An $O^*(1.4658^n)$-time exact algorithm for the maximum bounded-degree-1 set problem*

Maw-Shang Chang[1], Li-Hsuan Chen[2], Ling-Ju Hung[1]†, Yi-Zhi Liu[2],
Peter Rossmanith[3], Somnath Sikdar[3]
[1]Department of Computer Science and Information Engineering
HungKuang University
43302 Sha Lu, Taichung, Taiwan
{mschang,ljhung}@sunrise.hk.edu.tw
[2]Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi 62102, Taiwan
{clh100p,lyichih100m}@cs.ccu.edu.tw
[3]Department of Computer Science, RWTH Aachen University
52056 Aachen, Germany
{rossmani,sikdar}@cs.rwth-aachen.de

Abstract

A bounded-degree-1 set *S in an undirected graph $G = (V, E)$ is a vertex subset such that the maximum degree of $G[S]$ is at most one. Given a graph G, the* Maximum Bounded-Degree-1 Set *problem is to find a bounded-degree-1 set S of maximum size in G. A notion related to bounded-degree sets is that of an* s-plex *used to define the cohesiveness of subgraphs in social networks. An s-plex S in a graph $G = (V, E)$ is a vertex subset such that for each $v \in S$, $\deg_{G[S]}(v) \geq |S| - s$. One can easily show that a graph G has a 2-plex of size k iff the complement graph of G has a bounded-degree-1 set of size k. Both the* Maximum 2-Plex *problem and the* Maximum Bounded-Degree-1 Set *problem are NP-hard. We give a simple branch-and-reduce algorithm using branching strategies with at most three branches for the* Maximum Bounded-Degree-1 Set *problem. We analyze the running time of the algorithm using* measure-and-conquer *and show that it runs in time $O^*(1.4658^n)$ which is faster than previous exact algorithms.*

*This research is partially supported by the National Science Council of Taiwan under grants NSC 101–2221–E–241–019–MY3 and NSC 102–2221–E–241–007–MY3.

†Ling-Ju Hung (corresponding author) is supported by the National Science Council of Taiwan under grant NSC 102–2811–E–241–001.

1 Introduction

A *bounded-degree-d set* S in an undirected graph $G = (V, E)$ is a vertex subset such that the maximum degree of $G[S]$ is at most d. The Maximum Bounded-Degree-d Set problem is to find a bounded-degree-d set S of maximum size in the input graph $G = (V, E)$. An s-plex S in a graph $G = (V, E)$ is a vertex subset such that for each $v \in S$, $\deg_{G[S]}(v) \geq |S| - s$.

Maximum Bounded-Degree-d Set (Max d-bds)
Input: A graph $G = (V, E)$.
Output: A vertex set $S \subseteq V$ of maximum cardinality such that S is a bounded-degree-d set.

Maximum s-Plex (Max s-plex)
Input: A graph $G = (V, E)$.
Output: A vertex set $S \subseteq V$ of maximum cardinality such that S is a s-plex.

Note that a vertex subset S is an s-plex in G if and only if it is a bounded-degree-$(s-1)$ set in the complement graph $\bar{G}$.

The Maximum Bounded-Degree-d Set (Max d-bds) problem is NP-complete because its equivalent problem, the Maximum $(d+1)$-plex (Max $(d+1)$-plex) problem is NP-complete [3]. The dual of the Max d-bds problem is Minimum Bounded-Degree-d Deletion Set (Min d-bdd) where one is required to find

a vertex subset D of minimum size in the input graph $G = (V, E)$ such that $V \setminus D$ is a bounded-degree-d set. This dual problem has received some attention from the parameterized complexity community. A series of fixed-parameter tractable algorithms were developed for MIN d-BDD problem, with $|D|$ as parameter [24, 21, 12, 9, 15, 23]. Betzler *et al.* [5] showed that when parameterized by the treewidth of the input graph, MIN d-BDD problem is W[1]-hard. In the same paper, they showed that the problem is fixed-parameter tractable for the following parameters: (1) the combined parameter treewidth *and* the number of vertices to delete; (2) the feedback edge set number. Chang *et al.* [9] gave a $O^*(1.5171^n)$-time branch-and-reduce algorithm for the MAX 1-BDS problem. By applying measure-and-conquer analysis, this algorithm runs in time $O^*(1.4834^n)$ [10]. Chang and Hung [11] showed that the MAX 1-BDS problem cannot be approximated to a ratio greater than $n^{\epsilon-1}$ in polynomial time for all $\epsilon > 0$ unless P = NP. Some moderately exponential time approximation algorithms were given for the MAX 1-BDS problem in [11].

The notion of s-plexes is a degree relaxed variant of cliques and was defined to study the cohesiveness of subgroups in social networks [26]. The MAX s-PLEX problem can be formulated as a 0/1 integer program [1, 3] and was shown to be W[1]-hard with respect to the size of s-plexes as parameter [21]. Some branch-and-bound algorithms were given for solving the MAX s-PLEX problem based on different upper bounds and lower bounds found by heuristic algorithms [27, 22]. Some graph editing problems are studied on finding a disjoint union of s-plexes [20, 6]. Balasundaram *et al.* [2] referred the MAX d-BDS problem as the problem of finding maximum-cardinality co-$(d+1)$-plexes. Wu and Pei [28] gave an algorithm to enumerate all maximal s-plexes.

