Frequency-driven tabu search for the maximum s-plex problem

Yi Zhou [a] and Jin-Kao Hao [a,b,*]

[a] *LERIA, University of Angers, 2 bd Lavoisier, 49045 Angers, France*

[b] *Institut Universitaire de France, Paris, France*

Computers and Operations Research,
http://dx.doi.org/10.1016/j.cor.2017.05.005

Abstract

The maximum s-plex problem is an important model for social network analysis and other studies. In this study, we present an effective frequency-driven multi-neighborhood tabu search algorithm (FD-TS) to solve the problem on very large networks. The proposed FD-TS algorithm relies on two transformation operators (*Add* and *Swap*) to locate high-quality solutions, and a frequency-driven perturbation operator (*Press*) to escape and search beyond the identified local optimum traps. We report computational results for 47 massive real-life (sparse) graphs from the SNAP Collection and the 10th DIMACS Challenge, as well as 52 (dense) graphs from the 2nd DIMACS Challenge (results for 48 more graphs are also provided in the Appendix). We demonstrate the effectiveness of our approach by presenting comparisons with the current best-performing algorithms.

Keywords: Clique relaxation; Heuristic; Massive network; s-plex.

1 Introduction

Given a simple undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, let $N(v)$ denote the set of vertices adjacent to v in G. Then, an s-plex for a given integer $s \geq 1$ ($s \in \mathbb{Z}^+$) is a subset of vertices $C \subseteq V$ that satisfies the following condition: $\forall v \in C$, $|N(v) \cap C| \geq |C| - s$. Thus, each vertex of an s-plex C must be adjacent to at least $|C| - s$ vertices in the subgraph $G[C] = (C, E \cap (C \times C))$ induced by C.

* Corresponding author.
Email address: `hao@info.univ-angers.fr` (Jin-Kao Hao).

The maximum s-plex problem involves finding, for a fixed value of s, an s-plex of maximum cardinality among all possible s-plexes of a given graph. As indicated in [3], the maximum s-plex problem can be formulated as a binary linear program as follows:

$$\begin{aligned} \max \quad & \omega_s(G) = \textstyle\sum_{i \in V} x_i \\ \text{s.t.} \ & \textstyle\sum_{j \in V \setminus (N(i) \cup \{i\})} x_j \leq (s-1)x_i + \bar{d}_i(1 - x_i), \forall i \in V, \\ & x_i \in \{0, 1\}, \forall i \in V \end{aligned} \tag{1}$$

where x_i is the binary variable associated with vertex i, such that $x_i = 1$ if vertex i is in an s-splex, $x_i = 0$ otherwise. Also, $\bar{d}_i = |V \setminus N(i)| - 1$ denotes the degree of vertex i in the complement graph $\bar{G} = (V, \bar{E})$. Note that $i \notin N(i)$ by definition.

The s-plex concept was first introduced for graph-theoretic social network studies [27]. The decision version of the maximum s-plex problem with any fixed positive integer s is known to be NP-complete [3]. When s equals 1, the maximum s-plex problem reduces to the popular maximum clique problem, the decision version of which was among Karp's 21 NP-complete problems [14]. The maximum s-plex problem is often referred to as a *clique relaxation* model [23, 24]. Other clique relaxation models include s-defective clique [37], quasi-clique [8,21,22], and k-club [7], which are defined by relaxing the edge number, the edge density, and the pairwise distance of vertices in an induced subgraph, respectively. In addition to studies of social networks, the maximum s-plex problem has also been investigated in other contexts [5, 6, 10]. For instance, an interesting application of the maximum s-plex model was described in [6], where the maximum s-plex algorithm of [29] was used to find profitable diversified portfolios on the stock market.

Similar to a clique, an s-plex C has the heredity property, which means that every subset of vertices $C' \subset C$ remains an s-plex, i.e., the subgraph induced by C' always has the property of an s-plex [29]. The most successful combinatorial algorithms for s-plex essentially rely on the heredity property and a polynomial feasibility verification procedure. For example, a powerful exact algorithmic framework was introduced in [29] for detecting optimal hereditary structures (s-plex and s-defective clique), which is based on the maximum clique algorithm proposed in [20]. This algorithm performed well on the maximum s-plex problem for graphs in the 2nd DIMACS Challenge and popular large-scale social networks. Other exact algorithms for the s-plex problem include the following. A branch-and-cut algorithm was introduced in [3] based on a polyhedral study of the s-plex problem. Two branch-and-bound algorithms were presented in [16], which are based on popular exact algorithms for the maximum clique problem [9,20]. In [19], exact combinatorial algorithms were investigated using methods from parameterized algorithmics. Finally, a parallel algorithm for listing all the maximal s-plexes was introduced in [36].

However, given the computational complexity of the maximum s-plex problem, any exact algorithm is expected to require an exponential computational time to determine the optimal solution in the general case. Thus, it is useful to investigate heuristic approaches, which aim to provide satisfactory solutions within an acceptable time frame, but without a provable optimal guarantee for the solutions obtained. However, our literature review found only two heuristics for the maximum s-plex problem [12, 17], which are based on the general GRASP method [26]. This situation contrasts sharply with the huge body of heuristics for the conventional maximum clique problem [34] and other clique relaxation problems [24]. We note that exact and heuristic approaches may complement each other, and together they can enlarge the classes of problem instances that can be solved effectively. Moreover, they can even be combined within a hybrid approach, as exemplified in [17] where the GRASP heuristic was used to enhance the exact algorithm proposed in [3] to solve very large social network instances.

In this study, we aim to partially fill the gap in terms of heuristic methods for solving the maximum s-plex problem by introducing an effective heuristic approach. The main contributions of this study can be summarized as follows.

- From an algorithmic perspective, this is the first study to employ the tabu search metaheuristic [11] to solve the maximum s-plex problem (Section 2). Thus, the proposed frequency-driven tabu search algorithm (FD-TS) integrates several original components. First, FD-TS jointly employs three dedicated move operators called *Add*, *Swap*, and *Press*, two of which (*Swap* and *Press*) are applied for the first time to the maximum s-plex problem. Second, we introduce a frequency-based mechanism for perturbation and constructing initial solutions, which is proven to be more effective than a random mechanism. We also apply a peeling procedure to dynamically reduce the graph with the best identified lower bound. Finally, specific design decisions are made in order to handle very large networks with thousands and even millions of vertices.
- From a computational perspective, our experimental results indicate that the proposed algorithm performs very well with both sparse and dense graphs (Section 4). For 47 very large networks from the SNAP collection and the 10th DIMACS Challenge benchmark set, our algorithm successfully obtained or improved the best-known results from previous studies for $s = 2, 3, 4, 5$. Our algorithm even proved the optimality of many instances for the first time using the peeling procedure. For 52 dense graphs from the collection used in the 2nd DIMACS Challenge, our algorithm also obtained or improved the best-known results for $s = 2, 3, 4, 5$. To comprehensively assess the performance of our algorithm, we compared FD-TS with several

cutting edge algorithms, including the commercial CPLEX solver (version 12.6.1). Results of 48 additional graphs for the s-plex problem are also presented for the first time in the Appendix.

The remainder of this paper is organized as follows. Section 2 presents the FD-TS algorithm. Section 3 discusses the implementation and complexity issues related to FD-TS. Section 4 presents the computational results obtained on benchmark instances and provides comparisons with state-of-the-art algorithms. In the final section, we give our conclusions and discuss future research.

2 FD-TS algorithm for the maximum s-plex problem

2.1 General procedure

The general scheme of the proposed FD-TS algorithm is shown in Algorithm 1. FD-TS starts from an initial feasible solution (s-plex) built using the *Init_Solution()* procedure (Section 2.4), before entering the main multi-neighborhood local search procedure, *Freq_Tabu_Search()*, to improve the initial solution (Section 2.5). A vector $freq$, which records the number of times each vertex is moved in the last round of the *Freq_Tabu_Search()* procedure, is initialized as a null vector (Algorithm 1, line 3). This vector is used by the *Init_Solution()* procedure as well as the perturbation method explained in Section 2.5.3. If the solution returned by tabu search is better than the current best solution C^*, C^* is updated (Algorithm 1, lines 7–8). The new lower bound $|C^*|$ is then given to the *Peel()* procedure (Section 2.6) to reduce the current graph (Algorithm 1, line 9). If *Peel()* returns a reduced subgraph with fewer vertices than $|C^*|$, then C^* must be an optimal solution and the overall algorithm stops. Otherwise, the algorithm enters a new round of search to build a new starting solution with *Init_Solution()*, before improving the new starting solution with *Freq_Tabu_Search()* and reducing the graph with *Peel()* if this is possible. The algorithm continues until a given stopping condition (e.g., a cut-off time limit) is met.

2.2 Preliminary definitions

Given $G = (V, E)$, $s \in \mathbb{Z}^+$, let $C \subseteq V$ be a subset of vertices and $N(v)$ the set of vertices adjacent to v. The following definitions are provided, which are useful for the description of our algorithm.

We say that C is a (feasible) solution or an s-plex if $\forall v \in C, |N(v) \cap C| \geq$

Algorithm 1: Main framework of Frequency Driven Tabu Search

Input: Problem instance (G, s), predefined sample size q, maximum allowed iterations in tabu search L.

Output: The largest s-plex ever found

```
begin
    C* ← ∅;                                     /* the best solution found so far */
    freq(v) ← 0 for all v ∈ V;                  /* frequency count of vertex moves */
    while the stopping condition is not met do
        {C, freq} ← Init_Solution(G, s, freq, q);               /* §2.4 */
        {C, freq} ← Freq_Tabu_Search(G, s, C, freq, L);         /* §2.5 */
        if |C| > |C*| then
            C* ← C;
            G ← Peel(G, s, |C*|);                               /* §2.6 */
            if |V| ≤ |C*| then
                return C* ;                     /* return the best solution found */
end
return C*
```

$|C| - s$; otherwise, C is an infeasible solution (i.e., $\exists v \in C, |N(v) \cap C| < |C| - s$). For a vertex $v \in C$, we say that v is *saturated* (first introduced in [29]) if $|N(v) \cap C| = |C| - s$. If $|N(v) \cap C| < |C| - s$, v is *deficient*. Obviously, whenever a deficient vertex exists in C, C is an infeasible solution. The saturated set S of set C is defined as the set of all saturated vertices in C, i.e., $S = \{v \in C : |N(v) \cap C| = |C| - s\}$. We can see that if C is a 1-plex (i.e., a clique), then all the vertices in C are saturated. The search space Ω of G includes all s-plexes, $\Omega = \{C \subseteq V : \forall v \in C, |N(v) \cap C| \geq |C| - s\}$. For brevity, we also use $N(C)$ to denote the set of vertices in $V \setminus C$ with at least one adjacent vertex in C, $N(C) = \bigcup_{v \in C} N(v) \setminus C$.

In Figures 1 and 2, we provide examples of S and $N(C)$, as well as their uses in the definitions of the *Add* and *Swap* operators introduced in the next section.

Finally, the quality of any candidate solution (s-plex) $C \in \Omega$ is evaluated by its cardinality $|C|$. Thus, given two candidate solutions C' and C, C' is better than C if $|C'| > |C|$.

