        Update tabu lists $tl_0$ and $tl_1$ with tabu tenure $tt$
        for $i = 1$ to $n$ do
            Update $\delta_0(i)$ and $\delta_1(i)$
        if $S$ is a legal $k$-IUC then
            return $S$
        //Update the best solution found so far
        if $f(S) < f(S_{best})$ then
            $S_{best} \leftarrow S$
            $iter \leftarrow 0$
        else
            $iter \leftarrow iter + 1$
return $S_{best}$
```

difficult combinatorial optimization problems [39, 25, 24, 37]. Our CSTS procedure iteratively replaces the current solution with a neighboring solution from the parametric constrained swap-based neighborhood CN_{swap} defined in Section 2.5.1. The procedure terminates when a legal k-IUC is found or if the maximum non-improving iterations td is reached.

2.5.1. Parametric constrained swap-based neighborhood and explorations

To explore the search space Ω of k-IUC, CSTS iteratively transforms the current solution to a neighboring solution by swapping a vertex of S with another vertex of $V \setminus S$. Clearly, such an unconstrained swap move generates a large neighborhood of size $k \times (n-k)$. This unconstrained (complete) neighborhood is not sufficiently focused because it includes many unpromising neighboring solutions of bad quality. For this reason, we introduce a parametric constrained swap-based neighborhood, which is both more focused and smaller in size.

Let S be the current IUC, and the swap move $swap(u,v)$ applied to solution S creates a new IUC S' by exchanging vertices $u \in S$ and $v \in V \setminus S$. Let $S \oplus swap(u,v)$ designate the neighboring solution S' obtained by applying $swap(u,v)$ to S, the complete swap-based neighborhood N_{swap} (i.e., the set of neighboring solutions) induced by $swap(u,v)$ is given by,

$$N_{swap}(S) = \{S' : S' = S \oplus swap(u,v) = S \setminus \{u\} \cup \{v\}, u \in S, v \in V \setminus S\} \quad (6)$$

The key concept related to a swap move is the move gain, which indicates the variation of the value of the evaluation function $f(S)$ (Eq. (5)). To quickly determine the move gain of a swap move, we devise the first incremental neighborhood evaluation technique for the maximum IUC problem that considerably speeds up our CSTS procedure. This technique relies on two n-dimensional integer vectors δ_0 and δ_1, where element $\delta_0(i)$ represents the number of open triangles formed by a vertex $i \in V$ with any two vertices $j, l \in S$, where j and l are adjacent, i.e., $\sum_{i \in V} t_{ijl}$, $\forall j, l \in S, \{j,l\} \in E, i, j, l = 1, \ldots, n, i \neq j \neq l$. Similarly, element $\delta_1(i)$ denotes the number of open triangles constructed by $i \in V$ with $j, l \in S$, where j and l are not adjacent, i.e., $\sum_{i \in V} t_{ijl}$, $\forall j, l \in S, \{j,l\} \notin E, i, j, l = 1, \ldots, n, i \neq j \neq l$. With vectors δ_0 and δ_1, for each candidate neighboring solution S' obtained after applying a $swap(u,v)$ to S, the change in the evaluation function value (move gain) can be quickly computed in $\mathcal{O}(1)$ as below,

$$\triangle_f(u,v) = (\delta_0(v) + \delta_1(v)/2) - (\delta_0(u) + \delta_1(u)/2) - \sum_{i \in S} t_{iuv} \tag{7}$$

where $\sum_{i \in S} t_{iuv}$ denotes the total number of open triangles formed by $u \in S$ and $v \in V \setminus S$ with a vertex $i \in S$. Let $d_V(i) = \delta_0(i) + \delta_1(i)/2$, $i \in V$, Eq. (7) can be conveniently rewritten as follows,

$$\triangle_f(u,v) = d_{V \setminus S}(v) - d_S(u) - \sum_{i \in S} t_{iuv} \tag{8}$$

Thus, a negative (positive) move gain indicates a better (worse) neighboring solution S' compared to S, while a zero move gain corresponds to a neighboring solution of equal quality.

From Eq. (8), we note that the evaluation function value of the resulting solution, i.e., $f(S') = f(S) + \triangle_f(u,v)$, depends on the values of $d_{V \setminus S}(v)$, $d_S(u)$, and $\sum_{i \in S} t_{iuv}$, respectively. To minimize $f(S')$, we prefer to replace a vertex $u \in S$ of a large $d_S(u)$ value with a vertex $v \in V \setminus S$ of a small $d_{V \setminus S}(v)$ value. Thus, we select swapped vertices from a subset $A \subseteq S$ containing vertices with large $d_S(u)$ values and a subset $B \subseteq V \setminus S$ consisting of vertices with small $d_{V \setminus S}(v)$ values. Let d'_S ($d'_{V \setminus S}$) denote the vectors d_S ($d_{V \setminus S}$) in an ascending order, d_{max} be the i-th element in d'_S, and d_{min} be the j-th element in $d'_{V \setminus S}$, where $i = (1 - \tau) \times k$ and $j = \tau \times (n - k)$ where τ ($0 < \tau \leq 1$) is the parametric constrained neighborhood coefficient that dynamically scales the size of the neighborhood ($\tau = 1$ corresponds to the complete neighborhood). The two subsets A and B are then defined by,

$$A = \{u | u \in S, d_S(u) \geq d_{max}\} \tag{9}$$

$$B = \{v | v \in V \setminus S, d_{V \setminus S}(v) \leq d_{min}\} \tag{10}$$

Our constrained swap neighborhood $CN_{swap}(S)$ is thus composed of the specific swap moves induced by subsets A and B as follows,

$$CN_{swap}(S) = \{S' : S' = S \oplus swap(u,v) = S \setminus \{u\} \cup \{v\}, u \in A, v \in B\} \tag{11}$$

This constrained swap-based neighborhood is clearly much smaller than the complete swap neighborhood N_{swap} defined by Eq. (6). It is also more focused because neighboring solutions of bad quality are excluded. We will compare the performance of the two neighborhoods in Section 4.1.

As shown in Algorithm 2, at each iteration, CSTS constructs a parametric constrained swap-based neighborhood CN_{swap} by identifying two subsets A and B with regard to S (lines 9-17). The best admissible $swap(u, v)$ ($u \in A$, $v \in B$, u and v are not prohibited by the tabu list) (ties are broken randomly) is then selected from CN_{swap} with the smallest move gain $\triangle_f(u, v)$ (line 19). The tabu list, which is defined as two n-dimensional integer vectors tl_0 and tl_1, is used to prevent the search from short-term cycling. It is first set to 0 for each $i \in V$ to ensure that no vertex is forbidden by the tabu list (lines 4-5). Once a $swap(u, v)$ is performed (line 20), vertices u and v are added to the tabu list so as to prevent them from being selected again for the next tt iterations (line 21). If the new solution is a legal k-IUC (i.e., $f(S) = 0$), the procedure returns the k-IUC found and stops (lines 24-25). Otherwise, the best IUC found is updated whenever a better solution is achieved and the search continues to its next iteration (lines 26-31). The tabu status of a move is overridden if it leads to a solution better than all visited solutions (aspiration criterion). A $swap(u, v)$ is thus considered admissible if neither u nor v is marked as tabu or if the aspiration criterion is met. The time complexity of CSTS is bounded by $\mathcal{O}(n \times k)$.

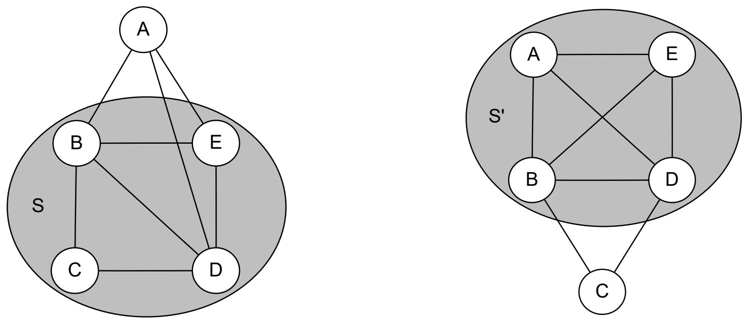
Once a $swap(u, v)$ move is performed, for each vertex $i \in V$ where the open triangle formed by i with u or v, the values of its vectors $\delta_0(i)$ and $\delta_1(i)$ are updated in $\mathcal{O}(1)$ as follows,

$$\delta_0(i) = \begin{cases} \delta_0(i) - 1, & t_{iuw} = 1, u, w \in S, i \neq w, \{u, w\} \in E \\ \delta_0(i) + 1, & t_{ivw} = 1, v \in V \setminus S, w \in S, i \neq w \neq u, \{v, w\} \in E \\ \delta_0(i), & otherwise. \end{cases} \tag{12}$$

$$\delta_1(i) = \begin{cases} \delta_1(i) - 1, & t_{iuw} = 1, u, w \in S, i \neq w, \{u, w\} \notin E \\ \delta_1(i) + 1, & t_{ivw} = 1, v \in V \setminus S, w \in S, i \neq w \neq u, \{v, w\} \notin E \\ \delta_1(i), & otherwise. \end{cases} \tag{13}$$

where $t_{iuw} = 1$ (or $t_{ivw} = 1$) denotes the open triangle formed by vertices i, u, w (or vertices i, v, w).

Fig. 1 shows a simple example on a graph with five vertices. In Fig. 1(a), $S = \{B, C, D, E\}$ is an IUC with $\delta_0 = \{2, 0, 2, 0, 2\}$ and $\delta_1 = \{0, 2, 0, 2, 0\}$,



(a) An illegal IUC (b) A legal IUC

Figure 1: An example shows that from an IUC $S = \{B, C, D, E\}$, a legal IUC $S' = \{A, B, D, E\}$ is obtained by swapping vertices $A \in S$ and $C \in V \setminus S$ through a swap move.

which is not a legal IUC ($f(S) = 2$). We see that swapping C and A ($\triangle_f(C, A) = -2$) leads to a legal IUC $S' = \{A, B, D, E\}$ ($f(S') = f(S) + \triangle_f(C, A) = 0$) in Fig. 1(b), which is better than other swaps, i.e., B and A ($\triangle_f(B, A) = 0$), D and A ($\triangle_f(D, A) = 0$), or E and A ($\triangle_f(E, A) = 0$). After swapping C and A, the vectors δ_0 and δ_1 are updated as $\delta'_0 = \{0, 0, 4, 0, 0\}$ and $\delta'_1 = \{0, 0, 0, 0, 0\}$.

2.6. Frequency-based perturbation

To encourage our LDTPS algorithm to explore distant new regions in the search space Ω when it is unable to find a legal k-IUC during the search, we perturb the current solution S to start a new search from a different initial point. To make an informed perturbation, we use a long-term frequency memory. In this memory, we keep track of how many times a vertex has been relocated during the search. To maintain the frequency $freq(i)$ of each vertex $i \in V$, we use the following rules, 1) we first set for each vertex $i \in V$, $freq(i) = 0$, 2) during the search, each time a vertex i is removed from or added to the current solution S, $freq(i)$ is incremented by 1, i.e., $freq(i) = freq(i) + 1$, and 3) at the end of each round of LDTPS, if for $i \in V, freq(i) > k$, we reset $freq(i) = 0$.

We create a perturbed (new) solution S by the frequency information (Algorithm 3). First, we calculate the average frequency $freq_{avg}$ (i.e., $freq_{avg} = \sum_{i \in S} freq(i)/k$) from the best solution obtained in the current search round (line 3). The solution destruction process (lines 4-8) is triggered

Algorithm 3: Frequency-based perturbation procedure.

```
    Input: $G = (V, E)$
    Output: Perturbed IUC $S$
1   begin
2       $l \leftarrow k$
3       Calculate the average frequency $freq_{avg} = \sum_{i \in S} freq(i)/k$
4       //Solution destruction process
5       for $i = 1$ to $n$ do
6           if $freq(i) < freq_{avg}$ then
7               $S \leftarrow S \setminus \{i\}$
8               $l \leftarrow l - 1$
9       //Solution expansion process
10      while $l < k$ do
11          Assign a random vertex $i$ to $S$
12          $l \leftarrow l + 1$
13  return $S$
```

to remove from S the vertices i whose frequency $freq(i)$ is less than $freq_{avg}$. The solution repair process (lines 9-12) is then used to extend the current partial solution S by adding vertices at random until S contains exactly k vertices. The time complexity of the procedure is bounded by $\mathcal{O}(n + k)$.