In this paper, we give an $O^*(1.4658^n)$-time algorithm for the MAX 1-BDS problem. This algorithm is faster than the previous best due to Chang *et al.* [10] and can be used to solved the MAX 2-PLEX problem in time $O^*(1.4658^{\Delta(G)})$ where $\Delta(G)$ is the maximum degree of the input graph.

2 Preliminaries

For functions f and g we write $f(n) = O^*(g(n))$ if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is a polynomial.

A *branch-and-reduce algorithm* is a recursive procedure consisting of *reduction rules* and *branching rules*. The reduction rules are used to reduce the problem size, and the branching rules are used to branch the original problem into smaller subproblems. The *measure-and-conquer* approach is used to analyze the running time of branch-and-reduce algorithms [13, 4, 16, 14, 17, 18, 7, 19]. In a usual analysis of a branch-and-reduce algorithm, the input size n is the number of vertices in the input graph. The branching step is written as a recurrence in terms of n and the running time is obtained by standard techniques of solving recurrences. In a measure-and-conquer analysis, each vertex is assigned a weight in the range $[0, 1]$ and the size of the problem is measured by the total weight w_G of all vertices. The key issue in a measure-and-conquer analysis is how to assign weights to the vertices to obtain a tighter bound on the running time of the algorithm.

In [9, 10, 23], the branch-and-reduce algorithms designed for solving the MAX 1-BDS problem use a *standard branching rule*: for a vertex v, it branches into $\deg(v) + 2$ subproblems that either (i) v is not in the solution set, or (ii) v is in the solution set but none of its neighbors is in the solution set, or (iii) v and one of its neighbors are in the solution set where $\deg(v) = |N(v)|$. Since the number of branches is unbounded, this branching strategy makes the algorithm and the time analysis more complicated than algorithms applying branching strategies with bounded number of branches. In this paper, we give a simple branch-and-reduce algorithm for the MAX 1-BDS problem that applies the branching strategies having at most three branches. It also uses a 2-coloring strategy to record those vertices in the graph having a neighbor being selected in the solution set. By applying the measure-and-conquer approach that assigns an appropriate weight to vertices having the same color, we obtain that our simple branch-and-reduce algorithm runs in time $O^*(1.4658^n)$.

We close this section with some notation. All graphs in this paper are undirected and simple. Given a graph $G = (V, E)$, we use n to denote the number of vertices in G. For a vertex $v \in V$, we let $N_G(v)$ be the *open neighborhood* of v in G and $N_G[v] = N_G(v) \cup \{v\}$ be the *closed neighborhood* of v in G. Let $\deg(v)$ be the number of vertices in $N(v)$. A vertex v is called a degree-d vertex if $\deg(v) = d$. Let $\Delta(G) = \max_{v \in V}\{\deg(v)\}$ denote the maximum degree of G. For sets A and B, we use $A \uplus B$ to denote $A \cup B$ and $A \cap B = \emptyset$.

3 An $O^*(1.4658^n)$-time algorithm

In this section, we define two problems related to Max 1-bds and show that it can be reduced to them and solved in time $O^*(1.4658^n)$.

Max Constrained Bounded-Degree-1 Set (Max 1-cbds)

Input: A graph $G = (V, E)$ and a bounded-degree-1 set $S' \subseteq V$.

Output: A bounded-degree-1 set S of maximum size such that $S' \subseteq S$.

It is easy to see that if $S' = \emptyset$, the Max 1-cbds problem is the Max 1-bds problem. Given a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, let $G_i = G[V_i]$ where $V_i = \{v_1, v_2, \ldots, v_i\}$. Let $bds(G)$ denote the size of a maximum bounded-degree-1 set in G and $cbds(G, S')$ be the size of a maximum bounded-degree-1 set that contains S'. Then

$$bds(G) = \max_{1 \leq i \leq n} \{cbds(G_i, \{v_i\})\}.$$

Thus, the Max 1-bds problem can be solved by solving the Max 1-cbds problem.

Next, we define another problem related to Max 1-bds called Max Partially Independent Bounded-Degree-1 Set (Max 1-pibds).

Max 1-pibds

Input: A graph $G = (V, E)$ whose vertices are colored either white or orange; $V = W \uplus O$, where W and O are the sets of white vertices and orange vertices, respectively.

Output: A bounded-degree-1 set $S \subseteq V$ of maximum size in G such that every $v \in S \cap O$ is of degree-0 in $G[S]$.

Notice that if all vertices in G are white, it is equivalent to the Max 1-bds problem. If all vertices are orange, the Max 1-pibds problem is equivalent to the Maximum Independent Set problem.