2.3 Move operators

Our FD-TS algorithm explores the search space Ω by jointly applying three move (or transformation) operators, *Add*, *Swap*, and *Press*, to generate new solutions in Ω from the current solution (or s-plex). If we let C be the incumbent solution, then each move operator transfers one vertex $v \in N(C)$ inside

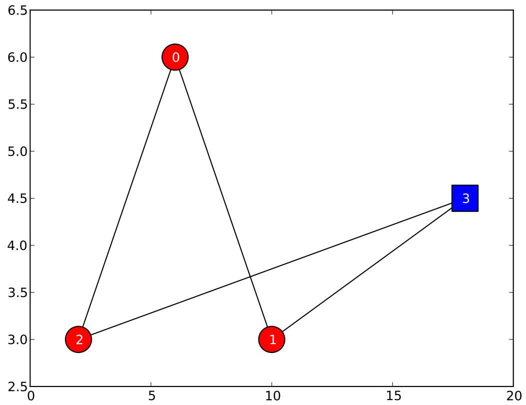

Fig. 1. Suppose $s = 2$, $C = \{0, 1, 2\}$ is the incumbent solution, $S = \{1, 2\}$ is the saturated set of C, then $M_1 = \{3\}$ and $C \cup \{3\}$ is an extended s-plex.

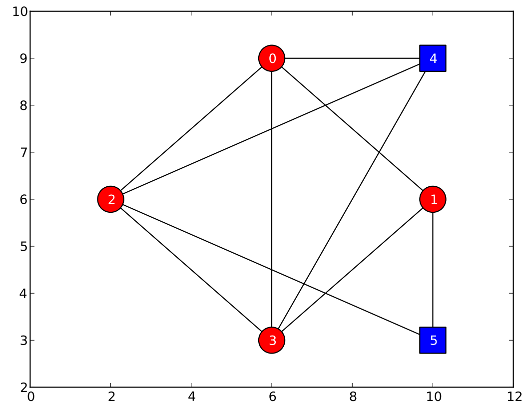

Fig. 2. Suppose $s = 2$, $C = \{0, 1, 2, 3\}$ is the incumbent solution, $S = \{1, 2\}$ is the saturated set of C, $N(C) = \{4, 5\}$. Then for the *Swap* operator, we get $A = \{4\}$ and exchangeable pair $< 4, 1 >$; $B = \{5\}$ and exchangeable pairs $< 5, 0 >$ and $< 5, 3 >$.

C and eliminates zero, one, or more vertices from C to keep C feasible. If we let OP be a move operator, then we use $C' \leftarrow C \oplus OP(v, X)$ to denote the new (neighboring) solution obtained by applying OP to C (X represents the subset of vertices eliminated from C, which can possibly be empty). The details of these operators are described as follows. For simplicity, when the subset of eliminated vertices X is empty or a singleton, the set notation of X is ignored.

(1) $Add(v)$: This operator extends the incumbent solution C by including a new vertex from $N(C)$. Clearly, each application of this operator will increase the cardinality of the solution by one, which always leads to a better solution. However, we must take special care to ensure that the extended solution remains an s-plex. Thus, we identify the following

vertex subset $M_1 \subseteq N(C)$, which has the required feasibility property (first introduced in [29]):

$$M_1 = \{v \in N(C) : |N(v) \cap C| \geq |C| - s + 1, S \setminus N(v) = \emptyset\} \quad (2)$$

By definition (2), if a vertex v in $N(C)$ is adjacent to at least $|C|-s+1$ vertices in C and adjacent to all the saturated vertices of C, then adding v to C yields a new solution C', the size of which is increased by one (see Fig. 1 for an example).

The set of neighboring solutions of C induced by $Add(v)$ is then given by:

$$\mathcal{N}_{Add} = \{C' : C' \leftarrow C \oplus Add(v), v \in M_1\} \quad (3)$$

The *Add* operator is used by the search algorithm to improve the quality of the incumbent solution.

(2) $Swap(v, u)$: This operator exchanges a vertex $v \in N(C)$ with another vertex u in the incumbent solution C $(u \in C)$, while keeping the quality of the solution unchanged. Similar to *Add*, to ensure the feasibility of the transformed solutions, we need to identify the set of suitable candidate pairs $< v, u > \in N(C) \times C$. Considering the definition of s-plex, a pair of vertices $< v, u >$ is eligible for exchange only if it satisfies one of the following two conditions.

- First, v is adjacent to at least $|C| - s$ vertices in C and u is the unique saturated vertex that is not adjacent to v (i.e., $S \setminus N(v) = \{u\}$).
- Second, v is adjacent to exactly $|C| - s$ vertices in C and these $|C| - s$ vertices must include all the saturated vertices (i.e., $S \setminus N(v) = \emptyset$), and u is an arbitrary vertex from $C \setminus N(v)$.

We use sets A and B to denote the candidate sets of v that satisfy the two conditions above, respectively, (see Fig. 2 for an example of these two types of vertices):

$$\begin{aligned} A &= \{v \in N(C) : |N(v) \cap C| \geq |C| - s, |S \setminus N(v)| = 1\} \\ B &= \{v \in N(C) : |N(v) \cap C| = |C| - s, S \setminus N(v) = \emptyset\} \end{aligned} \quad (4)$$

The set of neighboring solutions induced by $Swap(v, u)$ is then given by:

$$\mathcal{N}_{Swap} = \{C' : C' \leftarrow C \oplus Swap(v, u), (v \in A, u \in S \setminus N(v)) \vee (v \in B, u \in C \setminus N(v))\} \quad (5)$$

If we let $M_2 = A \cup B$, a practical method for generating a suitable pair $< v, u >$ is to first build the set M_2, then pick a vertex v from M_2, and finally determine an appropriate vertex u from $S \setminus N(v)$ or $C \setminus N(v)$. The *Swap* operator is used by the search algorithm to visit neighboring solutions of equal quality, so a transition with *Swap* is also called a sidewalk move.

(3) $Press(v, X)$: If we let $M_3 = N(C) \setminus (M_1 \cup M_2)$, this operator adds one vertex $v \in M_3$ to C and then eliminates two or more vertices from $C \setminus N(v)$ so the s-plex structure of the new solution is maintained. Obviously, given a vertex $v \notin M_1$, the set $C' = C \cup \{v\}$ is an infeasible solution because v or the vertices in $S \setminus N(v)$ become deficient (see Section 2.2) in $G[C']$. Therefore, to restore the feasibility of the transformed solution, this operator iteratively eliminates vertices from $(C' \setminus \{v\}) \setminus N(v)$ (i.e., $C \setminus N(v)$) until the solution becomes an s-plex. Preference is given to the *deficient* vertices in $C \setminus N(v)$ and if no deficient vertex exists in $C \setminus N(v)$, the vertices to be eliminated are selected randomly from C. All of the eliminated vertices are collected in X. The set of neighboring solutions induced by the $Press(v, X)$ operator is given by:

$$\mathcal{N}_{Press} = \{C' : C' \leftarrow C \oplus Press(v, X), v \in V \setminus C, X \subseteq C \setminus N(v)\} \quad (6)$$

By definition, *Press* eliminates at least two vertices from the solution (i.e., $|X| > 1$). Thus, the application of *Press* always degrades the quality of the current solution. Hence, this operator is only used by the perturbation procedure when the *Add* and *Swap* operators are no longer applicable, or when the search stagnates in local optima.

Next, we provide some additional comments about these operators and discuss some implementation issues.

- *Add* and *Swap* have been used in several algorithms for the maximum clique problem and the equivalent maximum independent set problem [4,13,25,35]. However, given the generality of the maximum s-plex problem, the definitions of these operators are different and more complex in the present study; in particular, the saturated set S must be involved. The *Press* operator can be treated as a dedicated application to the s-plex problem of the $PUSH$ operator introduced in [38] for the maximum weight clique problem.
- It should be noted that traditional maximum clique benchmark graphs, such as those from the 2nd DIMACS Challenge, include many *dense* graphs. Thus, most maximum clique algorithms typically operate on the complement graph $\bar{G}$ to search for the maximum independent sets in terms of runtime efficiency [13, 25, 33]. The FD-TS algorithm proposed in the present study is designed to handle very large real-world networks, which are typically sparse graphs. Consequently, FD-TS operates directly on the original graph based on its adjacency-list representation. In addition, we restrict the candidate vertices considered by the three move operators to $N(C)$ instead of $V \setminus C$ because $N(C)$ is much smaller than $V \setminus C$ for very large networks. Indeed, given the size of the networks considered (up to millions of ver-

tices), even a simple operation such as scanning all the vertices of $V \setminus C$ becomes too expensive, and it can slow down the algorithm considerably. Thus, special care is taken to avoid ineffective or unpromising examinations.
- When *Swap* and *Press* are applied, each dropped vertex is forbidden from rejoining the solution during a number of subsequent iterations to avoid revisiting previously examined solutions. This is achieved by using a tabu list (see Section 2.5.4).

2.4 Constructing the initial solutions

Each round of the FD-TS algorithm requires a starting solution (see Algorithm 1, line 5). In general, the starting solutions can be generated by any method that ensures the s-plex property. In our method, we employ the following construction procedure, which applies the *Add* operator while considering the frequency information for vertex moves.

From a random sample of q vertices ($q \in [50, 150]$), we use the vertex with the minimum frequency (ties are broken randomly) to create a singleton set C. From the singleton s-plex C, the procedure repeats the following three steps to extend the current solution: 1) generate the M_1 set from $N(C)$, 2) select one vertex with the minimum frequency from M_1 (ties are broken randomly), and 3) add the selected vertex to C. We repeat this process until M_1 becomes empty. The final s-plex C is returned as the initial solution.

The intuitive assumption that less frequently moved vertices are preferred is intended to make the initial solution as diverse as possible. Moreover, using a sample of q vertices instead of the whole set V for seeding the solution helps to reduce the computational overheads in the initialization procedure. This is particularly true for massive graphs because scanning all the vertices of the graph can be time consuming in this case. In Section 4.2, we discuss the calibration of the parameter q.

2.5 FD-TS

2.5.1 General procedure

The key search procedure employed in the FD-TS algorithm (see Algorithm 2) combines a double-neighborhood search procedure (we refer to this procedure as TS^2; Algorithm 2, lines 11–24) to facilitate intensification (to obtain local optima) and a frequency-based perturbation procedure (we refer to this procedure as PERTURB; Algorithm 2, lines 25–30) for diversification (to escape from local optima).