2.7. Probability updating rule

After each descent local search and tabu search (lines 11 and 15, Algorithm 1), the LDTPS algorithm performs a probability learning mechanism to update the probability vectors P_0 and P_1, following the idea of [38] for grouping problems. We compare the improved solution and the starting solution, and update P_0 and P_1 by observing whether a vertex is moved from its original subset to another subset.

For each vertex $i \in V$, we compare its located subset in S or in S'. If a vertex i stays in its original subset s, then we reward this subset (the correct subset) and update the probability vector $p_j(i)$ as follows,

$$p_j(i) = \begin{cases} \alpha + (1-\alpha) \cdot p_j(i), & j = s \\ (1-\alpha) \cdot p_j(i), & j = 1 - s. \end{cases} \quad (14)$$

where α $(0 < \alpha < 1)$ is a reward factor for the correct subset, and $s = 0$ (or $s = 1$) denotes subset $V \setminus S$ (or S).

When a vertex i moves from a subset s of the input solution to a new subset $1 - s$ of the improved solution, we penalize the original subset s (the incorrect subset), compensate the new subset $1 - s$ (the expected subset), and finally update the probability vector $p_j(i)$ as follows,

$$p_j(i) = \begin{cases} (1-\gamma) \cdot (1-\beta) \cdot p_j(i), & j = s \\ \gamma + (1-\gamma) \cdot \beta + (1-\gamma) \cdot (1-\beta) \cdot p_j(i), & j = 1 - s. \end{cases} \tag{15}$$

where β $(0 < \beta < 1)$ is a penalization factor for the incorrect subset, and γ $(0 < \gamma < 1)$ is a compensation factor for the expected subset. The time complexity of the probability updating rule is bounded by $\mathcal{O}(n)$.

3. Computational experiments

In this section, we carry out extensive experiments to evaluate the proposed LDTPS algorithm with respect to the best-performing exact and heuristic algorithms in the literature.

3.1. Benchmark graphs

We use 83 benchmark graphs[1] from the well-known 2nd DIMACS Implementation Challenge for clique problems [17]. These graphs have 28 to 4,000 vertices and 210 to 4,619,898 edges that encompass various real-world problems (e.g., coding theory, fault diagnosis, Keller's conjecture on tilings using hypercubes, and the Steiner triple problem) as well as random graphs. Then, we divide these graphs into three categories: Training Set, which contains 15 graphs for parameter calibration, Test Set I (45 small and medium graphs) and Test Set II (23 large graphs) for algorithm evaluation. Table 1 shows the main features of all the training and tested graphs.

3.2. Parameter setting

The proposed LDTPS algorithm requires six parameters (τ, td, tt, α, β, and γ), which are calibrated by the automatic parameter configuration package IRACE [4, 22]. The parameter calibration process was conducted on the 15 graphs from the Training set. The training budget was set to 1000

[1] `http://archive.dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/`

Table 1: Main features of the 86 benchmark graphs from the well-known 2nd DIMACS Implementation Challenge.

Graph	$\|V\|$	$\|E\|$	Graph	$\|V\|$	$\|E\|$
Training Set (15)					
brock400_4	400	59,765	hamming10-4	1,024	434,176
brock800_1	800	207,505	johnson16-2-4	120	5,460
brock800_2	800	208,166	san200_0.7_1	200	13,930
brock800_3	800	207,333	san400_0.5_1	400	39,900
c-fat200-2	200	3,235	san400_0.7_2	400	55,860
c-fat500-5	500	23,191	san400_0.7_3	400	55,860
gen400_p0.9_55	400	71,820	san400_0.9_1	400	71,820
hamming10-2	1,024	518,656			
Test Set I (45)					
brock200_1	200	14,834	johnson32-2-4	496	107,880
brock200_2	200	9,876	keller4	171	9,435
brock200_3	200	12,048	MANN_a9	45	918
brock200_4	200	13,089	MANN_a27	378	70,551
brock400_1	400	59,723	p_hat300-1	300	10,933
brock400_2	400	59,786	p_hat300-2	300	21,928
brock400_3	400	59,681	p_hat300-3	300	33,390
c-fat200-1	200	1,534	p_hat500-1	500	31,569
c-fat200-5	200	8,473	p_hat500-2	500	62,946
c-fat500-1	500	4,459	p_hat500-3	500	93,800
c-fat500-2	500	9,139	p_hat700-1	700	60,999
c-fat500-10	500	46,627	p_hat700-2	700	121,728
DSJC500.5	500	62,624	p_hat700-3	700	183,010
gen200_p0.9_44	200	17,910	san200_0.7_2	200	13,930
gen200_p0.9_55	200	17,910	san200_0.9_1	200	17,910
gen400_p0.9_65	400	71,820	san200_0.9_2	200	17,910
gen400_p0.9_75	400	71,820	san200_0.9_3	200	17,910
hamming6-2	64	1,824	san400_0.7_1	400	55,860
hamming6-4	64	704	sanr200_0.7	200	13,868
hamming8-2	256	31,616	sanr200_0.9	200	17,863
hamming8-4	256	20,864	sanr400_0.5	400	39,984
johnson8-2-4	28	210	sanr400_0.7	400	55,869
johnson8-4-4	70	1,855			
Test Set II (23)					
brock800_4	800	207,643	flat1000_76_0	1,000	246,708
C1000.9	1,000	450,079	keller5	776	225,990
C2000.5	2,000	999,836	keller6	3,361	4,619,898
C2000.9	2,000	1,799,532	p_hat1000-1	1,000	122,253
C4000.5	4,000	4,000,268	p_hat1000-2	1,000	244,799
DSJC500.1	500	12,458	p_hat1000-3	1,000	371,746
DSJC500.9	500	112,437	p_hat1500-1	1,500	284,923
DSJC1000.1	1,000	49,629	p_hat1500-2	1,500	568,960
DSJC1000.5	1,000	249,826	p_hat1500-3	1,500	847,244
DSJC1000.9	1,000	449,449	r1000.1c	1,000	485,090
flat1000_50_0	1,000	245,000	r1000.5	1,000	238,267
flat1000_60_0	1,000	245,830			

Table 2: Parameter setting of the LDTPS algorithm.

Parameter	Section	Description	Calibrated values	Final value
τ	§2.5	parametric constrained neighborhood coefficient	[0.1, . . . , 0.7, 0.9]	0.1
td	§2.5	maximum non-improving iterations of TS	$[0.5, \ldots, 2.0, 2.5] \times 10^4$	5000
tt	§2.5	tabu tenure	[25, . . . , 185, 205]	85
α	§2.7	reward factor for the correct subset	[0.1, . . . , 0.8, 0.9]	0.4
β	§2.7	penalization factor for the incorrect subset	[0.1, . . . , 0.8, 0.9]	0.1
γ	§2.7	compensation factor for the expected subset	[0.1, . . . , 0.8, 0.9]	0.5

runs, and the cutoff time of each run was limited to 600 seconds. Table 2 shows the range of the calibrated parameter values and the final values provided by IRACE. The final parameter values define the default setting of LDTPS, which was consistently used for all the experiments. We will discuss the impact of the parameters on the performance of LDTPS in Section 4.3.

3.3. Experimental setting

The LDTPS algorithm was programmed in C++[2] and compiled by g++ 10.2.1 with the optimization option '-O3'. All the experiments were conducted on a Linux operating system with an Intel Xeon E5-2695 v4 processor (2.10 GHz and 12GB RAM). When we solved the DIMACS machine benchmarking program dfmax.c[3] without the compilation optimization flag, the runtime on our machine was 0.05, 0.39, 2.42, and 9.19 seconds for graphs r200.5, r300.5, r400.5, and r500.5, respectively.

To evaluate our algorithm, we compare it with the following state-of-the-art reference methods from the literature. First, we implemented the maximum IUC model (see Section 1) and solved it with a commercial heuristic solver called LocalSolver 10.5 [3, 21] in one run with a time limit of 60 minutes. Second, we used the two most recent exact methods in the literature [6]: 1) IP, an integer programming based approach, and 2) RDS, a Russian doll search (an effective combinatorial branch-and-bound algorithm). For IP, we follow [6] and solve the maximum IUC model using the newest CPLEX solver 22.1 with a cutoff time of 600 minutes.

To further enhance the comparative study, we implemented two other metaheuristic algorithms: a restart simulated annealing algorithm (denoted

[2]The code of our LDTPS algorithm will be made publicly available at: `https://github.com/hellozhilu/LDTPS`, upon publication of the paper.

[3]`http://archive.dimacs.rutgers.edu/pub/dsj/clique/`.

by RSA) and a genetic algorithm (denoted by GA). For both algorithms, we reused as many search components as possible from our LDTPS algorithm (fast evaluation function technique, constrained swap neighborhood, key data structures, etc.). The implementation details of RSA and GA are provided in Appendix A. Given the stochastic nature of RSA, GA, and LDTPS, we performed 20 independent runs per graph with a cutoff time of 60 minutes per run.

To ensure a fair comparison, RSA, GA, LocalSolver heuristic, IP with CPLEX, and our LDTPS algorithm were run on the same computing platform mentioned above with their default parameter settings. The numerical results of the RDS algorithm were taken directly from [6].

In addition to the maximum IUC problem, we also report results on the equivalent maximum multi-partite clique problem (MPC). A subset of vertices S is a multi-partite clique if S can be partitioned into $1 \leq k \leq |S|$ independent sets $S_1, \ldots, S_k$ such that any two vertices from two different subsets are adjacent. The maximum MPC problem is to find an MPC of maximum cardinality. Note that an MPC is an independent set if $k = 1$ and is a clique if $k = |S|$. It is obvious that S is a maximum IUC of G if and only if S is a maximum MPC of the complement graph $\overline{G}$.

Hence, RSA, GA, LocalSolver heuristic, IP with CPLEX, and our LDTPS algorithm were also used to solve the maximum MPC problem by computing the IUC number of the complement graph $\overline{G}$.

3.4. Performance assessments

We present the computational results of the proposed LDTPS algorithm along with those of the reference methods for both the maximum IUC problem and the maximum MPC problem on the 83 DIMACS2 benchmark graphs (including the 45 graphs from Test Set I and the 23 large graphs from Test Set II).

Note that 1) the results of the IP-based approach and the RDS algorithm are missing on Test Set II because the CPLEX solver did not find feasible solutions on these large graphs within the limit of 10 hours, and the RDS algorithm did not report results on these graphs in [6], and 2) the results of the LocalSolver heuristic, the IP-based approach, and the RDS algorithm were obtained from a single run, while the other algorithms were run 20 times. To make a fair comparison, we use our average results (instead of the best results) to compare with the results of the single-run algorithms (IP,

Table 3: Summary results for the maximum IUC problem and the maximum MPC problem on Test Set I (45 graphs) and Test Set II (23 large graphs) of our LDTPS algorithm, the restart simulated annealing RSA, the genetic algorithm GA, the LocalSolver heuristic [21], the IP-based approach [6], and the RDS algorithm [6].