Lemma 1. *If the* Max 1-pibds *problem can be solved in time $O^*(c^n)$, then the* Max 1-cbds *problem can be solved in time $O^*(c^n)$.*

Proof. Let $G = (V, E)$ and $S' \subseteq V$ be an input of the Max 1-cbds problem. Let S_1 be the set of vertices of degree one in $G[S']$ and let D be the set of vertices in $N_G(S')$ adjacent to at least two vertices in S'. Let S^* be an optimal solution of the Max 1-cbds problem. Notice that $S' \subseteq S^*$ and both S' and S^* are bounded-degree-1 sets. If $v \in S'$ is a degree-one vertex in $G[S']$, then all vertices in $N_G(v) \setminus S'$ are not in S^*. If there is a vertex x in $V \setminus S'$ having at least two neighbors in S' then x is not in S^*. We then construct $G' = (V' = W \uplus O, E')$ as the input of the Max 1-pibds problem according to G and S' by letting $V' = V \setminus (S' \cup D \cup N_G(S_1))$, $O = N_G(S') \setminus (N_G(S_1) \cup D)$, $W = V' \setminus O$. We see that in G every vertex in O has exactly a neighbor in S' and every vertex in W has no neighbor in S'. For $x, y \in O$, if in G they have common neighbors in S', add an edge between them in G'. Let $E' = \{(u, v) \mid u, v \in V', (u, v) \in E\} \cup \{(x, y) \mid x, y \in O, N_G(x) \cap N_G(y) \cap S' \neq \emptyset\}$. Let X be an optimal solution of the Max 1-pibds problem in G'. Since $X \cap O$ is an independent set, $X \cup S'$ is a bounded-degree-1 set, $|X \cup S'| \leq |S^*|$ where S^* is an optimal solution of the Max 1-cbds problem in G. Since S^* and S' are both bounded-degree-1 sets in G, $S^* \setminus S'$ is bounded-degree-1 in G. Notice that every $v \in S^* \cap N_G(S')$ is orange and $|N_G(v) \cap S'| = 1$. If $S^* \cap N_G(S')$ is not an independent set in G', then S^* is not bounded-degree-1, a contradiction. Thus $S^* \setminus S$ is a feasible solution of Max 1-pibds problem in G', $|S^* \setminus S'| \leq |X|$. Thus $X \cup S'$ is an optimal solution of the Max 1-cbds problem. Suppose that there exists an algorithm that solves the Max 1-pibds problem in time $O^*(c^n)$. Let $G = (V, E)$ and S' be an input of the Max 1-cbds problem. It takes polynomial time to construct $G' = (V', E')$ and it takes $O^*(c^{|V'|})$, $|V'| \leq |V|$, to find an optimal solution X of the Max 1-pibds problem in G'. Since $X \cup S'$ is an optimal solution of the Max 1-cbds problem, this shows that the Max 1-cbds problem can be solved in time $O^*(c^n)$ where n is the number of vertices in G. □

3.1 Some Observations

Let $G = (W \uplus O, E)$ be an input of the Max 1-pibds problem and S be an optimal solution of the Max 1-pibds problem in G. A vertex $v \in W \cup O$ is called *selected* if $v \in S$ and is called *discarded* if $v \notin S$. We give some observations of the Max 1-pibds problem.

Lemma 2 (Degree-1 Rule)**.** *If there exists a vertex $v \in W \cup O$ having degree one, then there exists an optimal solution that v is selected.*

Proof. Let u be the only neighbor of v. Assume that S be an optimal solution of the Max 1-pibds problem in G. If u is not selected, then $S \cup \{v\}$ is a solution of larger size, a contradiction. Suppose that $u \in S$ and $v \notin S$. We see that $S \setminus \{u\} \cup$

$\{v\}$ is an optimal solution of the same size. This completes the proof. □

Lemma 3 (Domination Rule)**.** *If there exist $v \in W \cup O$ and $u \in O$ satisfying $N[v] \subseteq N[u]$, then there exists an optimal solution that u is discarded.*

Proof. Suppose that S is an optimal solution of the MAX 1-PIBDS problem and $u \in S$. By definition all vertices in $N(u)$ are not in S. Since $N[v] \subseteq N[u]$, we obtain that $S \cup \{v\} \setminus \{u\}$ is also an optimal solution of the MAX 1-PIBDS problem. Thus, u can be always discarded. This completes the proof. □

Lemma 4 (Disconnected Rule)**.** *If G is disconnected and C is a connected component in G, then $S_C \cup S'$ is an optimal solution of the* MAX 1-PIBDS *problem in G where S_C and S' are optimal solutions of the* MAX 1-PIBDS *problem in $G[C]$ and $G[V \setminus C]$ respectively.*

Proof. Let S be an optimal solution of the MAX 1-PIBDS problem in G. It is easy to see that $S_C \cup S'$ is a feasible solution of the MAX 1-PIBDS problem in G, $|S_C \cup S'| \leq |S|$. Notice that $S \cap C$ and $S \cap (V \setminus C)$ are feasible solutions of the MAX 1-PIBDS problem in $G[C]$ and $G[V \setminus C]$, respectively. Since S_C and S' are optimal solutions of the MAX 1-PIBDS problem in $G[C]$ and $G[V \setminus C]$, $|S_C| \geq |S \cap C|$ and $|S'| \geq |S \cap (V \setminus C)|$. This shows that $S = S_C \cup S'$. □

Lemma 5 (Degree-2 Rule)**.** *If there exists a vertex $v \in W \cup O$ of degree two, $N(v) = \{x, y\}$, satisfying one of the following conditions*

1. *both x and y are white, or*
2. *at least one of x and y is orange and $(x, y) \notin E$,*

then either v is selected or v is discarded and both x, y are selected.