Algorithm 2: Frequency driven tabu search

```
Input: Problem instance (G, s), current solution C, max. allowed iterations L
Output: The largest s-plex found C_best
begin
    C_best ← C;
    tabu_list ← ∅;
    l ← 0 ;                          /* Counter of cycles of TS^2 + PERTURB runs */
    fix ← ∅;                         /* The vertices that are forbidden to drop */
    freq(v) ← 0 for all v ∈ V ;      /* Reset frequency records */
    λ ← 0 ;                          /* Counter of consecutive non-improving iterations */
    // Run TS^2 + PERTURB a maximum of L iterations
    while l ≤ L do
        // Start the two neighborhood tabu search procedure - TS^2
        Updating the saturated subset S ;                                /* Sect. 2.2 */
        Decompose set N(C) into M'_1 = {v ∈ M_1 : v ∉ tabu_list ∨ |C| + 1 > |C_best|},
        M'_2 = {v ∈ M_2 : v ∉ tabu_list}, M'_3 = {v ∈ M_3 : v ∉ tabu_list} ;  /* Sect. 2.3 */
        if M'_1 ≠ ∅ then
            v ← a random vertex from M'_1;
            C ← C ⊕ Add(v);
            freq(v) ← freq(v) + 1;
        else if M'_2 ≠ ∅ then
            (v, u) ← two random exchangeable vertices from M'_2 × N(C);   /* See Sect. 2.5.2 for
            selection rule */
            C ← C ⊕ Swap(v, u);
            Add u to tabu_list with tabu tenure T_u;                     /* Sect. 2.5.4 */
            freq(v) ← freq(v) + 1, freq(u) ← freq(u) + 1;
        if |C| > |C_best| then
            C_best ← C;
            λ ← 0;
        else
            λ ← λ + 1;
        // Run the PERTURB procedure
        if (No feasible Add and Swap operation) ∨ (λ > s ∗ |C_best|) ∧ M'_3 ≠ ∅) then
            v ← a vertex with maximum freq(v) from M'_3, break ties randomly;
            C ← C ⊕ Press(v, X) ;   /* X collects the removed vertices from C, Sect. 2.3 */
            fix ← {v}, λ ← 0;
            freq(v) ← 0, freq(u) ← freq(u) + 1 for all u in X;
            Add each u ∈ X to tabu_list with tabu tenure T_u;            /* Sect. 2.5.4 */
        l ← l + 1
end
Return C_best, freq;
```

Based on a given initial solution, TS^2 uses *Add* and *Swap* to improve the current solution until search stagnation occurs. PERTURB then applies *Press* to modify (perturb) the current local optimum and passes the modified solution to TS^2 for further improvement. FD-TS iterates this TS^2+PERTURB process a maximum of L times and then starts the next round of its search procedure.

In addition to the three move operators (*Add*, *Swap*, and *Press*), FD-TS employs a tabu mechanism (see Section 2.5.4) [11] and a frequency technique to ensure the effective exploration of the search space. A tabu list (*tabu_list*) is used to mark the vertices that are forbidden from joining the current solution during a specific number of iterations. Information related to the move frequency of each vertex v is collected, where $freq(v)$ records the number of times that v is operated upon by a move operator in the recent history. Initially, *tabu_list* is empty and the frequency of each vertex is set to 0. The

fix set (a singleton set used by the PERTURB procedure) records an added vertex, which is forbidden from moving out of the current solution in TS[2] and the counter l counts the completed iterations, the upper limit of which is given by the parameter L. Moreover, in order to avoid being trapped by local optima, a counter λ records the consecutive iterations that have passed since the last improvement of the current solution. Each time that the counter λ reaches a threshold, the perturbation procedure is triggered to modify the current solution using the $Press$ operator.

At the beginning of each iteration, set C is checked to identify the saturated subset S. Next, $N(C)$ is decomposed into three disjoint subsets: M_1', M_2', and M_3'. These sets correspond to M_1, M_2, and M_3 (defined in Section 2.3), respectively, but they exclude the vertices in the *tabu_list*. However, a vertex $v \in M_1$ is always retained in M_1' if adding v to C leads to a new solution that is better than the best solution found previously, i.e., $|C| + 1 > |C_{best}|$, regardless of the tabu status of the vertex.

2.5.2 Solution improvement with Add and Swap

The current solution is transformed successively by applying *Add* and *Swap*. Preference is given to the *Add* operator. Thus, whenever M_1' is not empty, *Add* is applied to improve the current solution by adding one vertex of M_1' to the solution (Algorithm 2, lines 11–14). If no vertex can be added to the solution ($M_1' = \emptyset$), but M_2' is not empty, then the search continues with the $Swap(v, u)$ operator by visiting solutions of equal quality (Algorithm 2, lines 15–19).

To provide $Swap(v, u)$ with an appropriate exchangeable pair $< v, u >$, v is first selected randomly from M_2', and u is then selected from the candidate set defined by the rules given in Section 2.3, while excluding the vertex recorded in fix. In particular, if v is a vertex of type (set) A, then u is selected from $S \setminus (N(v) \cup fix)$ (which is a trivial set with zero or one vertex); and if v is a vertex of type (set) B, u is selected randomly from $C \setminus (N(v) \cup fix)$. If there is no eligible candidate for u ($S \setminus N(v) = fix$ or $C \setminus N(v) = fix$), then we simply give up attempting to apply $Swap(v, u)$ and move on to the PERTURB procedure (Algorithm 2, line 25).

2.5.3 Perturbation with Press

Inevitably, at a certain search stage, no candidate vertex v or candidate pair $< v, u >$ is available for the $Add(v)$ or $Swap(v, u)$ operator (i.e., both M_1' and M_2' are empty or no eligible vertex can be found for u), or the search stagnates on the $Swap(v, u)$ operator. In the latter case, search stagnation occurs when the current solution has not been improved for $s * |C_{best}|$ consecutive iterations. The self-adaptive threshold, $s * |C_{best}|$, is identified based on the assumption that if the current s-plex cannot be improved even after the replacement of each of its vertices at least s times by the $Swap$ operator, then no better solution can be found in the current search region. To continue the search, the algorithm triggers the PERTURB procedure in order to move the search to a distant new region. This procedure applies the $Press(v, X)$ operator, where v is the vertex from M_3' with the largest $freq$ value and X collects the dropped vertices from $C \setminus N(v)$ to recover a feasible solution (Algorithm 2, lines 26–27). It is important to reset the frequency of vertex v (Algorithm 2, line 29), or the accumulated frequency of this vertex could dominate other vertices in subsequent cycles. The dropped vertices in X are added to the tabu list (Algorithm 2, line 30) and they will not be considered during the forbidden period, as explained in the next section.

2.5.4 Tabu tenure and management

As mentioned above, to avoid revisiting recently examined solutions, we use a tabu list to record the vertices dropped from the current solution in order to exclude them from consideration during a number of consecutive iterations. According to the definitions in Section 2.3, each application of $Swap(v, u)$ or $Press(v, X)$ removes one or more vertices from the current s-plex. Each dropped vertex u will be kept in the tabu list for the next T_u iterations (the tabu tenure), which is set according to the following two rules:

$$\begin{cases} T_u = 10 + random(0, |M_2|), \ u \text{ is dropped by } Swap(v, u) \\ T_u = 7, \ u \in X \text{ is dropped by } Press(v, X) \end{cases} \tag{7}$$

where $random(0, I)$ is a random integer in $\{0, \ldots, I\}$. These rules are based on previous studies of the related maximum clique problem [13, 35]. The first rule estimates the forbidden period for vertices for side-walk moves with the $Swap$ operator, which ensures that a dropped vertex will not be reconsidered for at least 10 iterations, and the second rule for the $Press$ operator prevents any dropped vertex from being reconsidered for a small number of iterations (seven in this case). Based on experiments, we observed that other values around these tabu tenures obtained similar performance. Thus, we selected the values used in [13, 35].

Finally, the tabu list is more useful when the number of candidate vertices for $Swap$ or $Press$ is limited, because a dropped vertex u will have a high probability of being added again to the solution if it is not prohibited. However, if numerous candidate vertices exist (such as in massive graphs), there is little chance of a dropped vertex being re-selected immediately. Thus, the tabu mechanism is more useful for graphs of limited size than massive graphs.

Given a graph $G = (V, E)$ and a parameter s, suppose that after tabu search (Algorithm 1, line 6), the current best s-plex has cardinality $|C^*|$ (lower bound of the maximum s-plex of G). Clearly, to further improve C^*, considering any vertex in V with a degree smaller than or equal to $|C^*| - s$ would not be beneficial because such a vertex cannot extend C^*. Thus, these vertices can be safely removed from the graph [1]. In FD-TS, we explore this strategy using the $Peel(G, s, |C^*|)$ procedure (Algorithm 1, line 9), which recursively deletes the vertices (and their incident edges) with a degree less than or equal to $|C^*| - s$ until no such vertex exists. Finally, if the subgraphs obtained after $Peel(G, s, |C^*|)$ have fewer vertices than $|C^*|$, then C^* must be an optimal solution because no better solution can exist.

For very dense graphs, the *Peel* procedure may not reduce the graph size greatly because the degrees of most vertices will remain larger than $|C^*| - s$. However, this technique is highly effective when it is applied to large sparse graphs such as massive real-world complex networks. As shown in Section 4.3, by using the high-quality lower bound $|C^*|$ provided by our tabu search procedure, this pruning technique can effectively reduce large sparse graphs to very small graphs (even the null graph).

We note that the idea of removing unpromising vertices was used previously in a GRASP heuristic for detecting dense subgraphs (quasi-cliques) in massive sparse graphs [1], as well as in several exact algorithms for the maximum clique and s-plex problems [3, 29, 32].

3 Implementation and time complexity

To effectively implement FD-TS, we maintain two structures: the vector $deg_C[v]$ (i.e., $deg_C[v] = |N(v) \cup C|, v \in V$) and the set $N(C)$ (i.e., $N(C) = \bigcup_{v \in C} N(v) \setminus C$), which are updated whenever the current solution C changes. Thus, each time a vertex (say u) is added to or removed from C by a move operator, we increase or decrease $deg_C[v]$ by one for each $v \in N(u)$. Since the set $N(C)$ must only contain vertices with $deg_C[v] > 0$, it is also adjusted when $deg_C[v]$ changes.

Next, we discuss the time complexity of the main components of the proposed algorithm. First, we consider the procedure for constructing initial solutions (Section 2.4). In each iteration, we need to build the subset M_1 from $N(C)$ and update $deg_C[v]$ and $N(C)$ after adding a vertex, which can be achieved in $O(|N(C)| + \Delta)$ ($\Delta = max_{v \in V}\{|N(v)|\}$).

The efficiency of TS^2 (Section 2.5) and PERTURB (Section 2.5.3) is closely related to the method used for building sets M'_1, M'_2, and M'_3 from $N(C)$ from scratch in each iteration. In our implementation, we build the saturated set S (Algorithm 2, line 9) from C in a time of $O(|C|)$ at the very beginning of each iteration. Then, for each vertex in $N(C)$ (e.g., u), we count the number of saturated vertices in the set of vertices adjacent to u, i.e., $|S \cap N(u)|$ (the saturated connectivity of u). Obviously, if the saturated connectivity of u is 0, $S \setminus N(u) = \emptyset$; and if the saturated connectivity of u is 1, $|S \setminus N(u)| = 1$. According to the definitions of M_1, M_2, and M_3, once the saturated connectivity of u and $deg_C[u]$ is known, it is trivial to identify u as an element of M'_1, M'_2 or M'_3. Consequently, decomposing $N(C)$ (Algorithm 2, line 10) can be achieved in $O(|N(C)| * \Delta)$.

Next, we consider the time complexity when employing the move operators. First, for the *Add* operator, we only need to update the vector $deg_C[v]$ and set $N(C)$ after reallocating a vertex v, which can be achieved in $O(\Delta)$. Second, to apply *Swap* with a vertex $v \in N(C)$, we first need to identify the other vertex $u \in C$. According to the rule defined in Section 2.5.2, the sets $S \setminus N(v)$ and $C \setminus N(v)$ can be identified by traversing sets C and $N(v)$ respectively. Thus, the time required to identify u is bounded by $O(|C|+\Delta) = O(2*\Delta+s) = O(\Delta+s)$ (because $|C| \leq \Delta + s$), while updating $deg_C[v]$ and $N(C)$ is bounded by $O(2 * \Delta) = O(\Delta)$ because two vertices are displaced during each application of *Swap*. Finally, each application of the $Press$ operator can be achieved in $O(\Delta^2)$.