Benchmark set	Algorithm pair	Indicator	#Better	#Equal	#Worse	p-value
		The maximum IUC problem				
Test Set I (45 graphs)	**LDTPS** *vs.* IP [6]	sol	24	20	1	1.76e-05
	LDTPS *vs.* RDS [6]	sol	23	19	3	1.07e-04
	LDTPS *vs.* LocalSolver [21]	sol	26	19	0	8.24e-06
	LDTPS *vs.* RSA	sol_{best}	6	37	2	7.78e-02
		sol_{avg}	16	25	4	6.57e-03
	LDTPS *vs.* GA	sol_{best}	39	6	0	5.17e-08
		sol_{avg}	40	5	0	3.57e-08
Test Set II (23 graphs)	**LDTPS** *vs.* LocalSolver [21]	sol	23	0	0	2.38e-07
	LDTPS *vs.* RSA	sol_{best}	19	2	2	1.80e-04
		sol_{avg}	19	0	4	6.03e-05
	LDTPS *vs.* GA	sol_{best}	23	0	0	2.38e-07
		sol_{avg}	23	0	0	2.38e-07
		The maximum MPC problem				
Test Set I (45 graphs)	**LDTPS** *vs.* IP [6]	sol	24	21	0	1.79e-05
	LDTPS *vs.* RDS [6]	sol	18	27	0	1.96e-04
	LDTPS *vs.* LocalSolver [21]	sol	22	23	0	3.72e-05
	LDTPS *vs.* RSA	sol_{best}	7	38	0	1.58e-02
		sol_{avg}	13	32	0	2.18e-03
	LDTPS *vs.* GA	sol_{best}	38	7	0	7.71e-08
		sol_{avg}	42	3	0	1.65e-08
Test Set II (23 graphs)	**LDTPS** *vs.* LocalSolver [21]	sol	23	0	0	2.38e-07
	LDTPS *vs.* RSA	sol_{best}	23	0	0	2.38e-07
		sol_{avg}	23	0	0	2.38e-07
	LDTPS *vs.* GA	sol_{best}	23	0	0	2.38e-07
		sol_{avg}	23	0	0	2.38e-07

RDS, LocalSolver) while for multi-run algorithms (RSA, GA, LDTPS), we compare them using both the best and average results.

Table 3 summarizes the overall comparison while the detailed results are reported in Tables B.2 to B.5 of Appendix B. In Table 3, column 1 shows the names of the benchmark sets. Column 2 indicates the pairs of compared algorithms. Column 3 provides the quality indicator in terms of the best and average statistics. Columns 4-6 show the number of instances on which LDTPS reached a better (#Better), equal (#Equal), or worse (#Worse) results in terms of each quality indicator. Furthermore, the last column provides the p-values from the Wilcoxon signed-rank tests with a confidence level of 95% to assess whether there exists a statistically significant performance difference between LDTPS and each reference algorithm.

According to the summarized results of Table 3, LDTPS outperforms all the reference algorithms. With respect to the exact algorithms, LDTPS dominates the IP-based approach and the RDS algorithm by achieving better or equal results in all but 4 cases. Compared to the heuristic algorithms, LDTPS dominates all of them, always achieving better or equal results in terms of the best and average results, with only a few exceptions. The small p-values (far less than 0.05) confirm that the performance differences are statistically significant between LDTPS and each compared method, except for one case (LDTPS *vs.* RSA concerning $\mathrm{sol}_{\mathrm{best}}$ for the maximum IUC problem) where the p-value slightly exceeds 0.05.

Specifically, on Test Set I (45 graphs), Tables B.2 and B.3 of Appendix B show that our LDTPS algorithm achieves the best-known results for all graphs, except 4 instances for the maximum IUC problem. The average results obtained by LDTPS also equal or surpass the best-known results in the literature in most cases. In particular, for the maximum IUC problem, LDTPS attains 19 out of 20 instances with known optimal solutions, while improving the best-known results (new lower bounds) in 14 out of 25 instances. For the maximum MPC problem, LDTPS reaches all 20 known optimal solutions, and improves the best-known results for 13 out of 25 instances with unknown optimal solutions. Regarding stability, LDTPS can consistently achieve its best results for each run (100% success rate) for 42 instances for the maximum IUC problem (93.3%), and all 45 instances for the maximum MPC problem (100.0%). In terms of computational time, LDTPS achieves its best results rather quickly by requiring a similar or shorter time to find equal or better solutions. The LocalSolver heuristic fails to find feasible solutions over 1 hour for 10 (7) instances for the maximum IUC (MPC) problem. The IP-based approach fails to find feasible solutions over 10 hours for 11 (8) instances, while the RDS algorithm fails to optimally solve 26 (26) instances over 10 hours for both problems. On Test Set II (23 large graphs), Tables B.4 and B.5 of Appendix B show that our LDTPS algorithm outperforms the reference algorithms. Particularly, LDTPS finds new lower bounds for 21 (23) out of 23 (23) instances for the maximum IUC (MPC) problem, and even its average results surpass the best results of the compared algorithms in most cases. In terms of stability and computational time, LDTPS achieves highly success rates and requires shorter time to find the best solutions than the compared algorithms. The LocalSolver heuristic can only find 3 (6) feasible solutions for the maximum IUC (MPC) problem.

Algorithm 4: Greedy clustering algorithm.

```
   Input: $G = (V, E)$
   Output: Clustering $C = \{C_1, ..., C_{k^*}\}$
1  begin
2      $S^*, k^* \leftarrow$ the best $k^*$-IUC found by the LDTPS algorithm
        /* Algorithm 1 */
3      Let $C_1, ..., C_{k^*}$ be the sets of vertices corresponding to the different
        connected components of the best $k^*$-IUC
4      while $V \setminus \cup_{i=1}^{k^*} C_i \neq \emptyset$ do
5          Find $v \in V \setminus \cup_{i=1}^{k^*} C_i$ and $C_i$ with the largest size of $N_G(v) \cap C_i$
            /* $N_G(v)$ is the set of neighbors of $v$ */
6          $C_i = C_i \cup v$
7  return $C = \{C_1, ..., C_{k^*}\}$
```

3.5. Application to real-life social network analysis

In this section, the proposed LDTPS algorithm is applied to the analysis of real-world social networks. The result of LDTPS (i.e., the independent unions of cliques) is used as a seed for a network clustering method that partitions the nodes of the network into clusters. Following [7], we use a greedy clustering algorithm and evaluate its performance on the three popular networks: Zachary's karate club, Terrorist network compiled by Krebs, and College football network.

The greedy clustering algorithm is outlined in Algorithm 4. We first compute an IUC by our LDTPS algorithm, which typically results in several cliques $C_1, \ldots, C_{k^*}$ corresponding to different connected components in the graph G. Each clique C_i $(i = 1, ..., k^*)$, then forms an initial seed. For each iteration of the 'while' loop, we select an unassigned node v with the largest number of neighbors in one of the cliques C_i and assign v to C_i. The procedure is repeated until all nodes are assigned to a cluster.

3.5.1. Zachary's karate club

Zachary's karate club is the social network of the well-known karate club investigated by Zachary over three years [36]. The network has 34 members (nodes) and 78 edges that represent the friendships between the club members (see Fig. 2). During the study period, a political conflict arose between the club president (node 34, John), and the instructor (node 1, Mr. Hi). Later, as a result of the conflict, the club split into two parts (with 16 and 18 members). Zachary recorded a network of friendships among the club

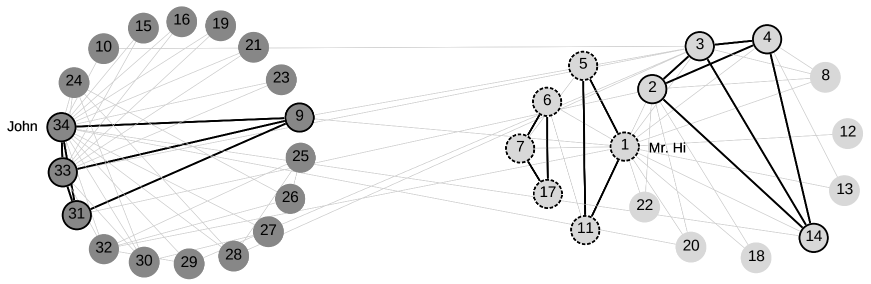


Figure 2: Application to the Zachary's karate club network.

members just before the split, so this real-life example is often used as a benchmark for network clustering algorithms.

Fig. 2 shows the results of our network clustering algorithm on the karate club network. The black edges constitute the cliques provided by LDTPS, and the gray edges indicate the other links. Our clustering algorithm finds 4 clusters, one corresponding to John's faction (dark-gray nodes) and the other three coinciding with Mr. Hi's faction (light-gray nodes). Mr. Hi's faction contains a group of 5 students nodes 5, 6, 7, 11, 17) who only interact with themselves and Mr. Hi (the corresponding nodes are indicated in dashed circles), and the other cluster contains the remaining light-gray nodes. Our clustering can be considered as an improvement over the results reported in [36] because it accurately identifies the two factions and distinguishes a distinct subgroup within one faction. Additionally, it is comparable to the findings of [7].

3.5.2. Terrorist network compiled by Krebs

Created by Krebs, the terrorist interaction network [18] used the available data about the terrible events of September 11, 2001. The network consists of 62 nodes representing terrorists involved in the attacks, and the edges correspond to pairs of individuals known to have interacted in the past. A total of 153 interactions were observed. The resulting clusters are shown in Fig. 3. The algorithm finds 5 clusters, three of which correspond to the actual hijackers and roughly to the WTC North, Pentagon, Pennsylvania, and WTC South attacks. The remaining two consist of the other association of hijackers. Our results successfully identified the actual hijackers, and the rest of the clusters are comparable to the results of [7].

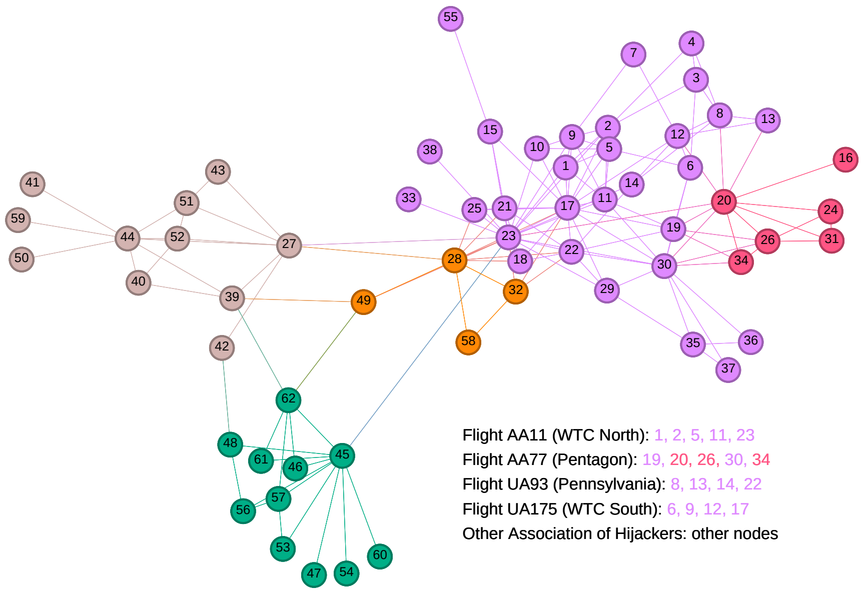

Flight AA11 (WTC North): 1, 2, 5, 11, 23
Flight AA77 (Pentagon): 19, 20, 26, 30, 34
Flight UA93 (Pennsylvania): 8, 13, 14, 22
Flight UA175 (WTC South): 6, 9, 12, 17
Other Association of Hijackers: other nodes

Figure 3: Application to the Terrorist network compiled by Krebs.

3.5.3. College football network

Finally, we consider a more complex real-world network with established community structures. The network shows the schedule of United States football games between Division IA colleges for the regular season in the fall of 2000 [11]. The known communities are defined by conferences, each containing about 8 to 12 teams. In general, teams from the same conference are more likely to play each other than teams from other conferences. There are also some independent teams that do not belong to any conference.