Proof. We show that if v is discarded then both x, y must be selected. Let S be an optimal solution of the MAX 1-PIBDS problem and $v \notin S$. Notice that if none of x, y is selected in S, then $S \cup \{v\}$ is a solution of larger size, a contradiction. If only one of x, y is selected in S, say x, then $S \cup \{v\} \setminus \{x\}$ is also an optimal solution. This completes the proof. □

Lemma 6 (White-orange Edge Rule 1)**.** *If there exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree at least three, $N(v) = \{u, x, y\}$, satisfying $N(v) \cap N(u) = \{x\}$, then either v is selected and u is discarded or v is discarded and y is selected.*

Proof. Let S be an optimal solution of the MAX 1-PIBDS problem. Since $u \in O$, if v is selected, then u must be discarded. Now we show that if v is discarded then y must be selected. Suppose that $v \notin S$ and $y \notin S$. Notice that one of u and x must be discarded since $u \in O$ and $(u, x) \in E$. If none of u, x, y is selected in S, then $S \cup \{v\}$ is a solution of larger size, a contradiction. If $u \in S$ and $x, y \notin S$, then $S \cup \{v\} \setminus \{u\}$ is also a optimal solution. If $x \in S$ and $u, y \notin S$, then $S \cup \{v\} \setminus \{x\}$ is also an optimal solution. Thus if $v \notin S$, either $y \in S$ or $y, x \in S$, or $y, u \in S$. This completes the proof. □

Lemma 7 (White-orange Edge Rule 2)**.** *If there exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree three, $N(v) = \{u, x, y\}$, $N(u) = \{v, z, w\}$, and $\{x, y\} \cap \{z, w\} = \emptyset$, then either v is selected and u is discarded, or u is selected and v, z, w are discarded, or both u, v are discarded and x, y, z, w are selected.*

Proof. Let S be an optimal solution of the MAX 1-PIBDS problem. Since $u \in O$ and $(u, v) \in E$, by definition at least one of u, v must be discarded. Either u is discard and v is selected, or v is discarded and u is selected, or both u, v are discarded. Notice that if u is selected in S, then v, z, w must be discarded. Claim that if $u, v \notin S$ then x, y, z, w are in S. If $x, y, z, w \notin S$, then $S \cup \{v\}$ is an optimal solution of larger size, a contradiction. If only one of x, y, z, w is selected in S, say x, then $S \cup \{v\} \setminus \{x\}$ is an optimal solution of the same size. If only one of x, y, z, w is selected in S, say z, then $S \cup \{v\}$ is an optimal solution of larger size. Suppose that two of x, y, z, w are selected in S. If $x, y \in S$ and $z, w \notin S$, then $S \cup \{u\}$ is a solution of larger size. If $z, w \in S$ and $x, y \notin S$, then $S \cup \{v\}$ is a solution of larger size. If $x, z \in S$ and $y, w \notin S$, then $S \cup \{v\} \setminus \{x\}$ is also an optimal solution. Suppose that only one of x, y, z, w is not in S. If $x \notin S$ and $y, z, w \in S$, then $S \cup \{v\} \setminus \{y\}$ is also an optimal solution. If $z \notin S$ and $x, y, w \in S$, then $S \cup \{u\} \setminus \{w\}$ is also an optimal solution. Thus if both $u, v \notin S$, all x, y, z, w must be selected in S. This completes the proof. □

Lemma 8 (White-orange Edge Rule 3)**.** *If there exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree at least four, $N(v) = \{u, x, y\}$, satisfying $N(v) \cap N(u) = \emptyset$, then either v is selected and u is discarded, or u is selected and all*

vertices in $N(u)$ are discarded, or both u, v are discarded and x, y are selected.

Proof. Let S be an optimal solution of the MAX 1-PIBDS problem. Since $u \in O$ and $(u, v) \in E$, by definition at least one of u, v must be discarded. Either u is discard and v is selected, or v is discarded and u is selected, or both u, v are discarded. Notice that if u is selected in S, then all vertices in $N(u)$ must be discarded. Claim that if $u, v \notin S$ then x, y are in S. If $x, y \notin S$, then $S \cup \{v\}$ is an optimal solution of larger size, a contradiction. If only one of x, y is selected in S, say x, then $S \cup \{v\} \setminus \{x\}$ is an optimal solution of the same size. Thus if both $u, v \notin S$, both x, y must be selected in S. This completes the proof. □

3.2 The algorithm for MAX 1-PIBDS

We give a branch-and-reduce algorithm for the MAX 1-PIBDS problem. The algorithm consists of a series of reduction rules and branching rules. The description of the algorithm consists of a sequence of cases and subcases. To avoid a confusing nesting of if-then-else statements let us use the following convention: The first case which applies is used in the algorithm. Thus, inside a given case, the hypotheses of all previous cases are assumed to be false. Given an input graph $G = (W \uplus O, E)$ of the MAX 1-PIBDS problem, the following algorithm computes a bounded-degree-1 set S of maximum size such that all vertices in $S \cap O$ are of degree-0 in $G[S]$.