Overall, one operator (*Add*, *Swap* or $Press$) is applied during one iteration, so the total time complexity of TS^2 and PERTURB for each iteration is bounded by $O(|C|+|N(C)|*\Delta+\Delta^2)$. We note that for sparse graphs, $|C|$, $N(C)$, and Δ are usually extremely small compared with the number of vertices in a graph.

4 Computational assessment

4.1 Benchmarks

In this section, we present computational evaluations of the proposed FD-TS algorithm for the maximum s-plex problem based on the following three sets of 79 (19, 17, and 43, respectively) benchmark graphs.

- **Stanford Large Network Dataset Collection (SNAP)** [1] . SNAP provides a large range of large-scale social and information networks [15], in-

[1] http://snap.stanford.edu/data/

cluding graphs retrieved from social networks, communication networks, citation networks, web graphs, product co-purchasing networks, Internet peer-to-peer networks, and Wikipedia networks. Some of these networks are directed graphs, so we simply ignored the direction of each edge and eliminate self-looping and duplicated edges.

- **The 10th DIMACS Implementation Challenge Benchmark (10th DIMACS)** [2] . This testbed contains many large networks, including artificial and real-world graphs from different applications. This benchmark set is popular for testing graph clustering and partitioning algorithms. More information about the graphs can be obtained from [2].
- **The 2nd DIMACS Implementation Challenge Benchmark (2nd DIMACS)** [3] . This set is from the 2nd DIMACS Implementation Challenge for the maximum clique problem. These instances cover real-world problems (e.g., coding theory, fault diagnosis, and the Steiner triple problem) and random graphs. The instances range from small graphs (50 vertices and 1,000 edges) to large graphs (4,000 vertices and 5,000,000 edges). These DIMACS graphs are very popular and they are generally used as a testbed for evaluating clique and s-plex algorithms. Unlike the instances in the two first benchmark sets, most of these instances are dense graphs.

4.2 Experimental protocol and parameter tuning

The proposed FD-TS algorithm was implemented in C++ [4] and compiled by g++ with optimization option '-O3'. All experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8 GHz and 2 GB RAM) running CentOS 6.5. When we solved the DIMACS machine benchmarking program fdmax.c [5] without compilation optimization flag, the run time on our machine was 0.40, 2.50, and 9.55 seconds for graphs r300.5, r400.5, and r500.5, respectively.

An interesting feature of FD-TS is that it has very few parameters. In addition to the tabu tenure discussed in Section 2.5.4, the parameter q (the sample number of vertices in Section 2.4) was set to 100. As indicated in Section 2.4, q is not a sensitive parameter so we simply fixed it to the middle value in the range of [50,150]. However, according to our experiments, the best value of parameter L, which specifies the maximum number of iterations in each round of tabu search, was highly dependent on the instance considered. We compared different values in $\{10, 100, 1000, 5000\}$ for L with $s = 2, 3, 4, 5$ for six selected instances from the 2nd DIMACS benchmark set (*MANN_a27, brock400_2,*

[2] http://www.cc.gatech.edu/dimacs10/downloads.shtml
[3] http://www.cs.hbg.psu.edu/txn131/clique.html
[4] We will make our program available.
[5] ftp://dimacs.rutgers.edu/pub/dsj/clique/

brock800_2, C1000.9, keller6, p_hat1500-3). As a trade-off, we retained $L = 1000$ because the algorithm achieved the best average solution quality in most cases. We also observed that fine tuning L for each pair (instances, s) could improve the performance. However, to report our computational results, we used the fixed setting $L = 1000$, which allowed the algorithm to obtain highly competitive results. Given the stochastic nature of our algorithm, we ran FD-TS to solve each instance 20 times for each given s, where each run was limited to a maximum of 180 CPU seconds (3 minutes).

4.3 Computational results for very large networks from SNAP and the 10th DIMACS Challenge

Table 1 shows the performance of FD-TS on 47 instances taken from the SNAP and 10th DIMACS benchmarks, namely, *all* 37 instances tested in [29] for the s-plex problem, as well as 10 instances from the recent literature [32]. Note that in [32], only the best clique size was reported, which is just a lower bound of the maximum s-plex for $s = 2, 3, 4, 5$. To be complete, we also report our results for the remaining 20 instances from [32] in the Appendix (Table A.1).

For each instance, the columns "Instance," "$|V|$," and "$|E|$" indicate basic information for the name, number of vertices, and number of edges, respectively. For each $s = 2, 3, 4, 5$, column "BKV" denotes the best-known objective values collected from [29] (for the first 37 instances) and [32] (for the last 10 instances). For the items in this column, an additional symbol "*" indicates that this objective value was proved to be optimal in [29], and the symbol ω shows that this value is the maximum clique size mentioned in [32] (which is a *lower bound* of the maximum s-plex). The "max" column shows the best objective value found by FD-TS among its 20 trials and the "time" column indicates the average time (in seconds) required for runs to obtain the best objective value (excluding the time spent reading the graph). The "$|V'|$" column shows the number of remaining vertices in the reduced subgraph after executing the $Peel(G, s, max)$ procedure (see Section 2.6). As mentioned earlier, if the number of vertices in the reduced subgraph (column "$|V'|$") is less than or equal to the lower bound (column "max"), then the latter is guaranteed to be the optimal solution. In these cases, we put a "*" beside the value.

Moreover, for instances where the optimum could not be determined in either [29] or FD-TS, we conducted an additional experiment with the CPLEX solver (version 12.6.1). First, we reduced the original graph by applying the $Peel(G, s, max)$ procedure. Then, for the reduced subgraph and a given $s \in \{2, 3, 4, 5\}$, a cutoff time of one CPU hour was used when running CPLEX with the mathematical model (1) presented in Section 1. Experiments were conducted using the same machine employed for running FD-TS. The best objective values found by CPLEX are listed in the "cplex" column, where "*" indicates the optimal values. If CPLEX was unable to load the model, the entry is marked by "N/A."

Table 1 shows that FD-TS always obtained the same or better objective values compared with the current best-known results (BKV values) (better objective values are highlighted with a bold font). In particular, FD-TS improved the best-known results for more instances as s increased (FD-TS found better solutions for $7, 15, 19, 21$ instances with $s = 2, 3, 4, 5$, respectively). This observation indicates that the instances become more challenging for exact algorithms with a larger s. Using the *Peel* procedure, FD-TS was also provably optimum for the first time for $6, 6, 6, 10$ instances and $s = 2, 3, 4, 5$, respectively. For the cases where the number of vertices in the reduced subgraph remained larger than the lower bound given by FD-TS, the majority of these cases were manageable when the subgraph had less than 10,000 vertices (exceptions include the instances 333SP, cage15, cit-Pattens, and wiki-Talk for $s = 2$). In terms of the computational time, FD-TS was able to obtain the best solutions in less than one second in most cases, including instances with millions of vertices. The average time required by FD-TS on cit-Patents for $s = 3$ was the longest but still less than one minute. Unfortunately, and similarly to CPLEX, it could not determine any solution better than that found by FD-TS in one hour when given the reduced subgraph. However, for the instances rgg_n_2_17_s0 with $s = 5$, and rgg_n_2_20_s0 with $s = 4$ and 5, optimal solutions were also obtained by CPLEX. Finally, we note that for the instances coPapersCiteseer, coPapersDBLP, and cond-mat-2005, the size of the maximum clique was the same as the size of the maximum s-plex for $s = 2, 3, 4, 5$.

Table 1
Computational results of FD-TS on 47 large networks from the SNAP Collection and the 10th DIMACS Implementation Challenge.