Fig. 4 shows the clustering results with our method on the football network. Nodes represent teams, colored differently to indicate different conferences, and links show regular season games between the two connected teams. The black nodes and edges constitute the cliques provided by LDTPS. Our method finds 12 clusters that almost match the actual conferences. The five independent teams are placed in the conferences where they have played the most because they rarely play against each other. In addition, the Sun Belt conference is divided into two groups: one with a Western Athletic team, the other with a Western Athletic team and an independent team. This makes sense because the teams from both parts played only one game. Then, a Conference USA team is assigned to the Western Athletic teams

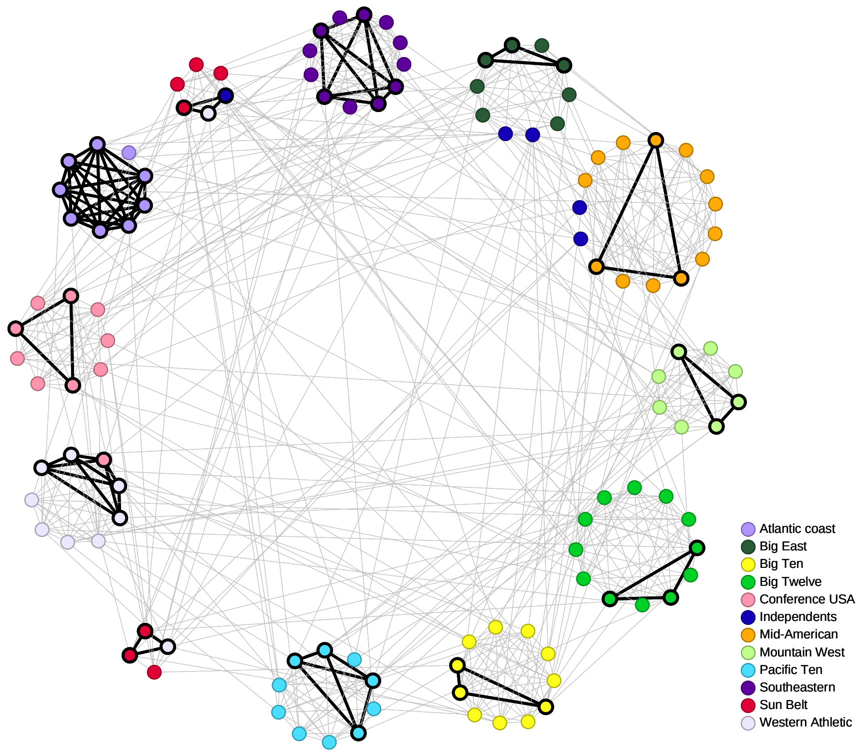


Figure 4: Application to the College football network.

because that team has played with every Western Athletic team but with no Conference USA team. In summary, our method perfectly identified the community structures established in regular-season-game associations. Furthermore, it identified the intra-conference association not reflected by the established community structures.

4. Analysis

In this section, we examine two key components of our LDTPS algorithm: the parametric constrained neighborhood and the probability learning mechanism, and then provide insights into the impact of the parameters.

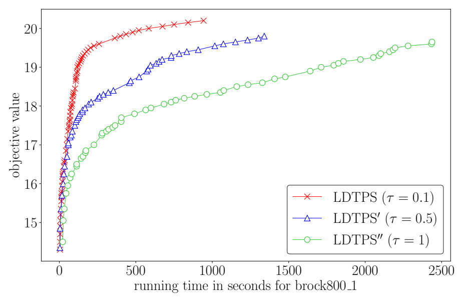
4.1. Effectiveness of the parametric constrained neighborhood

As mentioned in Section 2.5, LDTPS employs a parametric constrained neighborhood in its CSTS procedure. The constrained neighborhood is controlled by the parameter $0 \leq \tau \leq 1$, where a small (large) value of τ leads to a small (large) neighborhood ($\tau = 1$ corresponds to the complete neighborhood). To assess the effectiveness of the parametric constrained neighborhood, we study the running profiles of the algorithm with different values of τ based on 6 different types of hard graphs (brock800_1, gen400_p0.9_55, hamming10-2, johnson16-2-4, san400_0.9_1, sanr400_0.5). We compare LDTPS ($\tau = 0.1$) with two LDTPS variants: LDTPS$'$ ($\tau = 0.5$) and LDTPS$''$ ($\tau = 1$) under the same experimental protocol as described in Section 3.3. The running profiles for the 6 graphs are shown in Fig. 5. We observe that thanks to the parametric constrained neighborhood, LDTPS has a better convergence throughout the search. This experiment confirms the relevance of the parametric constrained neighborhood for the tabu search procedure.

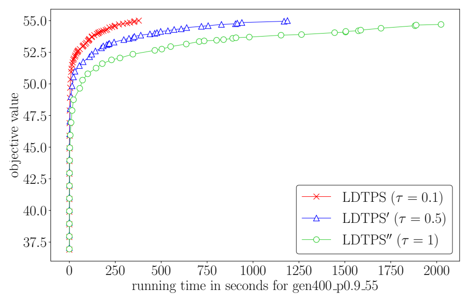
4.2. Effectiveness of the probability learning mechanism

We evaluate the impact of the probability learning mechanism used in LDTPS, by comparing LDTPS and its variant TPS, where the probability learning mechanism is removed while keeping only the local search procedure. To ensure a fair comparison, TPS was performed in a multi-start way, until the cutoff time (60 minutes) was reached. The experiment was conducted on Test Set I (45 graphs) and used the same experimental protocol as described in Section 3.3. Table 4 reports the same information as in Section 3.4.

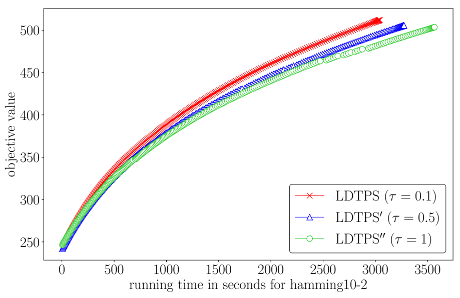
From Table 4, we observe that in terms of sol_{best} (sol_{avg}), LDTPS performs better on 3 (10) instances, equally well on 42 (35) instances. The average



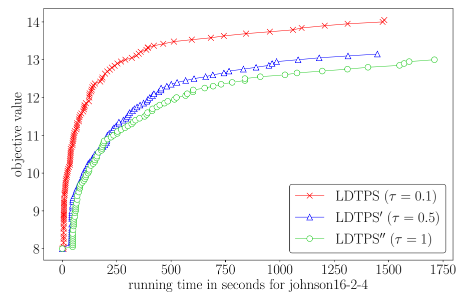
(a) brock800_1



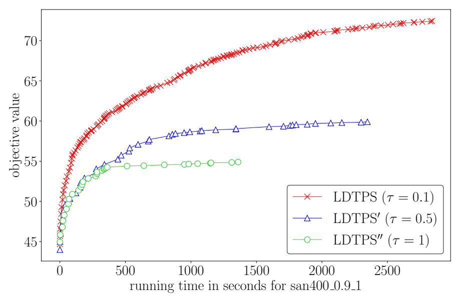
(b) gen400_p0.9_55



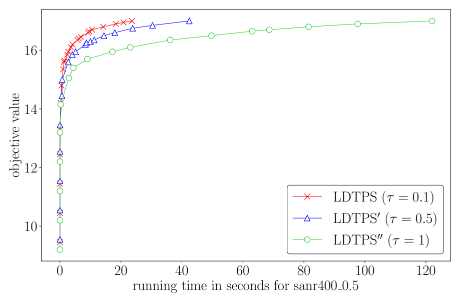
(c) hamming10-2



(d) johnson16-2-4



(e) san400_0.9_1



(f) sanr400_0.5

Figure 5: Running profiles of LDTPS ($\tau = 0.1$), LDTPS′ ($\tau = 0.5$), and LDTPS″ ($\tau = 1$) on 6 different types of hard graphs.

Table 4: Comparison results on Test Set I (45 graphs) between the LDTPS algorithm (with the probability learning mechanism) and its variant TPS algorithm (without the probability learning mechanism).

Graph	$\|V\|$	$\alpha^{\omega}(G)$	TPS				**LDTPS**			
			sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)
brock200_1	200	21*	**21**	21.0	20	24.5	**21**	21.0	20	31.5
brock200_2	200	15*	**15**	15.0	20	18.7	**15**	15.0	20	3.0
brock200_3	200	15*	14	13.8	15	1242.1	14	14.0	20	199.4
brock200_4	200	17*	16	16.0	20	16.2	**17**	16.1	1	73.5
brock400_1	400	23	$\underline{\mathbf{25}}$	25.0	20	93.0	$\underline{\mathbf{25}}$	25.0	20	150.8
brock400_2	400	23	$\underline{\mathbf{25}}$	25.0	20	112.3	$\underline{\mathbf{25}}$	25.0	20	35.4
brock400_3	400	23	$\underline{\mathbf{25}}$	25.0	20	144.2	$\underline{\mathbf{25}}$	25.0	20	35.8
c-fat200-1	200	130*	**130**	130.0	20	6.8	**130**	130.0	20	8.3
c-fat200-5	200	115*	**115**	115.0	20	4.8	**115**	115.0	20	59.4
c-fat500-1	500	332*	**332**	332.0	20	555.7	**332**	332.0	20	287.8
c-fat500-2	500	326*	**326**	326.0	20	447.3	**326**	326.0	20	521.5
c-fat500-10	500	313*	**313**	313.0	20	264.4	**313**	313.0	20	754.4
DSJC500.5	500	17	**17**	17.0	20	1024.8	**17**	17.0	20	158.3
gen200_p0.9_44	200	44	**44**	44.0	20	108.0	**44**	44.0	20	59.1
gen200_p0.9_55	200	55	**55**	55.0	20	38.6	**55**	55.0	20	11.4
gen400_p0.9_65	400	42	$\underline{\mathbf{65}}$	65.0	20	357.2	$\underline{\mathbf{65}}$	65.0	20	109.9
gen400_p0.9_75	400	39	$\underline{\mathbf{75}}$	73.4	3	2635.5	$\underline{\mathbf{75}}$	75.0	20	101.2
hamming6-2	64	32*	**32**	32.0	20	<0.1	**32**	32.0	20	<0.1
hamming6-4	64	16*	**16**	16.0	20	<0.1	**16**	16.0	20	<0.1
hamming8-2	256	128*	**128**	128.0	20	8.5	**128**	128.0	20	8.4
hamming8-4	256	16*	**16**	16.0	20	0.4	**16**	16.0	20	0.3
johnson8-2-4	28	7*	**7**	7.0	20	<0.1	**7**	7.0	20	0.1
johnson8-4-4	70	14*	**14**	14.0	20	0.1	**14**	14.0	20	<0.1
johnson32-2-4	496	31	16	16.0	20	<0.1	16	16.0	20	<0.1
keller4	171	15*	**15**	15.0	20	107.0	**15**	15.0	20	93.4
MANN_a9	45	16*	**16**	16.0	20	<0.1	**16**	16.0	20	<0.1
MANN_a27	378	126	**126**	125.6	12	987.1	**126**	125.7	14	1175.8
p_hat300-1	300	46	$\underline{\mathbf{48}}$	48.0	20	4.9	$\underline{\mathbf{48}}$	48.0	20	1.6
p_hat300-2	300	32	$\underline{\mathbf{33}}$	33.0	20	1.2	$\underline{\mathbf{33}}$	33.0	20	0.6
p_hat300-3	300	31	$\underline{\mathbf{36}}$	36.0	20	9.0	$\underline{\mathbf{36}}$	36.0	20	3.0
p_hat500-1	500	34	$\underline{\mathbf{62}}$	61.4	8	1206.6	$\underline{\mathbf{62}}$	62.0	20	28.9
p_hat500-2	500	35	$\underline{\mathbf{45}}$	45.0	20	13.7	$\underline{\mathbf{45}}$	45.0	20	3.8
p_hat500-3	500	36	$\underline{\mathbf{50}}$	50.0	20	125.1	$\underline{\mathbf{50}}$	50.0	20	16.5
p_hat700-1	700	33	$\underline{\mathbf{82}}$	80.9	1	1378.3	$\underline{\mathbf{82}}$	82.0	20	730.4
p_hat700-2	700	37	$\underline{\mathbf{60}}$	59.3	6	967.1	$\underline{\mathbf{60}}$	60.0	20	72.1
p_hat700-3	700	35	$\underline{\mathbf{62}}$	62.0	20	938.8	$\underline{\mathbf{62}}$	62.0	20	255.0
san200_0.7_2	200	17	17	17.0	20	3.9	$\underline{\mathbf{18}}$	17.2	3	366.3
san200_0.9_1	200	70*	57	55.8	6	2375.1	**70**	70.0	20	1675.3
san200_0.9_2	200	60	**60**	59.9	18	1942.6	**60**	60.0	20	62.7
san200_0.9_3	200	44	**44**	44.0	20	165.3	**44**	44.0	20	34.8
san400_0.7_1	400	22	$\underline{\mathbf{25}}$	25.0	20	155.0	$\underline{\mathbf{25}}$	25.0	20	626.3
sanr200_0.7	200	18*	**18**	18.0	20	4.3	**18**	18.0	20	2.3
sanr200_0.9	200	42	**42**	42.0	20	23.9	**42**	42.0	20	6.6
sanr400_0.5	400	17*	**17**	17.0	20	131.2	**17**	17.0	20	23.6
sanr400_0.7	400	21	**21**	21.0	20	50.6	**21**	21.0	20	8.8
Average		57.4	61.7	61.6	18.0	393.2	**62.1**	**62.0**	**19.1**	**173.3**

Notes. The $*$ symbol indicates an optimal value. An underlined value indicates an improved best result, i.e., a new lower bound. A bold value corresponds to the best value among the solutions found by the compared algorithms.