Note that the algorithm removes v from G after it decides to *discard* v. If the algorithm *selects an orange vertex* v in S, then the algorithm obtains a new input graph $G' = G[V \setminus N[v]]$ of the MAX 1-PIBDS problem after selecting v. If the algorithm *selects a white vertex* v in S, then the algorithm executes the following four steps, called *selecting white steps*, to obtain a new input graph G' of the MAX 1-PIBDS problem after selecting v.

1. Remove all orange neighbors of v from G.
2. Add edges between any two $x, y \in N(v) \cap W$.
3. Recolor all vertices in $N(v) \cap W$ orange.
4. Remove v from G.

Boundary condition: If all vertices in G are orange, then by definition the optimal solution of the MAX PI-1-BDS problem in G is a maximum independent set in G. The algorithm calls an exact algorithm for the MAXIMUM INDEPENDENT SET problem to solve the remaining problem.

Reduction rules:

R1. **Degree-1 Rule.** If there exists a vertex $v \in W \cup O$ having degree one, $N(v) = \{u\}$ then according Lemma 2, the algorithm selects v.

R2. **Domination Rule.** If there exist $v \in W \cup O$ and $u \in O$ satisfying $N[v] \subseteq N[u]$, then according to Lemma 3, the algorithm discards u.

R3. **Disconnected Rule.** If G is disconnected and C be a connected component in G, according to Lemma 4 the algorithm solves the problem recursively in $G[C]$ and $G[V \setminus C]$.

Branching rules:

B1. **Degree-2 Rule.** If there exists a vertex $v \in W \cup O$ of degree two, $N(v) = \{x, y\}$, satisfying one of the following conditions

   (a) both x and y are white, or

   (b) at least one of x and y is orange and $(x, y) \notin E$,

   and the degree-sum $\deg(x)+\deg(y)$ of the two neighbors of v is maximum among all degree two vertices in G, then according Lemma 5 the algorithm either (i) selects v or (ii) discards v and selects both x, y.

B2. **White Vertex Rule.** If there exists a vertex $v \in W$ of degree at least four, then the algorithm either (i) selects v or (ii) discards v.

B3. **White-orange Edge Rule 1.** If there exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree at least three, $N(v) = \{u, x, y\}$, satisfying $N(v) \cap N(u) = \{x\}$, then according to Lemma 6 the algorithm either (i) selects v and discards u or (ii) discards v and selects y.

B4. **White-orange Edge Rule 2.** If there exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree three, $N(v) = \{u, x, y\}$ and $N(u) = \{v, z, w\}$, satisfying $N(v) \cap N(u) = \emptyset$, then according to Lemma 7 the algorithm either (i) selects v and discards u, or (ii) selects u and discards v, z, w, or (iii) discards both u, v and selects x, y, z, w.

B5. **White-orange Edge Rule 3.** If there exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree at least four, $N(v) = \{u, x, y\}$, satisfying $N(v) \cap N(u) = \emptyset$, then according to Lemma 8 the algorithm either

(i) selects v and discards u, or (ii) selects u and discards all vertices in $N(u)$, or (iii) discards both u, v and selects x, y.

B6. **3-Regular Rule.** G is a 3-regular graph and all vertices are white. Let v be a vertex in G. The algorithm either (i) selects v or (ii) discards v.

If we use simple analysis to see the running time of the above algorithm, due to the White Vertex Rule and the 3-Regular Rule, one would say that the running time of the algorithm is $O^*(2^n)$. In the next section, by using the measure-and-conquer approach, we analyze the running time of the algorithm is $O^*(1.4658^n)$.

3.3 The analysis of the exact algorithm

Let every orange vertex have the same weight w_o, $0 < w_o \leq 1$ and let every white vertex have the same weight one. The total weight of graph $G = (W \uplus O, E)$ is $w_G = w_o \cdot |O| + |W|$. Let $\Delta w_o = 1 - w_o$. The size of the input is the weight w_G of the input graph. When branching rule b is applied, the current instance is branched into $r \geq 2$ instances of size at most $w_G - t_1, w_G - t_2, \ldots, w_G - t_r$. Note that $w_G \geq t_i$ for $i = 1, 2, \ldots, r$. We call $\mathbf{b} = (t_1, t_2, \ldots, t_r)$ the *branching vector* of branching rule b. This can be formulated in a linear recurrence

$$T(w_G) = T(w_G - t_1) + T(w_G - t_2) + \cdots + T(w_G - t_r).$$

By using well-known standard techniques to solve linear recurrence, the base solution of above linear recurrence is of the form α^{w_G} for some complex number α. We call α is the *branching number* of rule b and $O^*(\alpha^{w_G}) = O^*(\alpha^n)$ is the worst case running time of this rule since $w_G \leq n$.

Boundary Condition: Since all vertices in G are orange, we apply Robson's algorithm [25] to find a maximum independent set in G. The running time for graphs applying this rule is $O^*(2^{0.276 \times \frac{w_G}{w_o}}) = O^*(2^{0.276 \times \frac{n}{w_o}})$.

All reduction rules are done in polynomial time. Hence we can focus on branching rules in the analysis. We will analyze all branching rules one by one.