Instance	$\|V\|$	$\|E\|$	s=2					s=3					s=4					s=5				
			BKV	max	time	$\|V'\|$	cplex	BKV	max	time	$\|V'\|$	cplex	BKV	max	time	$\|V'\|$	cplex	BKV	max	time	$\|V'\|$	cplex
adjnoun	112	425	6*	6	0.00	102	-	8*	8	0.00	102	-	8*	8	0.00	89	-	10*	10	0.00	102	-
celegans_metabolic	453	2025	10*	10	0.00	313	-	11*	11	0.00	429	-	13*	13	0.00	240	-	14*	14	0.00	429	-
dolphins	62	159	6*	6	0.00	53	-	7*	7	0.00	36	-	7*	7	0.00	45	-	9*	9	0.00	45	-
email	1133	5451	12*	12	0.01	238	-	12*	12	0.02	349	-	12*	12	0.01	349	-	13*	13	0.01	848	-
football	115	613	10*	10	0.01	114	-	11*	11	0.01	115	-	12*	12	0.00	115	-	12*	12	0.00	115	-
jazz	198	2742	30*	30	0.00	130	-	30*	30	0.00	164	-	30*	30	0.00	127	-	30*	30	0.00	127	-
karate	34	78	6*	6	0.00	22	-	6*	6	0.00	10	-	8*	8	0.00	10	-	11^a	9*	0.00	0	-
netscience	1589	2742	20*	20	0.00	50	-	20*	20	0.00	137	-	20*	20	0.00	158	-	20	20*	0.00	20	-
polblogs	1490	16715	23*	23	0.02	541	-	27*	27	0.03	894	-	29*	29	0.04	489	-	32	32	0.03	293	32*
polbooks	105	441	7*	7	0.00	103	-	9*	9	0.00	103	-	10*	10	0.00	105	-	11*	11	0.00	105	-
power	4941	6594	6*	6	0.00	36	-	6*	6	0.00	231	-	6	8	0.01	12	8*	8	9	0.01	12	9*
PGPgiantcompo	10680	24316	29*	29	0.03	115	-	31*	31	0.02	43	-	33*	33	0.02	41	-	35*	35	0.02	41	-
as-22july06	22963	48436	19*	19	0.02	117	-	21*	21	0.02	110	-	22*	22	0.04	110	-	24*	24	0.09	104	-
astro-ph	16706	121251	57*	57	0.06	113	-	57*	57	0.05	113	-	57*	57	0.07	113	-	57*	57	0.07	165	-
caidaRouterLevel	192244	609066	20*	20	0.31	2039	-	22	**23**	0.50	1290	13	23	**24**	0.74	1290	12	23	**26**	0.82	1098	14
cnr-2000	325557	2738969	85*	85*	6.75	0	-	86*	86	10.51	0	-	86*	86*	7.66	86	-	86*	86*	8.09	86	-
coAuthorsCiteseer	227320	814134	87*	87*	0.21	87	-	87*	87*	0.21	87	-	87*	87*	0.22	87	-	87*	87*	0.22	87	-
coAuthorsDBLP	299067	977676	115*	115*	0.27	115	-	115*	115*	0.26	115	-	115*	115*	0.27	115	-	115*	115*	0.27	115	-
cond-mat-2005	40421	175691	30*	30*	0.08	30	-	30*	30*	0.07	30	-	30*	30*	0.08	30	-	30*	30	0.08	57	-
memplus	17758	54196	97*	97*	0.08	97	-	97*	97*	0.11	97	-	97*	97*	0.11	97	-	97*	97*	0.10	97	-
rgg_n_2_17_s0	131072	728753	16*	16*	0.15	0	-	17*	17*	0.15	0	-	18*	18*	0.14	0	-	18	18	0.14	34	18*
rgg_n_2_19_s0	524288	3269766	19*	19*	0.69	0	-	19*	19*	0.66	19	-	20*	20*	0.64	19	-	20	**21***	0.66	19	-
rgg_n_2_20_s0	1048576	6891620	18*	18	1.42	59	-	19*	19	1.37	59	-	19	20	1.36	59	20*	19	20	1.33	172	20*
cit-HepPh	34546	420877	24*	24	0.37	4768	-	25	**27**	0.27	3498	5	25	**30**	0.19	2344	11	25	**32**	0.30	1804	22
cit-HepTh	27770	352285	28*	28	1.20	4815	-	31	31	0.70	3999	N/A	31	**34**	0.81	2922	6	31	**37**	0.72	1726	23
email-EuAll	265214	364481	19*	19	0.15	1357	-	22	22	0.22	1198	20	22	**25**	0.22	1055	23	27	27	0.26	988	26
p2p-Gnutella04	10876	39994	5*	5	0.22	6089	-	5	**7**	0.09	5433	4	6	**9**	0.10	4857	5	7	**10**	0.01	4857	N/A
p2p-Gnutella24	26518	65369	5*	5	0.07	9271	-	5	**6**	0.02	9271	N/A	5	**8**	0.10	7480	N/A	5	**9**	0.08	7480	N/A
p2p-Gnutella25	22687	54705	5*	5	0.03	7892	-	5	**6**	0.01	7892	N/A	5	**8**	0.01	6091	5	5	**10***	0.01	0	-
soc-Epinions1	75879	405740	28*	28	0.09	4156	12	28	**32**	0.09	3791	12	28	**37**	0.09	3280	12	28	**39**	0.16	3178	10
soc-Slashdot0811	77360	469180	31*	31	0.05	3665	-	33	**34**	0.06	3188	6	36	**38**	0.12	2596	7	36	**40**	0.10	2416	7
soc-Slashdot0902	82168	504230	32*	32	0.05	3669	-	35	35	0.06	3185	6	36	**40**	0.10	2435	9	36	**42**	0.10	2287	7
web-BerkStan	685230	6649470	202*	202	4.19	392	-	202*	202	4.47	392	-	202*	202	4.21	392	-	202*	202	6.55	392	-
web-Google	875713	4322051	46*	46*	1.13	0	-	47*	47*	1.24	0	-	48*	48*	1.34	0	-	48*	48*	1.29	48	-
web-NotreDame	325729	1090108	155*	155	0.74	1367	-	155*	155	0.71	1367	-	155*	155	0.84	1367	-	155*	155	0.99	1367	-
web-Stanford	281903	1992636	64*	64	3.86	741	-	64*	64	5.61	985	-	64	65	4.77	985	65	64	**66**	6.80	985	65
wiki-Vote	7115	100762	21*	21	0.10	2098	-	24	24	0.11	2005	12	26	**27**	0.12	1925	16	27	**28**	0.13	1925	17
333SP	3712815	11108633	$4(\omega)$	**5**	1.19	2261408	N/A	$4(\omega)$	**6**	0.71	2261408	N/A	$4(\omega)$	**7**	0.51	2261408	N/A	$4(\omega)$	**8**	0.46	2261408	N/A
belgium.osm	1441295	1549970	$3(\omega)$	**5***	0.66	0	-	$3(\omega)$	**5***	0.35	5	-	$3(\omega)$	**6***	0.37	5	-	$3(\omega)$	**7***	0.34	5	-
cage15	5154859	47022346	$6(\omega)$	6	2.48	5135355	N/A	$6(\omega)$	**8**	5.68	5134115	N/A	$6(\omega)$	**10**	6.02	5091619	N/A	$6(\omega)$	**11**	26.71	5091619	N/A
coPapersCiteseer	434102	16036720	$845(\omega)$	845*	7.92	845	-	$845(\omega)$	845*	7.20	845	845	$845(\omega)$	845*	7.01	845	-	$845(\omega)$	845*	8.51	845	-
coPapersDBLP	540486	15245729	$337(\omega)$	337*	2.83	337	-	$337(\omega)$	337*	3.03	337	-	$337(\omega)$	337*	3.18	337	-	$337(\omega)$	337*	3.28	337	-
amazon0312	400727	2349869	$11(\omega)$	**12***	0.54	0	-	$11(\omega)$	**13***	0.58	0	-	$11(\omega)$	**14***	0.61	0	-	$11(\omega)$	**15***	0.60	0	-
amazon0505	410236	2439437	$11(\omega)$	**12***	0.50	0	-	$11(\omega)$	**13***	0.58	0	-	$11(\omega)$	**14***	0.62	0	-	$11(\omega)$	**15***	0.64	0	-
amazon0601	403394	2443408	$11(\omega)$	**12***	0.75	0	-	$11(\omega)$	**13***	0.82	0	-	$11(\omega)$	**14***	0.79	0	-	$11(\omega)$	**15***	0.74	0	-
cit-Patents	3774768	16518947	$11(\omega)$	**17**	21.25	29585	N/A	$11(\omega)$	**21**	51.02	14717	N/A	$11(\omega)$	**26**	16.23	6293	N/A	$11(\omega)$	**31**	11.09	3604	5
wiki-Talk	2394385	4659565	$26(\omega)$	**32**	1.18	11327	N/A	$26(\omega)$	**36**	3.60	9814	N/A	$26(\omega)$	**41**	1.51	8376	N/A	$26(\omega)$	**44**	3.72	7719	N/A

Note a: The value of 11 reported in [29] for 'karate' is wrongly claimed to be the optimal solution.

4.4 Computation results for graphs from the 2nd DIMACS Challenge

Table 2 shows the computational results obtained by FD-TS for 52 classical hard instances from the 2nd DIMACS set, with $s = 2, 3, 4, 5$. The first group contains *all* 36 instances that have been studied by four state-of-the-art s-plex algorithms [3, 16, 19, 29]. The second group includes 16 additional *large* 2nd DIMACS instances with at least 800 vertices, which have not been tested previously by any existing s-plex algorithm. For the sake of completeness, we also tested the remaining 28 instances of this benchmark set, for which no previous s-plex result is available. These results are reported in Table A.2 of the Appendix.

For each instance, the following information is included. The "$|V|$" column shows the number of vertices in the original graph. The "ω" column indicates the best-known maximum clique size reported previously [34] (lower bounds for the maximum s-plex). The "BKV" column indicates the best-known objective values obtained by the algorithms in [3, 16, 19, 29] (proven optima are indicated by "*"). The letters between parentheses following each "BKV" value indicate the algorithm(s) that obtained the BKV value.

- "B" - A branch-and-cut algorithm [3] based on polyhedral analysis of the convex hull of the maximum s-plex problem. This algorithm was evaluated based on instances from the 2nd DIMACS set with $s = 1, 2$. Each instance was solved within a maximum of 3 hours on a machine with a 2.66 GHz XEON® processor, 3 GB RAM, and 120 GB HDD.
- "M" - A branch-and-bound algorithm [16] adapted from the classical maximum clique algorithm [20]. Results were obtained based on 2nd DIMACS instances with $s = 2, 3, 4$. The experiments were conducted on a machine with a 2.2 GHz Dual-Core AMD Opteron processor and 3 GB RAM. A time limit of one hour was allowed to solve each instance.
- "T" - A generalized algorithm framework used to detect optimal hereditary structures in graphs [29]. For the maximum s-plex problem, this approach was tested based on instances from the 2nd DIMACS, 10th DIMACS and SNAP benchmark sets, for $s = 2, 3, 4, 5$. Experiments were conducted with a Dell Optiplex GX620 computer with an Intel Core™2 Quad 3 GHz processor and 4 GB RAM with a time limit of 3 hours for each instance.
- "H" - Exact combinatorial algorithms based on methods from parameterized algorithmics [19]. Results were reported for a subset of the 2nd DIMACS instances ($s = 1, 2$) with a time limit of 3 hours on a machine with an AMD Athlon 64 3700+ 2.2 GHz CPU, 3 GB RAM, and 1M L2 cache.

For instances where the optimal solution has not been proven by any of the algorithms mentioned above (i.e., no "*" is indicated for "BKV"), we used the CPLEX solver to solve these instances (i.e., their reduced subgraphs after applying the *Peel* procedure, see Section 2.6) with a time limit of one CPU hour on our computer. The "cplex" column shows the best feasible solutions attained by CPLEX. The "max(ave)" column reports the maximum value achieved by FD-TS in 20 runs and the average value (in parentheses) if the 20 best values were not the same. The "time" column shows the average time (in seconds) required by the runs that obtained the best value among the 20 runs. Obviously, the total time allowed to FD-TS in 20 runs (180*20 = 3600 s) was exactly one hour.

Table 2 shows that the FD-TS algorithm matched or improved (highlighted in bold font) the current best-known results with $s = 2, 3, 4, 5$. The average objective values obtained by our algorithm were even better than the best-known values based on these instances for different s (except for MANN_a27 and MANN_a45 with $s = 2$). In terms of the stability of the best solution, for most of the small instances ($|V| \leq 400$) in the first group, the best solution could be obtained in each of the 20 runs (except for MANN_a27, brock400_4 and san200_0.7_2 with $s = 2$, brock400_1 with $s = 5$). The larger graphs in the second group of the table (which were not reported previously), such as brock800_X, CXXX.X, hamming10-4, and keller6, represent the most challenging cases for FD-TS because the best solution could not be found in every run. Moreover, for instances such as brock400_1, brock800_2, brock800_3, keller6, and p_hat1500-2, FD-TS failed to achieve a 100% success rate as s increased. However, for instances such as MANN_a27 and brock800_4, there was no correlation between the success rate and the value of s. In terms of the computational time, FD-TS achieved its best values rather quickly, since it rarely exceeded one minute, whereas CPLEX failed to solve these instances (in fact, the reduced subgraphs) to optimality for any s within one hour. Nevertheless, for the cases where the optimal value is still unknown, the lower bounds obtained by CPLEX were competitive compared with the four other reference algorithms [3, 16, 19, 29].