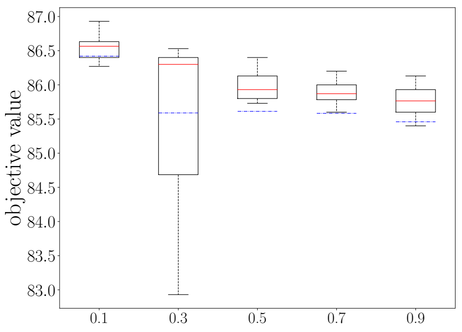
result in terms of sol_{best} (sol_{avg}) for LDTPS is 62.1 (62.0), which is marginally higher ($(62.1 - 61.7)/62.1 \times 100\% = 0.64\%$ and $(62.0 - 61.6)/62.0 \times 100\% = 0.65\%$) than 61.7 (61.6) of TPS. Furthermore, LDTPS generally requires a shorter average computing time 173.3 seconds *vs.* 393.2 seconds for TPS) to reach its final solution in one run, and achieves a much higher average successful rate ($19.1/20 \times 100\% = 95.50\%$ vs. $18.0/20 \times 100\% = 90.00\%$ for TPS). The small p-value = 5.01e-03 confirms a statistically average performance difference between TPS and LDTPS. But the differences for the best performance are not statistically significant (p-value = 1.02e-01). This experiment confirms our observation that the probability learning mechanism positively contributes to the high performance of LDTPS.

4.3. Analysis of the parameters

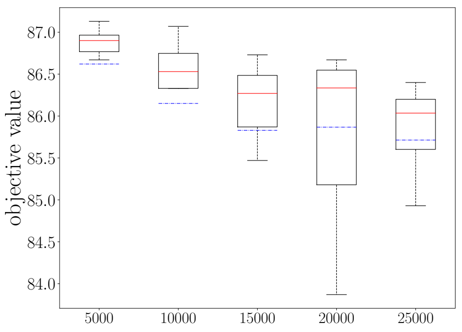
The LDTPS algorithm requires six parameters: τ, td, tt, α, β, and γ. Specifically, τ is the parametric constrained neighborhood coefficient, td and tt are the two parameters related to tabu search, where td is the maximum non-improving iterations of tabu search and tt is the tabu tenure, respectively. And α, β, and γ are the three parameters related to a probability learning mechanism, where α is the reward factor for the correct subset, β is the penalization factor for the incorrect subset, and γ is the compensation factor for the expected subset, respectively.

To analyze the effect of these parameters on the performance of LDTPS and the sensitivity of each parameter, a one-at-a-time sensitivity analysis [13] was performed as follows. We tested for each parameter the calibrated values, while fixing the other parameters to their default values from Table 2. The experiment relied on the same 15 graphs from the Training set and experimental protocol as before. Fig. 6 shows the box and whisker plots of the results, where the X-axis and the Y-axis indicate the parameter values and the accumulated best objective values (the IUC numbers) over the 15 graphs, respectively. The Friedman rank sum test with a confidence level of 95% is applied to assess whether there exists a statistically significant difference in solution samples for different values of a given parameter.

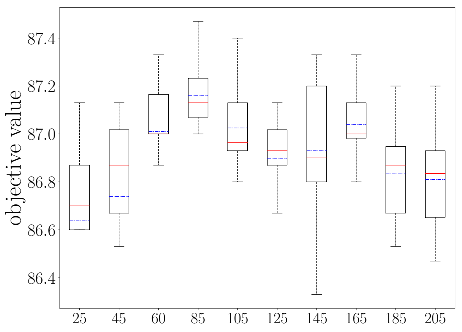
From Fig. 6, we observe that the performance of LDTPS varies according to the tested values of these parameters. The Friedman rand sum test indicates a significant difference in performance for the parameters τ (p-value=6.13e-06), td (p-value=3.71e-05), tt (p-value=1.26e-08), and γ (p-value=1.30e-02), while it does not exhibit any sensitivity for the parameters α and β (p-values > 0.05).



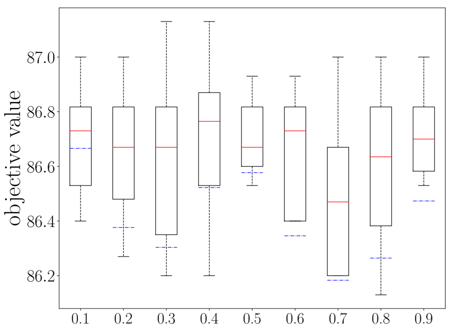
(a) τ (p-value=6.13e-06)



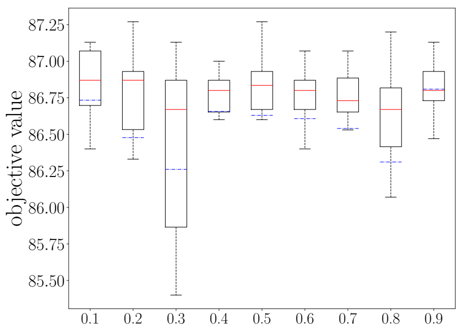
(b) td (p-value=3.71e-05)



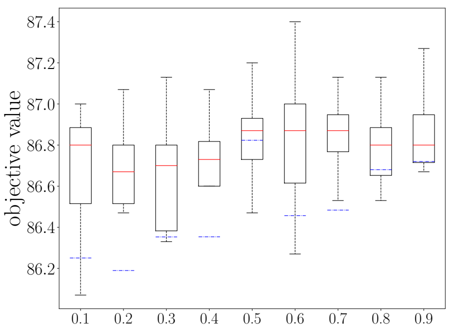
(c) tt (p-value=1.26e-08)



(d) α (p-value=1.97e-01)



(e) β (p-value=1.41e-01)



(f) γ (p-value=1.30e-02)

Figure 6: Analysis of the parameters τ, td, tt, α, β, and γ (by using the box and whisker plot) on the performance of the proposed LDTPS algorithm.

5. Conclusions and future work

In this work, we proposed the first learning driven three-phase search (LDTPS) algorithm for the NP-hard maximum independent union of cliques (IUC) problem. Specifically, we developed a parametric constrained swap-based tabu search that enables an effective examination of promising candidate solutions. We devised a frequency-based perturbation mechanism to enhance search diversification and explore new promising areas. Notably, we introduced a hybrid subset selection strategy and a probability updating rule derived from reinforcement learning, which collects useful information during the search to guide new solution construction.

We carried out extensive experiments to evaluate the proposed algorithm on the 83 challenging 2nd DIMACS graphs in the literature, and made comparisons with the state-of-the-art algorithms. The computational results indicated that our algorithm outperforms the reference algorithms on the benchmarks. Specifically, it reports 35 and 36 new lower bounds for the maximum IUC problem and the maximum MPC problem, respectively. We then presented an application of the proposed LDTPS algorithm to the analysis of three real-life social networks. Since we will make the code of our algorithm publicly available, we can expect other practical social network problems to benefit from this work. We also performed additional experiments to assess the effectiveness of the important ingredients of the algorithm.

The proposed algorithm follows basically the single trajectory local search framework. One possible future work would be to investigate the population-based hybrid approaches such as memetic algorithms where the LDTPS algorithm or its variants can beneficially be used as the local optimization component. Furthermore, the current work focuses on the design of an effective algorithm for the general IUC problem. It would be interesting to investigate its usefulness on practical applications. This is facilitated by the source code that we will make publicly available.

CRediT author statement

Zhi Lu: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft. **Jian Gao:** Conceptualization, Methodology, Investigation, Writing - Original Draft. **Jin-Kao Hao:** Supervising, Conceptualization, Methodology, Investigation, Writing - Original Draft. **Pingle Yang:** Conceptualization, Methodology, Investigation, Writing - Original

Draft. **Lixin Zhou:** Conceptualization, Methodology, Investigation, Writing - Original Draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors are grateful to the reviewers for their useful comments and suggestions, which helped us to improve the paper significantly. This research was partially supported by the National Natural Science Foundation of China (Grant no. 72101149 and 61972063); Shanghai Pujiang Program (Grant no. 22PJC080); the National Social Science Foundation of China (Grant no. 21BGL217 and 22CGL050).

References

[1] B. Balasundaram, S. Butenko, Graph domination, coloring and cliques in telecommunications, in: M. G. C. Resende, P. M. Pardalos (eds.), Handbook of Optimization in Telecommunications, Springer, 2006, pp. 865–890.

[2] U. Benlic, M. G. Epitropakis, E. K. Burke, A hybrid breakout local search and reinforcement learning approach to the vertex separator problem, European Journal of Operational Research 261 (3) (2017) 803–818.

[3] T. Benoist, B. Estellon, F. Gardi, R. Megel, K. Nouioua, Localsolver 1. x: a black-box local-search solver for 0-1 programming, 4OR 9 (3) (2011) 299–316.

[4] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-Race and iterated F-Race: An overview, Experimental Methods for the Analysis of Optimization Algorithms (2010) 311–336.

[5] V. Boginski, S. Butenko, P. M. Pardalos, Mining market data: A network approach, Computers & Operations Research 33 (11) (2006) 3171–3184.

[6] Z. Ertem, E. Lykhovyd, Y. Wang, S. Butenko, The maximum independent union of cliques problem: complexity and exact approaches, Journal of Global Optimization 76 (3) (2020) 545–562.

[7] Z. Ertem, A. Veremyev, S. Butenko, Detecting large cohesive subgroups with high clustering coefficients in social networks, Social Networks 46 (2016) 1–10.

[8] F. V. Fomin, S. Gaspers, D. Kratsch, M. Liedloff, S. Saurabh, Iterative compression and exact algorithms, Theoretical Computer Science 411 (7-9) (2010) 1045–1053.

[9] C. Friden, A. Hertz, D. de Werra, Stabulus: A technique for finding stable sets in large graphs with tabu search, Computing 42 (1) (1989) 35–44.

[10] Z.-H. Fu, J.-K. Hao, A three-phase search approach for the quadratic minimum spanning tree problem, Engineering Applications of Artificial Intelligence 46 (2015) 113–130.

[11] M. Girvan, M. E. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences 99 (12) (2002) 7821–7826.

[12] F. Glover, M. Laguna, Tabu search, in: D.-Z. Du, P. M. Pardalos (eds.), Handbook of Combinatorial Optimization: Volume 1–3, Springer US, 1998, pp. 2093–2229.

[13] D. Hamby, A review of techniques for parameter sensitivity analysis of environmental models, Environmental Monitoring and Assessment 32 (2) (1994) 135–154.

[14] P. He, J.-K. Hao, Iterated two-phase local search for the colored traveling salesmen problem, Engineering Applications of Artificial Intelligence 97 (2021) 104018.

[15] S. Hosseinian, S. Butenko, Polyhedral properties of the induced cluster subgraphs, Discrete Applied Mathematics 297 (2021) 80–96.

[16] F. Hüffner, C. Komusiewicz, H. Moser, R. Niedermeier, Fixed-parameter algorithms for cluster vertex deletion, Theory of Computing Systems 47 (1) (2010) 196–217.

[17] D. J. Johnson, M. A. Trick, Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993, American Mathematical Society, USA, 1996.

[18] V. Krebs, Uncloaking terrorist networks, First Monday.
URL `https://firstmonday.org/ojs/index.php/fm/article/view/941/863`

[19] C.-M. Li, Z. Fang, H. Jiang, K. Xu, Incremental upper bound for the maximum clique problem, INFORMS Journal on Computing 30 (1) (2018) 137–153.

[20] C.-M. Li, H. Jiang, F. Manyà, On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem, Computers & Operations Research 84 (2017) 1–15.

[21] LocalSolver, Localsolver 10.5 documentation (2022).