Degree-2 Rule. If there exists a vertex $v \in W \cup O$ of degree two, $N(v) = \{x, y\}$, satisfying one of the following conditions

1. both x and y are white, or
2. at least one of x and y is orange and $(x, y) \notin E$,

and the degree-sum $\deg(x) + \deg(y)$ of the two neighbors of v is maximum among all degree two vertices in G, then the algorithm either (i) selects v or (ii) discards v and selects both x, y. We have

$$T(w_G) = T(w_G - t_1) + T(w_G - t_2).$$

There are six cases listed in Fig 1.

1. If $v, x, y \in W$ as Fig. 1 (a), then $t_1 = 1 + 2 \cdot \Delta w_o$ and $t_2 \geq 3$.
2. If $v \in W$ and one of x, y is orange as Fig. 1 (b), then $t_1 = 1 + w_o + \Delta w_o = 2$ and $t_2 \geq 2 + w_o$.
3. If $v \in W$ and both x, y are orange as Fig. 1 (c), then $t_1 = 1 + 2 \cdot w_o$ and $t_2 \geq 1 + 2 \cdot w_o$.
4. If $v \in O$ and both x, y are white as Fig. 1 (d), then $t_1 = 2 + w_o$ and $t_2 \geq 2 + w_o$.
5. If $v \in O$ and one of x, y is white as Fig. 1 (e), then $t_1 = 1 + 2 \cdot w_o$ and $t_2 \geq 1 + 2 \cdot w_o$.
6. If $v, x, y \in O$ as Fig. 1 (f), then $t_1 = 3 \cdot w_o$ and $t_2 \geq 5 \cdot w_o$. Since the degree-sum $\deg(x) + \deg(y)$ is maximum among all degree-2 vertices, the number of vertices of distance two from v is at least two. Otherwise, we obtain a cycle consisting of all orange vertices such that either the **Disconnected Rule** can be applied or it satisfies the **Boundary Condition**.

White Vertex Rule. There exists a vertex $v \in W$ of degree at least four. The algorithm either (i) selects v or (ii) discards v. Suppose that v has p white neighbors and q orange neighbors. We have

$$T(w_G) = T(w_G - t_1) + T(w_G - t_2)$$

where $t_1 = 1$ and $t_2 = 1 + q \cdot w_o + p \cdot \Delta w_o$. It is easy to see that the worst case happens when the degree of v is four as Fig. 2. Suppose that $\deg(v) = 4$, we have the following cases.

1. All vertices in $N(v)$ are white. Then $t_2 = 1 + 4 \cdot \Delta w_o$.
2. There is only an orange vertex in $N(v)$. Then $t_2 = 1 + w_o + 3 \cdot \Delta w_o$.
3. There are two orange vertices in $N(v)$. Then $t_2 = 1 + 2 \cdot w_o + 2 \cdot \Delta w_o$.

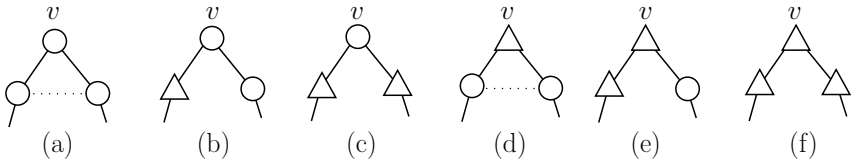

Figure 1: Cases of **Degree-2 Rule**. We use circles and triangles to denote white vertices and orange vertices, respectively. A dotted line denotes two vertices are possibly adjacent or possibly not adjacent.

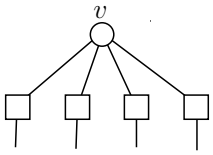

Figure 2: Cases of **White Vertex Rule**. We use a square to denote that the vertex is a white vertex or an orange vertex.

4. There is only a white vertex in $N(v)$. Then $t_2 = 1 + 3 \cdot w_o + \Delta w_o$.

5. All vertices in $N(v)$ are orange. Then $t_2 = 1 + 4 \cdot w_o$.

Notice that if all reduction rules, **Degree-2 Rule**, and **White Vertex Rule** can not be applied, then all orange vertices have degree at least three and all white vertices have degree three.

White-orange Edge Rule 1. There exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree at least three, $N(v) = \{u, x, y\}$, satisfying $N(v) \cap N(u) = \{x\}$ as Fig. 3(a), then the algorithm either (i) selects v and discards u or (ii) discards v and selects y. Since there are two branches, we see that

$$T(w_G) = T(w_G - t_1) + T(w_G - t_2).$$

There are four cases.

1. If $x, y \in W$, then $t_1 = 1 + w_o + 2 \cdot \Delta w_o$ and $t_2 \geq 2 + 2 \cdot \min\{w_o, \Delta w_o\}$.

2. If $x \in W$ and $y \in O$, then $t_1 = 2 + w_o$ and $t_2 \geq 1 + 3 \cdot w_o$.

3. If $x \in O$ and $y \in W$, then $t_1 = 2 + w_o$ and $t_2 \geq 2 + 2 \cdot \min\{w_o, \Delta w_o\}$.

4. If $x, y \in O$, then $t_1 = 1 + 3 \cdot w_o$ and $t_2 \geq 1 + 3 \cdot w_o$.

White-orange Edge Rule 2. There exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree three, $N(v) = \{u, x, y\}$ and $N(u) = \{v, z, w\}$ as Fig. 3 (b). The algorithm either (i) selects v and discards u, or (ii) selects u and discards v, z, w, or (iii) discards both u, v and selects x, y, z, w. Since there are three branches, we see that

$$T(w_G) = T(w_G - t_1) + T(w_G - t_2) + T(w_G - t_3).$$

Notice that $N(\{x, y, z, w\}) \setminus \{u, v\} \neq \emptyset$, otherwise G is a small component having only six vertices. There are 9 cases.