Table 2
Computational results of FD-TS on 52 benchmark instances of the 2nd DIMACS Implementation Challenge

			s=2				s=3				s=4				s=5			
instance	$\|V\|$	ω	BKV	cplex	max(ave)	time	BKV	cplex	max(ave)	time	BKV	cplex	max(ave)	time	BKV	cplex	max(ave)	time
MANN_a9	45	16	26*(BMTH)	-	26	0.00	36*(MT)	-	36	0.00	36*(MT)	-	36	0.00	44(T)[a]	**45**	**45**	0.00
MANN_a27	378	126	236*(H)	-	236(235.90)	12.64	351*(T)	-	351	0.41	351(M)	351	351	0.45	351(T)	351	351	0.45
MANN_a45	1035	345	662(BH)	662*	662(661.40)	5.46	990(M)	990*	990	7.40	990(M)	990*	990	7.48	990(M)	990*	990	7.44
brock200_1	200	21	25(B)	26	26	0.15	24(M)	29	**30**	0.05	27(M)	33	**35**	3.59	27(T)	38	**39**	2.36
brock200_2	200	12	13*(MTH)	-	13	0.01	16*(T)	-	16	0.27	17(TM)	17	**18**	0.06	17(TM)	20	20	0.04
brock200_3	200	15	17*(T)	-	17	0.02	19(T)	19	**20**	0.02	19(T)	22	**23**	0.03	19(T)	25	**26**	0.09
brock200_4	200	17	20*(TH)	-	20	0.06	20(T)	22	**23**	0.07	21(M)	26	26	0.03	21(M)	28	**30**	0.20
brock400_1	400	27	23(T)	29	**30**	0.42	23(T)	34	**36**	7.24	23(T)	37	**41**	41.37	23(T)	43	**46**(45.50)	36.57
brock400_2	400	29	27(B)	28	**30**	0.51	27(M)	33	**36**	15.37	29(M)	37	**41**	33.51	29(M)	41	**45**	2.14
brock400_4	400	33	27(B)	28	**33**(31.20)	64.18	27(B)	34	**36**	4.60	30(M)	36	**41**	1.76	30(M)	44	**46**	25.67
c-fat200-1	200	12	12*(BMTH)	-	12	0.00	12*(MT)	-	12	0.00	12*(MT)	-	12	0.00	14*(T)	-	14	0.00
c-fat200-2	200	24	24*(BMTH)	-	24	0.01	24*(MT)	-	24	0.01	24*(MT)	-	24	0.02	24*(T)	-	24	0.01
c-fat200-5	200	58	58*(BMTH)	-	58	0.01	58*(MT)	-	58	0.01	58*(MT)	-	58	0.00	58*(T)	-	58	0.00
c-fat500-1	500	14	14*(BMTH)	-	14	0.00	14*(MT)	-	14	0.00	14*(MT)	-	14	0.00	15*(T)	-	15	0.00
c-fat500-2	500	26	26*(BMTH)	-	26	0.00	26*(MT)	-	26	0.00	26*(MT)	-	26	0.00	26*(T)	-	26	0.00
c-fat500-5	500	64	64*(BMTH)	-	64	0.02	64*(MT)	-	64	0.02	64*(MT)	-	64	0.03	64*(T)	-	64	0.04
c-fat500-10	500	126	126*(BMTH)	-	126	0.08	126*(MT)	-	126	0.22	126*(MT)	-	126	0.23	126*(T)	-	126	0.16
hamming6-2	64	32	32*(BMTH)	-	32	0.00	32*(MT)	-	32	0.00	40*(MT)	-	40	0.00	48*(T)	-	48	0.00
hamming6-4	64	4	6*(BMTH)	-	6	0.00	8*(MT)	-	8	0.00	10*(MT)	-	10	0.00	12*(T)	-	12	0.00
hamming8-2	256	128	128*(BMT)	-	128	0.09	128*(MT)	-	128	0.09	128(MT)	128	**129**	22.41	128(MT)	152	152	0.97
hamming8-4	256	16	16*(BMT)	-	16	0.01	20(T)	20	20	0.01	20(T)	24	25	0.28	20(T)	32	32	0.02
hamming10-2	1024	512	512*(M)	-	512	8.97	512(M)	512	512	4.66	512(M)	512	512	8.62	512(M)	512	**513**(512.15)	16.82
johnson8-2-4	28	4	5*(BMTH)	-	5	0.00	8*(MT)	-	8	0.00	9*(MT)	-	9	0.00	12*(T)	-	12	0.00
johnson8-4-4	70	14	14*(BMTH)	-	14	0.00	18*(TH)	-	18	0.00	22*(T)	-	22	0.00	24(T)	24	**28**	0.00
johnson16-2-4	120	8	10*(T)	-	10	0.00	16(T)	16	16	0.00	19(T)	19	19	0.00	21(T)	24	**24**	0.00
keller4	171	11	15*(BMTH)	-	15	0.00	21*(T)	-	21	0.09	22(T)	23	23	0.06	22(T)	28	28	0.02
p_hat300-1	300	8	10*(MTH)	-	10	0.00	12*(MT)	-	12	0.00	14*(T)	-	14	0.01	14(T)	15	**16**	0.02
p_hat300-2	300	25	30*(T)	-	30	0.01	30(T)	36	36	0.02	33(M)	41	41	0.06	33(M)	46	46	0.03
p_hat300-3	300	36	43(B)	43	**44**	0.07	43(B)	52	52	0.06	43(B)	59	59	0.09	43(B)	65	65	0.11
p_hat500-1	500	9	12*(T)	-	12	0.06	14*(T)	-	14	0.20	14(T)	16	16	0.11	14(T)	17	**18**	0.15
p_hat700-1	700	11	13*(M)	-	13	0.06	13(M)	14	**15**	0.13	13(M)	15	**17**	0.45	13(M)	18	**19**	2.00
p_hat700-2	700	44	50(B)	51	**52**	0.06	50(B)	60	**62**	1.35	50(B)	68	**70**	0.26	50(B)	75	**79**	9.42
p_hat700-3	700	62	73(B)	75	**76**	0.54	73(B)	88	**89**	2.13	73(B)	97	**100**	1.46	73(B)	106	**109**	1.35
san200_0.7_2	200	18	24(M)	26	26(25.40)	9.74	36(M)	37	37	0.21	48(M)	49	49	1.90	48(M)	60	60	0.02
san200_0.9_1	200	70	90(M)	90*	90	0.01	125*(M)	-	125	0.02	125(M)	125*	125	0.03	125(M)	125*	125	0.03
hamming10-4	1024	40	41(BM)	44	**48**	1.53	46(M)	53	**64**	1.13	51(M)	64	**68**(67.20)	20.64	51(M)	73	**79**(78.05)	34.37
brock800_1	800	23	-	21	**25**	10.90	-	26	**29**	12.35	-	29	**34**(33.20)	24.43	-	33	**37**	27.12
brock800_2	800	24	-	22	**25**	11.36	-	27	**30**(29.30)	31.20	-	30	**34**(33.15)	26.40	-	33	**38**(37.15)	32.46
brock800_3	800	25	-	24	25	12.68	-	25	**30**(29.20)	11.71	-	30	**34**(33.15)	28.46	-	32	**38**(37.10)	36.78
brock800_4	800	26	-	22	26(25.55)	56.21	-	26	**29**	14.35	-	31	**33**	32.04	-	33	**37**	33.58
C1000.9	1000	68	-	71	**81**(80.55)	39.65	-	84	**95**(93.75)	58.59	-	97	**107**(106.00)	48.91	-	108	**119**(118.15)	60.39
C2000.5	2000	16	-	14	**19**(18.95)	27.96	-	18	**22**(21.90)	62.35	-	21	**25**(24.50)	22.79	-	23	**28**(27.15)	11.72
C2000.9	2000	80	-	76	**90**(88.90)	65.21	-	88	**105**(103.40)	69.41	-	94	**118**(116.80)	63.85	-	107	**132**(129.65)	76.66
C4000.5	4000	18	-	12	**20**	39.23	-	14	**23**	69.37	-	13	**26**(25.55)	53.64	-	16	**29**(28.20)	26.93
keller6	3361	59	-	52	**63**	3.60	-	65	**90**(87.80)	66.21	-	77	**107**(103.45)	67.88	-	90	**125**(123.20)	73.91
p_hat1000-1	1000	10	-	11	**13**	0.28	-	13	**15**	0.15	-	15	**18**	4.09	-	18	**20**	7.17
p_hat1000-2	1000	46	-	52	**56**	0.43	-	64	**67**	0.81	-	71	**76**	28.89	-	82	**84**	1.30
p_hat1000-3	1000	48	-	77	**82**	0.33	-	95	**98**	2.19	-	109	**111**	3.50	-	117	**122**	29.84
p_hat1500-1	1500	12	-	13	**14**	1.46	-	14	**17**	22.47	-	16	**19**	3.73	-	16	**21**	1.43
p_hat1500-2	1500	65	-	71	**80**	1.75	-	90	**93**	0.41	-	99	**107**(106.50)	29.46	-	113	**117**(116.55)	41.90
p_hat1500-3	1500	94	-	108	**114**	0.61	-	125	**133**	17.85	-	141	**150**	3.90	-	154	**164**	48.76
san1000	1000	15	-	17	17	9.39	-	25	25	6.73	-	33	33	1.70	-	41	41	6.39

Note a: The value of 44 reported in [29] for MANN_a9 is wrongly claimed to be the optimal solution.

4.5 Impact of frequency information

As described in Sections 2.4 and 2.5, the construction procedure and perturbation procedure are guided by frequency information. In this section, we evaluate the effectiveness of this frequency strategy. We compared the original FD-TS algorithm with a variant, FD-TS-R, in which the frequency-based vertex selection rule was replaced by a random selection rule. In particular, to create a new solution, FD-TS-R randomly adds a vertex from M_1 to the current solution (Section 2.4) and randomly selects a vertex from M_3' for perturbation (Algorithm 2, line 31).

To better differentiate FD-TS and FD-TS-R, we selected 27 instances from the three benchmark sets, such that the selected instances cover different characteristics (random vs real-world, dense vs sparse) and are sufficiently challenging based on the search effort required to attain the best solutions according to the results of Tables 1 and 2. For this experiment, both FD-TS and FD-TS-R were run 20 times to solve each instance, each run being limited to 20 seconds for the 2nd DIMACS instances and 180 seconds for the other (larger) instances. We compared the average objective values reached by both algorithms ("ave" columns), the average time required to first obtain the best objective value ("time" columns), and the improvement in the average objective value achieved by FD-TS as a percentage ("ave_imp" column).

Table 3 shows the results achieved for $s = 2, 3, 4, 5$ respectively. A difference in the average solution quality obtained by the two algorithms was only observed with the 2nd DIMACS instances (the first 12 instances). For the large instances, both FD-TS and FD-TS-R converged so fast that the best solution was found quite early (the average time required to first obtain the best solution was less than one second in most cases). For the 2nd DIMACS instances, FD-TS achieved better solutions than FD-TS-R for $5, 6, 8, 8$ instances with $s = 2, 3, 4, 5$, respectively (marked in bold font). In addition, for $4, 4, 4, 2$ instances with $s = 2, 3, 4, 5$, respectively, the average objective values found by FD-TS were worse than those found by FD-TS-R (marked in italic font). In general, there was a slight advantage when using the frequency mechanism, and it increased with s. This experiment confirms that the frequency mechanism is helpful for solving hard dense graphs that require persistent search efforts.