URL `https://www.localsolver.com/docs/last/index.html`

[22] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[23] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, in: F. W. Glover, G. A. Kochenberger (eds.), Handbook of Metaheuristics, vol. 57 of International Series in Operations Research & Management Science, Kluwer / Springer, 2003, pp. 320–353.

[24] Z. Lu, J.-K. Hao, Q. Wu, A hybrid evolutionary algorithm for finding low conductance of large graphs, Future Generation Computer Systems 106 (2020) 105–120.

[25] Z. Lu, J.-K. Hao, Y. Zhou, Stagnation-aware breakout tabu search for the minimum conductance graph partitioning problem, Computers & Operations Research 111 (2019) 43–57.

[26] N. Malod-Dognin, R. Andonov, N. Yanev, Maximum cliques in protein structure comparison, in: P. Festa (ed.), Experimental Algorithms, 9th International Symposium, SEA 2010, Proceedings, vol. 6049 of Lecture Notes in Computer Science, Springer, 2010, pp. 106–117.

[27] J. Pattillo, N. Youssef, S. Butenko, Clique relaxation models in social network analysis, in: M. T. Thai, P. M. Pardalos (eds.), Handbook

of Optimization in Complex Networks: Communication and Social Networks, Springer New York, 2012, pp. 143–162.

[28] Z. Sun, U. Benlic, M. Li, Q. Wu, Reinforcement learning based tabu search for the minimum load coloring problem, Computers & Operations Research 143 (2022) 105745.

[29] G. Valiente, Clique, independent set, and vertex cover, in: G. Valiente (ed.), Algorithms on Trees and Graphs, Springer Berlin Heidelberg, 2002, pp. 299–350.

[30] R. Van Bevern, H. Moser, R. Niedermeier, Approximation and tidying-A problem kernel for s-plex cluster vertex deletion, Algorithmica 62 (3) (2012) 930–950.

[31] G. Verfaillie, M. Lemaître, T. Schiex, Russian doll search for solving constraint optimization problems, in: W. J. Clancey, D. S. Weld (eds.), Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, Volume 1, AAAI Press / The MIT Press, 1996, pp. 181–187.

[32] Y. Wang, S. Pan, C. Li, M. Yin, A local search algorithm with reinforcement learning based repair procedure for minimum weight independent dominating set, Information Sciences 512 (2020) 533–548.

[33] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (6684) (1998) 440–442.

[34] Q. Wu, J.-K. Hao, An adaptive multistart tabu search approach to solve the maximum clique problem, Journal of Combinatorial Optimization 26 (1) (2013) 86–108.

[35] Q. Wu, J.-K. Hao, A review on algorithms for maximum clique problems, European Journal of Operational Research 242 (3) (2015) 693–709.

[36] W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33 (4) (1977) 452–473.

[37] Q. Zhou, U. Benlic, Q. Wu, An opposition-based memetic algorithm for the maximum quasi-clique problem, European Journal of Operational Research 286 (1) (2020) 63–83.

[38] Y. Zhou, J.-K. Hao, B. Duval, Reinforcement learning based local search for grouping problems: A case study on graph coloring, Expert Systems with Applications 64 (2016) 412–422.

[39] Y. Zhou, J.-K. Hao, A. Goëffon, A three-phased local search approach for the clique partitioning problem, Journal of Combinatorial Optimization 32 (2) (2016) 469–491.

Appendix A. Implemented RSA and GA algorithms

The restart simulated annealing (RSA) algorithm and the genetic algorithm (GA) follow the general simulated annealing and genetic algorithm frameworks. They integrate as many search components as possible from our LDTPS algorithm. They use the same general solution approach as LDTPS. That is, they seek a legal k-IUC for a fixed k, and each time a legal k-IUC is found, they continue the search process by setting $k \leftarrow k + 1$, until the given time limit (60 minutes as for LDTPS) is reached. At termination, the best IUC S^* and the IUC number k^* are returned. The codes of the RSA and GA algorithms are available at `https://github.com/hellozhilu/LDTPS`.

The RSA algorithm begins with an initial solution S generated by the method of Section 2.3, and performs its search in a multi-start way until the cutoff time is met. At each round of search, RSA starts with an initial temperature T (set to 1). It explores the constrained swap-based neighborhood CN_{swap} (see Section 2.5.1) and randomly swaps two vertices from subsets A and B to generate a neighboring solution S'. Then, if the acceptance probability $Pr\{S \leftarrow S'\} = \min(1, e^{\frac{f(S')-f(S)}{T}})$ is verified, the neighboring solution S' becomes the new current solution S; otherwise, it is ignored without changing the current solution during the current iteration. Once the number of evaluated solutions reaches a threshold (set to $16 \times |V|$), the temperature T is cooled down by a constant factor 0.96, and SA proceeds to the next search round with this lowered temperature. The frozen state of SA is reached when the acceptance rate, defined as $move/(16 \times |V|)$ ($move$ is the move counter), becomes smaller than 0.01 for 5 consecutive search rounds.

The GA algorithm starts with an initial population of solutions, where each solution is constructed using the method described in Section 2.3. The population of size 50 is then evolved through several generations using three search operators, including crossover, mutation, and pool updating. For each generation, two parent solutions are randomly selected from the population and combined by the uniform crossover. In the uniform crossover, each offspring bit is randomly selected from either parent with equal probability. If the offspring solution is infeasible, it is repaired by adding vertices to the solution or removing vertices from the solution. Finally, the new solution is inserted into the population, replacing the worst solution, if the offspring is better than the worst solution and different from all solutions in the population. The algorithm terminates when the cutoff time is reached.

Appendix B. Detailed Computational results

We present the computational results of the proposed LDTPS algorithm, the restart simulated annealing RSA, the genetic algorithm GA, the LocalSolver heuristic [21], the IP-based approach [6], and the RDS algorithm [6] for both the maximum IUC problem and the maximum MPC problem on the DIMACS2 benchmark graphs (including the 15 graphs of Training Set, the 45 graphs of Test Set I and the 23 large graphs of Test Set II).

The results of the IP-based approach and the RDS algorithm are missing on Test Set II because the CPLEX solver did not find feasible solutions on these large graphs, and the source code of the RDS algorithm is not available. The results of the LocalSolver heuristic, the IP-based approach, and the RDS algorithm were obtained from a single run, while the other algorithms (RSA, GA, LDTPS) were run 20 times per instance under the condition given in Section 3.3.

Table B.1: Computational results for the maximum IUC problem and the maximum MPC problem on Training Set (15 graphs) of our LDTPS algorithm.

Graph	$\lvert V\rvert$	IUC					MPC				
		$\alpha^{\omega}(G)$	sol$_{\text{best}}$	sol$_{\text{avg}}$	hit	t(s)	$\omega^{\alpha}(G)$	sol$_{\text{best}}$	sol$_{\text{avg}}$	hit	t(s)
brock400_4	400	33*	25	25.0	20	92.6	25	$\underline{\mathbf{31}}$	31.0	20	46.8
brock800_1	800	20	$\underline{\mathbf{21}}$	20.2	4	943.5	20	$\underline{\mathbf{26}}$	26.0	20	219.0
brock800_2	800	20	$\underline{\mathbf{21}}$	20.9	17	1601.1	21	$\underline{\mathbf{26}}$	26.0	20	620.5
brock800_3	800	20	$\underline{\mathbf{21}}$	20.6	11	1116.0	20	$\underline{\mathbf{26}}$	26.0	19	1104.0
c-fat200-2	200	134*	**134**	134.0	20	16.0	24*	11	10.5	10	1649.2
c-fat500-5	500	313*	**313**	313.0	20	265.0	64*	12	10.9	5	2343.7
gen400_p0.9_55	400	44	$\underline{\mathbf{55}}$	55.0	20	183.6	124	$\underline{\mathbf{126}}$	115.2	1	1090.1
hamming10-2	1024	512*	**512**	512.0	20	3041.7	512*	**512**	512.0	20	2109.6
hamming10-4	1024	18	$\underline{\mathbf{40}}$	40.0	20	99.5	32	$\underline{\mathbf{75}}$	74.3	5	790.6
johnson16-2-4	120	15*	**15**	14.1	5	1459.0	15*	**15**	15.0	20	<0.1
san200_0.7_1	200	30*	23	21.0	3	1772.9	105*	**105**	105.0	20	16.1
san400_0.5_1	400	29	22	20.0	1	2332.0	214*	**214**	208.4	13	178.3
san400_0.7_2	400	18	$\underline{\mathbf{24}}$	22.0	1	1386.0	205*	**205**	201.7	9	131.8
san400_0.7_3	400	16	$\underline{\mathbf{22}}$	20.0	1	1039.0	216*	**216**	205.6	13	155.4
san400_0.9_1	400	51	$\underline{\mathbf{75}}$	73.1	3	2804.5	200*	**200**	200.0	20	78.0
Average		84.9	88.2	87.4	11.1	1210.2	119.8	120.0	117.8	14.3	702.2

Notes. The $*$ symbol indicates an optimal value. An underlined value indicates an improved best result, i.e., a new lower bound. A bold value corresponds to the best-known value in the literature.

Table B.1 shows the results of LDTPS for both problems on Training Set (15 graphs), which were used for parameter calibration. Tables B.2 to B.5 show the detailed comparative results on Test Set I (45 graphs) and Test Set II (23 large graphs). Columns 1 and 2 indicate for each instance, the instance name (Graph) and the number of vertices ($|V|$). Column 3 ($\omega^{\alpha}(G)$ and $\alpha^{\omega}(G)$) shows the largest known IUC and MPC numbers ever reported in the literature, while an asterisk ($*$) indicates a proven maximum IUC and MPC number. For the single-run approaches IP, RDS, and LocalSolver, we report the IUC (MPC) number or the best lower bound (indicated by sol) and the run time in seconds (t(s)). For the multi-run algorithms RSA, GA, and LDTPS, we report the best IUC (MPC) number (sol_{best}), the average IUC (MPC) number over 20 independent runs (sol_{avg}), the number of times sol_{best} was reached across the 20 runs (hit), and the average run time in seconds required to reach the final solution in each run (t(s)). The '-' symbol indicates that the corresponding result is not available (e.g., no feasible solution was found, the time limit was reached, or the memory limit was exceeded). An underlined value indicates an improved best result (new lower bound). A bold value corresponds to the best-known value in the literature.

Table B.2: Comparison results for the maximum IUC problem on Test Set I (45 graphs) of our LDTPS algorithm, the restart simulated annealing RSA, the genetic algorithm GA, the LocalSolver heuristic [21], the IP-based approach [6], and the RDS algorithm [6].