1. If $x, y, z, w \in W$, then $t_1 = 2 + \Delta w_o$, $t_2 = 3 + w_o$, and $t_3 \geq 5 + w_o$.

2. If one of x, y is orange and $z, w \in W$, then $t_1 = 2 + w_o$, $t_2 = 3 + w_o$, and $t_3 \geq 4 + 2 \cdot w_o + \min\{w_o, \Delta w_o\}$.

3. If $x, y \in O$ and $z, w \in W$, then $t_1 = 1 + 3 \cdot w_o$, $t_2 = 3 + w_o$, and $t_3 \geq 3 + 3 \cdot w_o + \min\{w_o, \Delta w_o\}$.

4. If $x, y \in W$ and one of z, w is orange, $t_1 = 2 + \Delta w_o$, $t_2 = 2 + 2 \cdot w_o$, and $t_3 \geq 4 + 2 \cdot w_o + \min\{w_o, \Delta w_o\}$.

5. If one of x, y is orange and one of z, w is orange, $t_1 = 2 + w_o$, $t_2 = 2 + 2 \cdot w_o$, and $t_3 \geq 3 + 3 \cdot w_o + \min\{w_o, \Delta w_o\}$.

6. If $x, y \in O$ and one of z, w is orange, $t_1 = 1 + 3 \cdot w_o$, $t_2 = 2 + 2 \cdot w_o$, and $t_3 \geq 2 + 4 \cdot w_o + \min\{w_o, \Delta w_o\}$.

7. If $x, y \in W$ and $z, w \in O$, $t_1 = 2 + \Delta w_o$, $t_2 = 1 + 3 \cdot w_o$, and $t_3 \geq 3 + 3 \cdot w_o + \min\{w_o, \Delta w_o\}$.

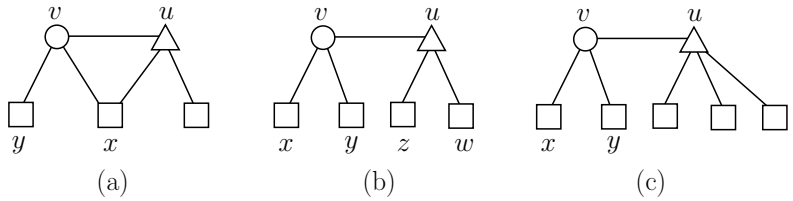

Figure 3: (a) Cases of **White-orange Edge Rule 1** (b) Cases of **White-orange Edge Rule 2** (c) Cases of **White-orange Edge Rule 3**. A circle denotes a white vertex, a triangle denotes an orange vertex, and a square denotes the vertex possibly white or orange.

8. If one of x, y is orange and $z, w \in O$, $t_1 = 2 + w_o$, $t_2 = 1 + 3 \cdot w_o$, and $t_3 \geq 2 + 4 \cdot w_o + \min\{w_o, \Delta w_o\}$.

9. If $x, y, z, w \in O$, $t_1 = 1+3 \cdot w_o$, $t_2 = 1+3 \cdot w_o$, and $t_3 \geq 1 + 6 \cdot w_o$.

White-orange Edge Rule 3. There exist a vertex $v \in W$ of degree three and a vertex $u \in O$ of degree at least four, $N(v) = \{u, x, y\}$, satisfying $N(v) \cap N(u) = \emptyset$ as Fig. 3 (c). The algorithm either (i) selects v and discards u, or (ii) selects u and discards all vertices in $N(u)$, or (iii) discards both u, v and selects x, y.

Since there are three branches, we see that

$$T(w_G) = T(w_G - t_1) + T(w_G - t_2) + T(w_G - t_3).$$

Notice that if all orange vertices adjacent to v have degree at least four. There are three cases.

1. If $x, y \in W$, then $t_1 = 2+\Delta w_o$, $t_2 \geq 1+4 \cdot w_o$, $t_3 \geq 3 + w_o + \min\{w_o, \Delta w_o\}$.

2. If one of x, y is orange, then $t_1 = 2 + w_o$, $t_2 \geq 1+4 \cdot w_o$, $t_3 \geq 2+2 \cdot w_o + \min\{w_o, \Delta w_o\}$.

3. If $x, y \in O$, then $t_1 = 1+3 \cdot w_o$, $t_2 \geq 1+4 \cdot w_o$, $t_3 \geq 1 + 6 \cdot w_o$.

Notice that there is at most one 3-regular graph that contains only white vertices assigned to a node of the search tree from the root to a leaf since every instance having no orange vertices generated by the algorithm is an induced subgraph of the original problem. Thus the **3-Regular Rule** applied to those instances can only increase the number of leaves by a multiplicative constant. We may neglect the time analysis of the **3-Regular Rule**.