Table 3
Impact of frequency information - comparison between FD-TS and FD-TS-R.

instance	s=2					s=3					s=4					s=5				
	FD-TS-R		FD-TS		ave_imp	FD-TS-R		FD-TS		ave_imp	FD-TS-R		FD-TS		ave_imp	FD-TS-R		FD-TS		ave_imp
	ave	time	ave	time		ave	time	ave	time		ave	time	ave	time		ave	time	ave	time	
C1000.9	79.40	8.59	79.70	7.45	**0.38%**	92.70	7.95	92.80	7.89	**0.11%**	104.95	5.18	105.10	6.68	**0.14%**	116.85	10.40	117.30	9.88	**0.38%**
C2000.5	18.45	3.25	18.50	5.05	**0.27%**	21.05	1.29	21.10	2.71	**0.24%**	24.15	3.35	24.10	4.06	*-0.21%*	26.85	6.27	26.95	9.35	**0.37%**
C2000.9	87.80	6.97	87.85	9.95	**0.06%**	102.20	9.50	101.70	7.97	*-0.49%*	115.05	8.65	114.85	9.61	*-0.17%*	128.25	9.76	127.70	8.95	*-0.43%*
C4000.5	19.60	4.80	19.25	2.75	*-1.82%*	22.35	3.96	22.25	3.86	*-0.45%*	25.05	3.99	25.20	4.18	**0.60%**	27.90	4.81	27.90	6.42	0.0%
brock800_1	24.85	4.98	24.70	4.30	*-0.61%*	28.90	7.04	28.95	5.38	**0.17%**	32.70	4.57	33.15	5.91	**1.36%**	36.40	3.93	36.50	4.00	**0.27%**
brock800_2	24.85	6.20	24.90	7.63	**0.20%**	29.00	7.62	28.95	5.08	*-0.17%*	32.80	5.74	32.75	5.48	*-0.15%*	36.45	5.56	36.65	6.22	**0.55%**
brock800_3	24.85	8.74	24.80	5.31	*-0.20%*	28.75	5.06	29.00	7.19	**0.86%**	32.80	7.74	32.95	6.84	**0.46%**	36.30	3.60	36.45	5.27	**0.41%**
brock800_4	24.70	6.84	24.85	7.28	**0.60%**	28.65	5.76	28.70	5.99	**0.17%**	32.40	5.43	32.55	5.49	**0.46%**	36.25	4.71	36.30	3.07	**0.14%**
hamming10-4	48.00	1.36	48.00	1.16	0.0%	64.00	0.98	64.00	1.12	0.0%	66.75	5.73	66.85	3.98	**0.15%**	77.50	3.92	77.45	3.05	*-0.06%*
keller6	63.00	2.98	63.00	3.33	0.0%	84.70	8.68	85.70	8.46	**1.17%**	99.50	9.00	98.95	11.69	*-0.56%*	121.00	9.89	121.40	14.53	**0.33%**
p_hat1500-2	80.00	1.66	80.00	2.02	0.0%	93.00	0.47	93.00	0.48	0.0%	106.00	1.40	106.05	1.50	**0.05%**	116.00	1.49	116.10	2.78	**0.09%**
san1000	16.80	4.73	16.75	2.50	*-0.30%*	25.00	3.30	24.95	2.46	*-0.20%*	32.95	3.35	33.00	1.29	**0.15%**	41.00	3.59	41.00	3.82	0.0%
333SP	5.00	1.21	5.00	1.19	0.0%	6.00	0.71	6.00	1.04	0.0%	7.00	1.03	7.00	0.51	0.0%	8.00	0.96	8.00	0.46	0.0%
cage15	6.00	2.41	6.00	2.48	0.0%	8.00	8.89	8.00	5.68	0.0%	10.00	5.84	10.00	6.02	0.0%	11.00	34.77	11.00	26.71	0.0%
caidaRouterLevel	20.00	0.55	20.00	0.31	0.0%	23.00	0.94	23.00	0.51	0.0%	24.00	0.81	24.00	0.74	0.0%	26.00	0.52	26.00	0.82	0.0%
cit-HepPh	24.00	0.32	24.00	0.37	0.0%	27.00	0.46	27.00	0.27	0.0%	30.00	0.37	30.00	0.19	0.0%	32.00	0.34	32.00	0.30	0.0%
cit-HepTh	28.00	1.31	28.00	1.20	0.0%	31.00	1.51	31.00	0.70	0.0%	34.00	1.02	34.00	0.81	0.0%	37.00	1.39	37.00	0.72	0.0%
cit-Patents	17.00	15.56	17.00	21.25	0.0%	21.00	18.90	21.00	51.02	0.0%	26.00	9.55	26.00	16.23	0.0%	31.00	10.13	31.00	11.09	0.0%
email-EuAll	19.00	0.13	19.00	0.15	0.0%	22.00	0.19	22.00	0.22	0.0%	25.00	0.17	25.00	0.22	0.0%	27.00	0.15	27.00	0.26	0.0%
p2p-Gnutella04	5.00	0.02	5.00	0.02	0.0%	7.00	0.08	7.00	0.09	0.0%	9.00	0.12	9.00	0.10	0.0%	10.00	0.00	10.00	0.01	0.0%
p2p-Gnutella24	5.00	0.04	5.00	0.07	0.0%	6.00	0.01	6.00	0.02	0.0%	8.00	0.12	8.00	0.10	0.0%	9.00	0.32	9.00	0.18	0.0%
p2p-Gnutella25	5.00	0.03	5.00	0.03	0.0%	6.00	0.01	6.00	0.01	0.0%	8.00	0.01	8.00	0.01	0.0%	10.00	0.08	10.00	0.10	0.0%
soc-Slashdot0811	31.00	0.06	31.00	0.05	0.0%	34.00	0.06	34.00	0.06	0.0%	38.00	0.13	38.00	0.12	0.0%	40.00	0.08	40.00	0.10	0.0%
soc-Slashdot0902	32.00	0.06	32.00	0.05	0.0%	35.00	0.07	35.00	0.06	0.0%	40.00	0.11	40.00	0.10	0.0%	42.00	0.12	42.00	0.10	0.0%
web-NotreDame	155.00	1.21	155.00	0.74	0.0%	155.00	1.30	155.00	0.71	0.0%	155.00	1.22	155.00	0.84	0.0%	155.00	1.18	155.00	0.99	0.0%
wiki-Talk	32.00	1.40	32.00	1.18	0.0%	36.00	3.41	36.00	3.60	0.0%	41.00	2.68	41.00	1.51	0.0%	44.00	3.82	44.00	3.72	0.0%
wiki-Vote	21.00	0.07	21.00	0.10	0.0%	24.00	0.13	24.00	0.11	0.0%	27.00	0.16	27.00	0.12	0.0%	28.00	0.08	28.00	0.13	0.0%

5 Conclusions and perspectives

The NP-hard maximum s-plex problem is significant in both theory and practice. In this study, we proposed an effective local search algorithm for solving this problem heuristically based on the general tabu search method. To ensure its efficiency, the proposed algorithm combines a multi-neighborhood search procedure with vertex-moving frequency, where the search process is driven by two intensification oriented operators (*Add* and *Swap*) and one diversification operator (*Press*). Dedicated rules are defined to explore the neighborhoods introduced by these operators. Information regarding vertex moves is collected and used to guide the construction of the starting solutions and the perturbation process. A graph peeling technique is also integrated to dynamically reduce large sparse graphs.

We assessed the performance of the proposed algorithm using three popular benchmark sets: 47 instances from the Stanford Large Network Dataset Collection and the 10th DIMACS Implementation Challenge, and 52 dense graphs from the 2nd DIMACS Implementation Challenge. For the SNAP and 10th DIMACS benchmarks, FD-TS obtained improved solutions (new lower bounds) for $7, 15, 19, 20$ instances when $s = 2, 3, 4, 5$, respectively. Moreover, many of these solutions were proved to be optimal using the *Peel* procedure. FD-TS also performed very well on the instances from the 2nd DIMACS benchmark set. The *Peel* procedure was no longer effective for these dense graphs, but FD-TS still obtained the current best-known results for all of the instances and discovered better solutions for most instances compared with four recent reference algorithms and the powerful CPLEX solver. Additional results for 48 more graphs from the above benchmark sets showed in the Appendix further demonstrated the performance of the proposed algorithm.

Several areas of research require further investigation. First, to achieve high search robustness across a large range of problem instances with very different characteristics, it would be useful to develop adaptive and learning techniques to help the algorithm to adjust its search strategies dynamically. Second, it would be interesting to explore other ways of employing frequency information to improve the performance of the algorithm. For instance, we could investigate frequency information in new selection rules for the transformation operators as well as other guided perturbation mechanisms such as that proposed in [4]. Finally, it would be interesting to adapt the ideas introduced in this work to design search algorithms for other clique-relaxations, such as s-defective clique [37], quasi-clique [1, 22, 31] and k-club [7, 18, 28, 30].

Acknowledgment

We are grateful to the reviewers and Dr. Jean-Yves Potvin, the Area Editor in charge of the paper for their constructive comments which have helped us to significantly improve the paper. The work was partially supported by the PGMO (2014-0024H) project from the Jacques Hadamard Mathematical Foundation (Paris, France). Support for Yi Zhou from the China Scholarship Council (2013-2017) is also acknowledged.

References

[1] J. Abello, M.G.C. Resende and S. Sudarsky. Massive quasi-clique detection. Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN 2002). Lecture Notes in Computer Science 2286: 598-612, 2002.

[2] D.A. Bader, H. Meyerhenke, P. Sanders, D. Wagner (eds.). Graph partitioning and graph clustering. 10th DIMACS Implementation Challenge Workshop. February 13-14, 2012. Georgia Institute of Technology, Atlanta, GA. Contemporary Mathematics 588. American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science, 2013.

[3] B. Balasundaram, S. Butenko and I.V. Hicks. Clique relaxations in social network analysis: the maximum k-plex problem. Operations Research 59(1): 133-142, 2011.

[4] U. Benlic and J.K. Hao. Breakout local search for maximum clique problems. Computers and Operations Research 40(1): 192-206, 2013.

[5] N. Berry, T. Ko, T. Moy, J. Smrcka, J. Turnley and B. Wu. Emergent clique formation in terrorist recruitment. Proceedings of The AAAI-04 Workshop on Agent Organizations: Theory and Practice, San Jose, California, July 25, 2004.

[6] V. Boginski, S. Butenko, O. Shirokikh, S. Trukhanov and J.G. Lafuente. A network-based data mining approach to portfolio selection via weighted clique relaxations. Annals of Operations Research 216(1):23-24, 2014.

[7] J.M. Bourjolly, G. Laporte and G. Pesant. Heuristics for finding k-clubs in an undirected graph. Computers and Operations Research 27(6): 559-569, 2000.

[8] M. Brunato, H.H. Hoos and R. Battiti. On effectively finding maximal quasi-cliques in graphs. Learning and Intelligent Optimization, Lecture Notes in Computer Sciences 5313: 41-55, 2008.

[9] R. Carraghan, P.M. Pardalos. An exact algorithm for the maximum clique problem. Operations Research Letters 9(6): 375-382, 1990.

[10] D. Gibson, R. Kumar and A. Tomkins. Discovering large dense subgraphs in massive graphs. Proceedings of the 31st International Conference on Very Large Data Bases, pp. 721-732, 2005.

[11] F. Glover and M. Laguna. Tabu Search. Kluwer, 1997.

[12] K.R. Gujjula, K.A. Seshadrinathan and A. Meisami. A hybrid metaheuristic algorithm for the maximum k-plex problem. Examining Robustness and Vulnerability of Networked Systems. NATO Science for Peace and Security Series - D: Information and Communication Security 83-92, 2014.

[13] Y. Jin and J.K. Hao. General swap-based multiple neighborhood tabu search for the maximum independent set problem. Engineering Applications of Artificial Intelligence 37: 20-33, 2015.

[14] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher (eds), Complexity of Computer Computations, pp. 85-103. Plenum Press, New York. 1972.

[15] J. Leskovec and A. Krevl, SNAP Datasets: Stanford Large Network Dataset Collection, http://snap.stanford.edu/data, June 2014.