Graph	$\|V\|$	$\alpha^\omega(G)$	IP [6]		RDS [6]		LocalSolver [21]		RSA				GA				**LDTPS**			
			sol	t(s)	sol	t(s)	sol	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)
brock200_1	200	21*	**21**	35978.6	**21**	201.5	18	707.0	**21**	21.0	20	7.5	18	14.5	1	258.6	**21**	21.0	20	31.5
brock200_2	200	15*	13	-	**15**	29.7	13	166.0	**15**	15.0	20	14.0	11	10.3	5	825.7	**15**	15.0	20	3.0
brock200_3	200	15*	13	-	**15**	4.2	13	139.0	14	14.0	20	84.7	11	9.5	1	520.5	14	14.0	20	199.4
brock200_4	200	17*	16	-	**17**	5.5	14	444.0	**17**	17.0	20	699.7	14	11.4	1	769.3	**17**	16.1	1	73.5
brock400_1	400	23	-	-	23	-	23	3359.0	**25**	25.0	20	201.8	19	16.4	1	<0.1	**25**	25.0	20	150.8
brock400_2	400	23	-	-	23	-	22	693.0	**26**	25.1	1	272.3	19	16.7	1	<0.1	25	25.0	20	35.4
brock400_3	400	23	-	-	23	-	21	3222.0	**27**	25.7	2	758.8	23	16.8	1	<0.1	25	25.0	20	35.8
c-fat200-1	200	130*	**130**	0.4	**130**	0.9	**130**	4.0	**130**	130.0	20	9.0	69	67.2	2	3288.0	**130**	130.0	20	8.3
c-fat200-5	200	115*	**115**	50.4	**115**	0.5	**115**	346.0	**115**	115.0	20	6.6	62	59.2	1	2242.3	**115**	115.0	20	59.4
c-fat500-1	500	332*	**332**	2.9	**332**	90.3	**332**	5.0	329	327.9	3	2167.9	73	70.6	2	2953.9	**332**	332.0	20	287.8
c-fat500-2	500	326*	**326**	43.8	**326**	83.7	**326**	656.0	**326**	324.8	1	1822.7	58	56.6	5	2887.1	**326**	326.0	20	521.5
c-fat500-10	500	313*	**313**	3712.5	**313**	67.4	297	2882.0	**313**	286.8	5	491.0	126	125.0	7	<0.1	**313**	313.0	20	754.4
DSJC500.5	500	17	-	-	**17**	-	-	-	**17**	16.1	1	473.3	10	9.9	19	1062.1	**17**	17.0	20	158.3
gen200_p0.9_44	200	44	**44**	-	34	-	39	747.0	**44**	44.0	20	9.2	33	29.4	1	<0.1	**44**	44.0	20	59.1
gen200_p0.9_55	200	55	**55**	-	42	-	**55**	201.0	**55**	55.0	20	6.9	34	30.4	2	<0.1	**55**	55.0	20	11.4
gen400_p0.9_65	400	42	42	-	33	-	47	3308.0	**65**	65.0	20	210.2	40	36.5	1	<0.1	**65**	65.0	20	109.9
gen400_p0.9_75	400	39	39	-	36	-	53	3479.0	**75**	75.0	20	624.1	42	38.9	2	<0.1	**75**	75.0	20	101.2
hamming6-2	64	32*	**32**	6.8	**32**	<0.1	**32**	3.0	**32**	32.0	20	<0.1	**32**	32.0	20	<0.1	**32**	32.0	20	<0.1
hamming6-4	64	16*	**16**	11.2	**16**	0.1	**16**	3.0	**16**	16.0	20	<0.1	**16**	16.0	20	84.4	**16**	16.0	20	<0.1
hamming8-2	256	128*	**128**	341.7	**128**	1.0	**128**	140.0	**128**	127.3	17	12.7	97	68.8	1	<0.1	**128**	128.0	20	8.4
hamming8-4	256	16*	**16**	16425.2	**16**	60.2	**16**	262.0	**16**	16.0	20	0.2	11	10.1	4	1104.4	**16**	16.0	20	0.3
johnson8-2-4	28	7*	**7**	0.1	**7**	<0.1	**7**	1.0	**7**	7.0	20	<0.1	**7**	7.0	20	0.1	**7**	7.0	20	0.1
johnson8-4-4	70	14*	**14**	2.7	**14**	<0.1	**14**	4.0	**14**	14.0	20	<0.1	**14**	14.0	20	<0.1	**14**	14.0	20	<0.1
johnson32-2-4	496	31	-	-	**31**	-	-	-	16	16.0	20	<0.1	16	16.0	20	<0.1	16	16.0	20	<0.1
keller4	171	15*	**15**	9539.7	**15**	3.0	11	56.0	**15**	15.0	20	0.4	11	10.6	12	1099.7	**15**	15.0	20	93.4
MANN_a9	45	16*	**16**	1.2	**16**	<0.1	**16**	2.0	**16**	16.0	20	<0.1	**16**	15.6	12	0.4	**16**	16.0	20	<0.1
MANN_a27	378	126	**126**	-	119	-	125	1312.0	**126**	125.9	18	1372.2	121	119.3	4	<0.1	**126**	125.7	14	1175.8
p_hat300-1	300	46	46	-	33	-	**48**	573.0	**48**	47.9	18	937.8	21	20.2	5	1723.1	**48**	48.0	20	1.6

Table B.2: Continued

Graph	$\|V\|$	$\alpha^{\omega}(G)$	IP [6]		RDS [6]		LocalSolver [21]		RSA				GA				**LDTPS**			
			sol	t(s)	sol	t(s)	sol	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)
p_hat300-2	300	32	28	-	32	-	**33**	1517.0	**33**	33.0	20	9.2	18	15.2	1	852.7	**33**	33.0	20	0.6
p_hat300-3	300	31	29	-	31	-	34	1563.0	**36**	36.0	20	5.9	28	22.1	1	<0.1	**36**	36.0	20	3.0
p_hat500-1	500	34	32	-	34	-	**62**	2189.0	60	59.8	15	1322.8	19	18.7	14	1905.4	**62**	62.0	20	28.9
p_hat500-2	500	35	-	-	35	-	-	-	**45**	44.5	9	888.8	23	17.3	1	432.1	**45**	45.0	20	3.8
p_hat500-3	500	36	-	-	36	-	-	-	**50**	50.0	20	482.4	33	27.8	2	<0.1	**50**	50.0	20	16.5
p_hat700-1	700	33	-	-	33	-	-	-	79	77.7	1	1479.4	19	18.6	12	1733.6	**82**	82.0	20	730.4
p_hat700-2	700	37	-	-	37	-	-	-	58	57.5	9	1319.7	29	20.8	1	2.5	**60**	60.0	20	72.1
p_hat700-3	700	35	-	-	35	-	-	-	**62**	61.9	17	1534.1	41	33.4	1	<0.1	**62**	62.0	20	255.0
san200_0.7_2	200	17	17	-	17	-	16	115.0	17	17.0	20	10.2	14	13.2	4	625.3	**18**	17.2	3	366.3
san200_0.9_1	200	70*	**70**	21497.1	37	-	**70**	1657.0	65	57.4	1	689.7	48	41.3	2	229.5	**70**	70.0	20	1675.3
san200_0.9_2	200	60	**60**	-	36	-	**60**	281.0	**60**	59.6	15	1900.8	38	32.2	3	91.4	**60**	60.0	20	62.7
san200_0.9_3	200	44	**44**	-	33	-	**44**	265.0	**44**	44.0	20	18.3	31	27.4	1	<0.1	**44**	44.0	20	34.8
san400_0.7_1	400	22	20	-	22	-	-	-	**25**	24.5	9	944.6	21	19.6	5	192.5	**25**	25.0	20	626.3
sanr200_0.7	200	18*	17	-	**18**	21.6	17	359.0	**18**	18.0	20	0.4	14	12.2	1	179.9	**18**	18.0	20	2.3
sanr200_0.9	200	42	**42**	-	35	-	**42**	213.0	**42**	42.0	20	5.9	34	29.0	1	<0.1	**42**	42.0	20	6.6
sanr400_0.5	400	17*	10	-	**17**	12622.9	-	-	**17**	16.9	19	874.1	11	9.9	3	990.9	**17**	17.0	20	23.6
sanr400_0.7	400	21	-	-	**21**	-	-	-	**21**	21.0	20	12.5	18	13.6	1	<0.1	**21**	21.0	20	8.8

Notes. The $*$ symbol indicates an optimal value. An underlined value indicates an improved best result, i.e., a new lower bound. A bold value corresponds to the best-known value in the literature.

Table B.3: Comparison results for the maximum MPC problem on Test Set I (45 graphs) of our LDTPS algorithm, the restart simulated annealing RSA, the genetic algorithm GA, the LocalSolver heuristic [21], the IP-based approach [6], and the RDS algorithm [6].

Graph	$\|V\|$	$\omega^\alpha(G)$	IP [6]		RDS [6]		LocalSolver [21]		RSA				GA				**LDTPS**			
			sol	t(s)	sol	t(s)	sol	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)
brock200_1	200	24	23	-	24	-	25	79.0	**28**	28.0	20	31.8	17	15.7	2	994.1	**28**	28.0	20	1.2
brock200_2	200	14*	12	-	**14**	37.3	13	197.0	**14**	14.0	20	4.1	10	9.9	18	458.9	**14**	14.0	20	0.9
brock200_3	200	18*	**18**	-	**18**	775.6	17	1058.0	**18**	18.0	20	21.6	13	11.8	1	608.5	**18**	18.0	20	3.4
brock200_4	200	20*	17	-	**20**	5652.8	19	3461.0	**20**	20.0	20	63.5	14	13.0	2	851.4	**20**	20.0	20	4.7
brock400_1	400	25	23	-	25	-	28	3489.0	30	29.3	6	1116.3	16	15.3	7	1131.3	**31**	31.0	20	31.8
brock400_2	400	27	24	-	27	-	27	1371.0	30	29.3	6	1151.8	16	15.1	2	743.8	**31**	31.0	20	28.1
brock400_3	400	26	24	-	26	-	28	1704.0	30	29.3	5	1012.8	16	15.2	5	1193.5	**31**	31.0	20	23.6
c-fat200-1	200	18	**18**	-	**18**	-	**18**	12.0	**18**	18.0	20	<0.1	**18**	17.4	7	902.2	**18**	18.0	20	<0.1
c-fat200-5	200	58*	**58**	1036.2	**58**	<0.1	**58**	351.0	**58**	58.0	20	2.1	22	20.0	2	1817.8	**58**	58.0	20	6.5
c-fat500-1	500	40	37	-	**40**	-	**40**	786.0	**40**	40.0	20	11.9	36	34.6	6	<0.1	**40**	40.0	20	25.2
c-fat500-2	500	20	-	-	**20**	-	**20**	83.0	**20**	20.0	20	0.4	19	17.9	1	634.3	**20**	20.0	20	0.4
c-fat500-10	500	126*	-	-	**126**	0.9	-	-	**126**	126.0	20	51.8	16	15.3	9	1368.2	**126**	126.0	20	229.8
DSJC500.5	500	17	-	-	**17**	-	-	-	**17**	16.2	3	690.4	10	9.9	17	432.2	**17**	17.0	20	72.1
gen200_p0.9_44	200	72	**72**	-	37	-	**72**	14.0	**72**	72.0	20	3.3	32	31.1	6	2506.2	**72**	72.0	20	1.2
gen200_p0.9_55	200	56	56	-	37	-	**58**	26.0	**58**	58.0	20	90.2	31	29.3	1	1335.7	**58**	58.0	20	1.6
gen400_p0.9_65	400	138	**138**	-	40	-	**138**	256.0	**138**	137.8	15	1088.1	29	27.2	1	1799.6	**138**	138.0	20	24.7
gen400_p0.9_75	400	136	**136**	-	37	-	**136**	183.0	**136**	135.7	13	1348.7	29	26.9	1	1256.4	**136**	136.0	20	15.8
hamming6-2	64	32*	**32**	1.2	**32**	<0.1	**32**	2.0	**32**	32.0	20	0.1	**32**	30.4	6	534.7	**32**	32.0	20	<0.1
hamming6-4	64	14*	**14**	19.2	**14**	<0.1	**14**	3.0	**14**	14.0	20	<0.1	**14**	14.0	20	2.6	**14**	14.0	20	<0.1
hamming8-2	256	128*	**128**	3198.4	**128**	2.3	**128**	3.0	**128**	128.0	20	364.5	50	48.3	3	2129.2	**128**	128.0	20	6.7
hamming8-4	256	32	**32**	-	**32**	-	28	252.0	**32**	32.0	20	77.6	16	14.9	10	626.8	**32**	32.0	20	17.4
johnson8-2-4	28	7*	**7**	0.2	**7**	<0.1	**7**	1.0	**7**	7.0	20	<0.1	**7**	7.0	20	<0.1	**7**	7.0	20	<0.1
johnson8-4-4	70	16*	**16**	16.9	**16**	0.7	**16**	2.0	**16**	16.0	20	<0.1	**16**	15.2	7	147.9	**16**	16.0	20	0.2
johnson32-2-4	496	31	**31**	-	**31**	-	**31**	164.0	**31**	31.0	20	<0.1	**31**	30.6	19	140.2	**31**	31.0	20	<0.1
keller4	171	26*	**26**	-	**26**	65.9	24	1072.0	**26**	26.0	20	2.4	16	15.0	3	909.3	**26**	26.0	20	1.6
MANN_a9	45	36*	**36**	<0.1	**36**	<0.1	**36**	<0.1	**36**	36.0	20	<0.1	**36**	36.0	20	0.7	**36**	36.0	20	<0.1
MANN_a27	378	351*	**351**	<0.1	**351**	-	**351**	<0.1	**351**	351.0	20	140.6	242	239.1	1	3498.9	**351**	351.0	20	80.3
p_hat300-1	300	37	36	-	37	-	**39**	2599.0	**39**	39.0	20	2.3	27	23.7	2	<0.1	**39**	39.0	20	0.6

Table B.3: Continued

Graph	$\|V\|$	$\omega^\alpha(G)$	IP [6]		RDS [6]		LocalSolver [21]		RSA				GA				**LDTPS**			
			sol	t(s)	sol	t(s)	sol	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)
p_hat300-2	300	31*	28	-	**31**	23452.4	**31**	1386.0	**31**	31.0	20	52.0	18	14.8	1	294.5	**31**	31.0	20	1.5
p_hat300-3	300	43	43	-	36	-	$\underline{\textbf{44}}$	625.0	$\underline{\textbf{44}}$	44.0	20	449.2	19	18.6	13	1672.7	$\underline{\textbf{44}}$	44.0	20	1.7
p_hat500-1	500	36	-	-	36	-	-	-	$\underline{\textbf{50}}$	50.0	20	73.2	33	27.5	1	<0.1	$\underline{\textbf{50}}$	50.0	20	7.5
p_hat500-2	500	36	-	-	36	-	-	-	42	41.8	15	1407.4	25	17.6	1	218.9	$\underline{\textbf{43}}$	43.0	20	11.8
p_hat500-3	500	42	42	-	36	-	62	2747.0	61	60.7	14	1244.2	20	18.8	4	1734.5	$\underline{\textbf{63}}$	63.0	20	264.1
p_hat700-1	700	39	-	-	39	-	-	-	$\underline{\textbf{65}}$	64.7	13	1637.7	40	31.7	1	<0.1	$\underline{\textbf{65}}$	65.0	20	368.7
p_hat700-2	700	35	-	-	35	-	-	-	51	50.2	3	1166.6	31	24.3	2	<0.1	$\underline{\textbf{53}}$	53.0	20	47.7
p_hat700-3	700	34	-	-	34	-	-	-	73	71.8	3	1449.0	20	18.2	1	1171.2	$\underline{\textbf{76}}$	76.0	20	372.1
san200_0.7_2	200	134*	**134**	3.8	**134**	1.1	**134**	78.0	**134**	134.0	20	5.7	85	81.3	1	3391.1	**134**	134.0	20	11.8
san200_0.9_1	200	125*	**125**	10.8	**125**	7.3	**125**	18.0	**125**	125.0	20	4.5	67	63.8	2	3344.5	**125**	125.0	20	6.0
san200_0.9_2	200	105*	**105**	332.1	**105**	8155.6	**105**	14.0	**105**	105.0	20	4.0	43	41.0	5	2863.9	**105**	105.0	20	3.3
san200_0.9_3	200	100*	**100**	697.1	**100**	848.6	**100**	22.0	**100**	100.0	20	3.4	40	37.1	1	2980.0	**100**	100.0	20	3.4
san400_0.7_1	400	200*	**200**	-	**200**	7.4	188	3594.0	**200**	200.0	20	53.6	34	30.8	1	1905.4	**200**	200.0	20	135.3
sanr200_0.7	200	22	21	-	**22**	-	21	177.0	**22**	22.0	20	2.4	14	13.9	17	587.9	**22**	22.0	20	0.3
sanr200_0.9	200	49	49	-	36	-	50	19.0	$\underline{\textbf{51}}$	50.9	19	1076.1	29	27.4	1	1562.8	$\underline{\textbf{51}}$	51.0	20	5.4
sanr400_0.5	400	16*	11	-	**16**	17000.4	15	2178.0	**16**	16.0	20	122.8	11	10.0	1	946.9	**16**	16.0	20	4.7
sanr400_0.7	400	22	19	-	22	-	23	652.0	$\underline{\textbf{27}}$	25.6	1	1324.8	14	13.8	15	1151.6	$\underline{\textbf{27}}$	27.0	20	21.0

Notes. The $*$ symbol indicates an optimal value. An underlined value indicates an improved best result, i.e., a new lower bound. A bold value corresponds to the best-known value in the literature.

Table B.4: Comparison results for the maximum IUC problem on Test Set II (23 large graphs) of our LDTPS algorithm, the restart simulated annealing RSA, the genetic algorithm GA, and the LocalSolver heuristic [21].

Graph	$\lvert V\rvert$	$\alpha^\omega(G)$	LocalSolver [21]		RSA				GA				**LDTPS**			
			sol	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)	sol_{best}	sol_{avg}	hit	t(s)
brock800_4	800	-	-	-	$\underline{\textbf{21}}$	20.0	2	1329.6	16	14.0	1	<0.1	$\underline{\textbf{21}}$	20.7	13	1314.5
C1000.9	1000	-	-	-	61	59.8	2	2109.2	51	45.3	2	<0.1	$\underline{\textbf{67}}$	66.2	6	1764.7
C2000.5	2000	-	-	-	17	16.0	1	1000.9	13	10.7	2	31.9	$\underline{\textbf{20}}$	19.5	10	818.4
C2000.9	2000	-	-	-	61	59.3	1	1888.7	55	51.6	2	<0.1	$\underline{\textbf{73}}$	71.8	3	2738.7
C4000.5	4000	-	-	-	16	15.8	15	1357.5	13	11.6	2	<0.1	$\underline{\textbf{21}}$	20.7	13	1035.7
DSJC500.1	500	-	67	3605.0	64	63.6	11	1697.0	27	25.5	1	1690.8	$\underline{\textbf{69}}$	68.9	19	1100.9
DSJC500.9	500	-	49	3600.0	$\underline{\textbf{56}}$	54.6	1	1286.4	44	39.0	1	<0.1	55	53.6	1	11.8
DSJC1000.1	1000	-	70	3600.0	68	66.6	2	1919.8	26	24.8	4	1692.2	$\underline{\textbf{79}}$	78.8	15	1484.4
DSJC1000.5	1000	-	-	-	17	16.1	1	616.9	12	9.9	1	200.2	$\underline{\textbf{19}}$	18.5	9	490.9
DSJC1000.9	1000	-	-	-	60	59.5	10	1723.3	48	44.3	1	<0.1	$\underline{\textbf{67}}$	65.7	2	1625.6
flat1000_50_0	1000	-	-	-	17	16.2	4	1004.9	11	10.2	6	241.7	$\underline{\textbf{20}}$	19.2	8	1351.5
flat1000_60_0	1000	-	-	-	17	16.1	1	479.9	12	10.2	1	675.0	$\underline{\textbf{19}}$	18.6	11	762.9
flat1000_76_0	1000	-	-	-	17	16.2	3	618.2	11	10.0	2	695.5	$\underline{\textbf{19}}$	18.4	8	430.3
keller5	776	-	-	-	$\underline{\textbf{27}}$	26.9	17	1010.6	21	16.8	1	<0.1	$\underline{\textbf{27}}$	26.3	8	1522.8
keller6	3361	-	-	-	42	40.9	4	2200.7	42	34.2	1	<0.1	$\underline{\textbf{53}}$	51.8	2	1711.0
p_hat1000-1	1000	-	-	-	85	83.5	2	2494.7	19	18.4	8	1621.0	$\underline{\textbf{92}}$	91.2	4	1266.3
p_hat1000-2	1000	-	-	-	62	61.5	9	1934.0	28	21.6	1	<0.1	$\underline{\textbf{67}}$	67.0	20	279.7
p_hat1000-3	1000	-	-	-	66	65.0	2	1967.8	42	33.4	1	<0.1	$\underline{\textbf{68}}$	68.0	20	470.2
p_hat1500-1	1500	-	-	-	92	91.0	3	3197.2	19	17.9	3	2370.7	$\underline{\textbf{105}}$	104.7	14	1888.2
p_hat1500-2	1500	-	-	-	68	66.4	1	2475.4	42	28.5	1	<0.1	$\underline{\textbf{77}}$	76.9	19	1201.2
p_hat1500-3	1500	-	-	-	93	92.1	3	2231.9	52	42.1	1	<0.1	$\underline{\textbf{94}}$	93.0	1	2117.2
r1000.1c	1000	-	-	-	$\underline{\textbf{87}}$	86.7	13	2255.6	71	66.0	1	<0.1	85	83.7	2	2126.5
r1000.5	1000	-	-	-	371	369.5	3	2895.6	225	195.1	1	0.1	$\underline{\textbf{372}}$	369.0	2	3556.5

Notes. An underlined value indicates an improved best result, i.e., a new lower bound. A bold value corresponds to the best-known value in the literature.

Table B.5: Comparison results for the maximum MPC problem on Test Set II (23 large graphs) of our LDTPS algorithm, the restart simulated annealing RSA, the genetic algorithm GA, and the LocalSolver heuristic [21].

Graph	$\|V\|$	$\omega^\alpha(G)$	LocalSolver [21]		RSA				GA				**LDTPS**			
			sol	t(s)	sol$_{\text{best}}$	sol$_{\text{avg}}$	hit	t(s)	sol$_{\text{best}}$	sol$_{\text{avg}}$	hit	t(s)	sol$_{\text{best}}$	sol$_{\text{avg}}$	hit	t(s)
brock800_4	800	-	-	-	23	22.2	3	913.7	13	12.1	3	1063.3	**26**	25.8	15	674.4
C1000.9	1000	-	71	3600.0	67	66.1	4	1871.8	25	24.8	15	1396.5	**79**	78.0	2	1903.4
C2000.5	2000	-	-	-	17	16.1	1	992.7	12	10.6	2	<0.1	**20**	19.5	9	691.1
C2000.9	2000	-	-	-	67	65.6	3	3033.9	25	24.1	4	1616.6	**86**	84.8	2	2234.2
C4000.5	4000	-	-	-	16	15.7	13	658.8	13	11.9	4	<0.1	**21**	20.6	12	1085.1
DSJC500.1	500	-	50	3600.0	56	55.7	14	1810.8	42	38.4	2	<0.1	**57**	57.0	20	285.2
DSJC500.9	500	-	63	3605.0	64	62.7	2	1761.1	26	25.7	14	1527.0	**69**	68.2	3	606.5
DSJC1000.1	1000	-	-	-	61	60.3	6	2091.7	50	44.8	1	<0.1	**67**	66.1	7	2124.7
DSJC1000.5	1000	-	-	-	17	16.2	3	789.4	11	10.0	2	514.9	**19**	18.1	2	200.5
DSJC1000.9	1000	-	75	3600.0	67	65.6	3	1940.8	26	24.6	2	1367.2	**78**	77.2	4	1355.4
flat1000_50_0	1000	-	-	-	16	15.9	17	1324.2	11	9.9	2	391.1	**18**	18.0	20	856.4
flat1000_60_0	1000	-	-	-	16	15.9	18	1496.2	11	10.1	6	401.7	**18**	18.0	20	308.4
flat1000_76_0	1000	-	-	-	16	16.0	20	1274.3	11	10.0	3	343.5	**19**	18.1	1	301.0
keller5	776	-	46	3600.0	51	50.2	7	1729.9	30	23.5	2	291.7	**57**	57.0	20	755.1
keller6	3361	-	-	-	73	71.3	3	3329.8	62	47.7	1	<0.1	**115**	114.5	10	2280.7
p_hat1000-1	1000	-	-	-	74	72.2	1	1732.2	43	36.9	1	<0.1	**75**	74.7	14	2121.2
p_hat1000-2	1000	-	-	-	52	51.7	13	1709.3	31	26.4	3	<0.1	**56**	56.0	20	62.1
p_hat1000-3	1000	-	-	-	76	74.7	1	1766.8	18	17.3	8	1094.2	**82**	82.0	20	1243.7
p_hat1500-1	1500	-	-	-	85	83.5	1	1930.7	50	39.5	1	<0.1	**87**	86.3	5	2015.9
p_hat1500-2	1500	-	-	-	70	68.6	1	2739.1	40	26.4	1	<0.1	**80**	79.7	15	1534.1
p_hat1500-3	1500	-	-	-	97	95.1	1	3463.6	19	17.7	2	1395.5	**113**	111.5	1	1949.6
r1000.1c	1000	-	401	3622.0	377	373.9	1	3545.3	64	62.1	1	2725.2	**411**	408.9	1	3250.1
r1000.5	1000	-	-	-	212	210.9	3	2737.8	18	16.4	2	1537.7	**236**	236.0	20	1350.5

Notes. An underlined value indicates an improved best result, i.e., a new lower bound. A bold value corresponds to the best-known value in the literature.