[16] B. McClosky and I.V. Hicks. Combinatorial algorithms for the maximum k-plex problem. Journal of Combinatorial Optimization 23(1): 29-49, 2012.

[17] Z. Miao and B. Balasundaram. Cluster detection in large-scale social networks using k-plexes. IIE Annual Conference. Proceedings January 1, 2012

[18] E. Moradi and B. Balasundaram. Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. Optimization Letters, DOI:10.1007/s11590-015-0971-7 (In press).

[19] H. Moser, R. Niedermeier and M. Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. Journal of Combinatorial Optimization 24(3): 347-373, 2012.

[20] P.R.J. Östergård. A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120(1): 197-207, 2002.

[21] F.M. Pajouh, Z. Miao and B. Balasundaram. A branch-and-bound approach for maximum quasi-cliques. Annals of Operations Research 216(1): 145-161, 2014.

[22] J. Pattillo, A. Veremyev, S. Butenko and V. Boginski. On the maximum quasi-clique problem. Discrete Applied Mathematics 161(1-2): 244-257, 2013.

[23] J. Pattillo, N. Youssef and S. Butenko. Clique relaxation models in social network analysis. Handbook of Optimization in Complex Networks 58: 143-162, 2012.

[24] J. Pattillo, N. Youssef and S. Butenko. On clique relaxation models in network analysis. European Journal of Operational Research 226(1):9-18, 2013.

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

[26] M.G.C. Resende and C.C. Ribeiro. Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. Handbook of Metaheuristics, pp. 283-319, 2010.

[27] S.B. Seidman and B.L. Foster. A graph-theoretic generalization of the clique concept. Journal of Mathematical Sociology 6: 139-154, 1978.

[28] S. Shahinpour and S. Butenko. Algorithms for the maximum k-club problem in graphs. Journal of Combinatorial Optimization 26(3): 520-554, 2013.

[29] S. Trukhanov, C. Balasubramaniam, B. Balasundaram and S. Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. Computational Optimization and Applications 56(1): 113-130, 2013.

[30] A. Veremyev and V. Boginski. Identifying large robust network clusters via new compact formulations of maximum k-club problems. European Journal of Operational Research 218(2): 316-326, 2012.

[31] A. Veremyev, O. A. Prokopyev, S. Butenko and E.L. Pasiliao. Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. Computational Optimization and Applications 64(1): 177-214, 2016.

[32] A. Verma, A. Buchana and S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. INFORMS Journal on computing 27(1): 164-177, 2015.

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

[34] Q. Wu and J.K. Hao. A review on algorithms for maximum clique problems. European Journal of Operational Research 242(2): 693-709, 2015.

[35] Q. Wu, J.K. Hao and F. Glover. Multi-neighborhood tabu search for the maximum weight clique problem. Annuals of Operations Research 196(1): 611-634, 2012.

[36] B. Wu and X. Pei. A parallel algorithm for enumerating all the maximal k-plexes. Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp.476-483, 2007.

[37] H. Yu, A. Paccanaro, V. Trifonov and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. Bioinformatics 22(7): 823-829, 2006.

[38] Y. Zhou, J.K. Hao and A. Goëffon. PUSH: a generalized operator for the maximum weight clique problem. European Journal of Operational Research 257(1): 41-54, 2017.

A Computational results on additional instances

This Appendix includes additional results of our FD-TS algorithm and CPLEX for 48 instances from the three benchmark sets. Note that these instances have not been tested previously by any s-plex algorithm. Only lower bounds (from the best-known maximum clique sizes [32, 34]) are available. Table A.1 contains the 20 large SNAP and 10th DIMACS instances while Table A.2 includes the remaining 28 instances of the 2nd DIMACS Challenge.

Table A.1
Computational results of FD-TS on 20 large networks from the SNAP Collection and the 10th DIMACS Implementation Challenge

				s=2				s=3				s=4				s=5			
Instance	$\|V\|$	$\|E\|$	ω [32]	max	time	$\|V'\|$	cplex	max	time	$\|V'\|$	cplex	max	time	$\|V'\|$	cplex	max	time	$\|V'\|$	cplex
p2p-Gnutella30	36682	88328	3	**5**	0.01	12097	N/A	**7**	0.04	9779	4	8	0.01	9779	5	**10**	0.02	1458	10
p2p-Gnutella31	62586	147892	4	**5**	0.04	19765	N/A	**6**	0.02	19765	N/A	**8**	0.02	16174	N/A	10	0.04	1004	10*
Amazon0302	262111	899792	7	**8***	0.24	0	-	**9***	0.53	0	-	**10***	0.24	0	-	**11***	0.25	0	-
kron_g500-simple-logn16	65536	2456071	136	**140**	0.89	6885	N/A	**144**	0.92	6885	30	**148**	1.22	6884	N/A	**152**	1.38	6883	18
citationCiteseer	268495	1156647	10	**13**	2.49	6731	N/A	**14**	0.54	6731	N/A	**16**	0.72	2779	4	**18**	0.81	1150	17
eu-2005	862664	16138468	387	388	4.73	405	388*	390	5.17	405	390*	391	7.57	405	391*	**393***	9.08	0	-
in-2004	1382908	13591473	489	**490***	3.51	0	-	**491***	6.90	0	-	**491***	7.54	491	-	**491***	8.89	491	-
rgg_n_2_21_s0	2097152	14487995	19	19*	2.83	19	-	19	2.82	115	19*	20	2.62	115	20*	**22***	2.53	19	-
rgg_n_2_22_s0	4194304	30359198	20	20*	5.67	20	-	**21***	5.64	20	-	**22***	5.38	20	-	**23***	5.29	20	-
rgg_n_2_23_s0	8388608	63501393	21	**22***	12.55	0	-	**22***	11.17	22	-	**23***	12.16	22	-	**24***	11.50	22	-
rgg_n_2_24_s0	16777216	132557200	21	**22***	24.94	0	-	**23***	25.33	0	-	**24***	21.84	0	-	**25***	24.95	0	-
uk-2002	18520486	261787258	944	944*	45.00	944	-	944*	44.23	944	-	944*	47.05	944	-	944*	47.13	944	-
G_n_pin_pout	100000	501198	3	**5**	3.95	98961	N/A	**5**	0.03	99734	N/A	**7**	42.96	98961	N/A	**8**	26.29	98961	N/A
preferentialAttachment	100000	499985	6	**7***	0.02	0	-	**8***	0.03	0	-	**9***	0.06	0	-	**10***	0.09	0	-
smallworld	100000	499998	5	**7**	0.00	99982	N/A	**8**	0.00	99982	N/A	**9**	0.00	99982	N/A	**10**	0.00	99982	N/A
luxembourg.osm	114599	119666	2	**4***	0.02	0	-	**5***	0.02	0	-	**6***	0.02	0	-	**7***	0.02	0	-
wave	156317	1059331	5	**9**	0.68	119747	N/A	**9**	0.85	155074	N/A	**11**	1.57	119747	N/A	**12**	0.81	119747	N/A
audikw1	943695	38354076	36	36	1.08	937779	N/A	**39**	34.83	936015	N/A	**45**	69.79	875349	N/A	**45**	75.77	929820	N/A
ldoor	952203	22785136	21	21	0.00	952203	N/A	21	0.01	952203	N/A	21	0.01	952203	N/A	**23**	0.10	952203	N/A
ecology1	1000000	1998000	2	**4***	0.00	0	-	**4**	0.00	1000000	N/A	**6***	0.01	0	-	**7***	0.00	0	-

Table A.2
Computational results of FD-TS on 28 benchmark instances of the 2nd DIMACS Implementation Challenge

instance	$\|V\|$	ω [34]	s=2			s=3			s=4			s=5		
			cplex	max(ave)	time	cplex	max(ave)	time	cplex	max(ave)	time	cplex	max(ave)	time
C125.9	125	34	43*	43	0.00	51*	51	1.89	58	58	0.07	65*	65	0.38
C250.9	250	44	53	**55**	8.43	63	**65**	22.29	75	75	4.81	84	84	4.72
C500.9	500	57	63	**69**	10.67	74	**81**(80.95)	57.72	84	**92**(91.75)	55.11	97	**103**(102.25)	36.65
DSJC1000_5	1000	15	15	**18**	26.61	18	**21**	25.35	21	**24**(23.05)	5.63	24	**27**(26.25)	32.14
DSJC500_5	500	13	15	**16**	0.29	17	**19**	2.81	20	**21**	0.12	23	**24**	1.07
MANN_a81	3321	1100	2162*	2162(2113.90)	139.35	3240*	3240(3125.35)	138.36	3240*	3240(2788.70)	147.51	3240*	*3135*(2660.75)	190.01
brock400_3	400	31	28	**30**	0.35	33	**36**	6.59	38	**41**	5.90	43	**46**(45.90)	55.95
gen200_p0.9_44	200	44	53	53	0.14	66	66	0.09	76	76	0.03	84	84	0.05
gen200_p0.9_55	200	55	57	57	0.02	64	64	0.14	73	73	0.12	80	80	0.19
gen400_p0.9_55	400	55	64	**68**(67.70)	65.76	85	**87**	26.62	108	**112**	0.19	124	124	0.16
gen400_p0.9_65	400	65	73	73(71.60)	28.01	100	**101**(100.45)	10.68	132	132	0.25	138	138	0.15
gen400_p0.9_75	400	75	78	**79**(78.05)	30.62	112	**114**	0.35	136	136	0.10	136	136	0.12
johnson32-2-4	496	16	21	21	0.02	32	32	0.05	38	38	0.38	48	48	0.09
keller5	776	27	31	31	0.09	41	**45**	8.19	46	**53**(52.75)	53.67	58	**61**	5.04
p_hat500-2	500	36	42	42	0.02	49	**50**	0.12	55	**57**	0.07	62	62	0.25
p_hat500-3	500	50	60	**62**	0.18	71	**72**	1.24	80	**81**	1.94	88	**89**	1.73
san200_0.7_1	200	30	31	31	0.59	46	46(45.70)	1.51	60	60	0.01	75*	75	0.01
san200_0.9_2	200	60	71	71	2.47	105*	105	0.02	105*	105	0.02	105*	105	0.03
san200_0.9_3	200	44	53	**54**(53.95)	72.31	73	73	7.48	96*	96	0.08	100*	100	0.03
san400_0.5_1	400	13	15	15	1.24	22	22	3.61	29	29	4.92	35	**36**(35.70)	55.40
san400_0.7_1	400	40	41	41	0.20	61	61	2.49	80	81(80.45)	17.54	100	100	0.06
san400_0.7_2	400	30	32	32	7.22	46	**47**(46.10)	0.46	61	61	0.57	76	76	10.34
san400_0.7_3	400	22	27	27(26.30)	12.27	38	38	11.17	50	50(49.45)	24.71	61	61	0.29
san400_0.9_1	400	100	102	102(101.30)	9.09	150	150	0.09	200*	200	0.12	200*	200	0.15
sanr200_0.7	200	18	22	22	0.01	25	26	0.03	30	30	0.14	33	33	0.05
sanr200_0.9	200	42	51	51	0.61	60	**61**	2.25	69	69	0.07	76	77	5.74
sanr400_0.5	400	13	14	**15**	0.02	18	18	0.09	20	**21**	0.56	23	**24**	1.63
sanr400_0.7	400	21	25	**26**	1.03	28	**30**	0.45	32	**35**	5.31	35	**39**	31.01