With the best choice of weight $w_o = 0.50035$, we obtain that the worst branching rule is the **White Vertex Rule**. The worst case happens when the **White Vertex Rule** is applied on a degree-4 white vertex having no orange neighbors. Its corresponding branching vector is $(1, 1 + 4 \cdot \Delta w_o) = (1, 2.9986)$ and branching number is 1.4658. For an instance satisfying the boundary condition, the algorithm applies Robson's algorithm [25] to find a maximum independent set and it takes

$$\begin{aligned} O^*(2^{0.276 \times \frac{w_G}{w_o}}) &= O^*(2^{0.276 \times \frac{n}{w_o}}) \\ &= O^*(2^{0.276 \times \frac{n}{0.50035}}) \\ &= O^*(1.4658^n) \end{aligned}$$

time to solve the problem. Thus, the running time of the algorithm is $O^*(1.4658^n)$.

Theorem 1. *The* Max 1-pibds *problem can be solved in time* $O^*(1.4658^n)$.

Corollary 1. *The* Max 1-bds *problem can be solved in time* $O^*(1.4658^n)$.

Remark 1. *Chang and Hung [11] showed that if the* Max 1-bds *problem can be solved in time* $O^*(c^n)$, *then the* Max 2-plex *problem can be solved in time* $O^*(c^{\Delta(G)})$ *where* G *is the input graph of the* Max 2-plex *problem. Since the* Max 1-bds *problem can be solved in time* $O^*(1.4658^n)$, *the* Max 2-plex *problem can be solved in time* $O^*(1.4658^{\Delta(G)})$.

Remark 2. *It was shown in [8, 11] that if the* Max 1-bds *problem can be solved in time* $O^*(\gamma^n)$ *time, then there exists an* p/q*-approximation algorithm that runs in time* $O^*(\gamma^{\frac{p}{q} \cdot n})$ *for the* Max 1-bds *problem for any two positive integers* $p < q$. *Since the* Max 1-bds *problem can be solved in time* $O^*(1.4658^n)$, *we see that there exists an* p/q*-approaximation algorithm that runs in time* $O^*(1.4658^{\frac{p}{q} \cdot n})$ *for the* Max 1-bds *problem for any two positive integers* $p < q$.

4 Conclusions

In this paper, we use simple branching strategies having at most three branches and a 2-coloring approach to design a branch-and-reduce

algorithm for the MAX 1-BDS problem. By applying measure-and-conquer analysis, we obtain that the running time of our branch-and-reduce algorithm is $O^*(1.4658^n)$. Chen *et al.* [12] gave a fixed-parameter algorithm running in time $O^*(3.24^k)$ for the MIN 2-BDD problem where k is the the input parameter that indicates the number of vertices being deleted to obtain a bounded-degree-2 set. The open problem is whether the idea in this paper can be extended to solve the MIN 2-BDD problem in time $O^*(c^k)$, $c < 3.24$.

References

[1] B. Balasundaram, Cohesive subgroup model for graph-based text mining, *Proceedings of the 2008 IEEE Conference on Automation Science and Engineering*, pp. 989–994, 2008.

[2] B. Balasundaram, S. Chandramouli, and S. Trukhanov, Approximation algorithms for finding and partitioning unit-disk graphs into co-k-plexes, *Optimization Letters* **4** (2010), pp. 311–320.

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

[4] R. Beigel and D. Eppstein, 3-coloring in time $O(1.3289^n)$, *Journal of Algorithms*, **54** (2005), pp. 168–204.

[5] N. Betzler, R. Bredereck, R. Niedermeier, and J. Uhlmann, On bounded-degree vertex deletion parameterized by treewidth, *Discrete Applied Mathematics* **160** (2012), pp. 53–60.

[6] R. van Bevern, H. Moser, and R. Niedermeier, Approximation and tidying–a problem kernel for s-plex cluster vertex deletion, *Algorithmica* **62** (2012), pp. 930–950.

[7] D. Binkele-Raible, *Amortized Analysis of Exponential Time- and Parameterized Algorithms: Measure & Conquer and Reference Search Trees*, PhD Thesis, Trier University, 2010.

[8] N. Bourgeois, B. Escoffier, V. Th. Paschos, Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms, *Discrete Applied Mathematics* **159** (2011), pp. 1954–1970.

[9] M.-S. Chang, L.-J. Hung, and P.-C. Su, Exact and fixed-parameter algorithms for problems related to 2-plex, *Proceedings of ICSEC 2011*, pp. 203–208, 2011.

[10] M.-S. Chang, L.-J. Hung, and P.-C. Su, Measure and conquer: analysis of a branch-and-reduce algorithm for the maximum bounded-degree-1 set problem, *Proceedings of the 29th Workshop on Combinatorial Mathematics and Computation Theory*, pp. 136–145, 2012.

[11] M.-S. Chang and L.-J. Hung, Moderately exponential time approximation algorithms for the maximum bounded-degree-1 set problem, in *Proceedings of the 30th Workshop on on Combinatorial Mathematics and Computation Theory*, pp. 23–30, 2013.

[12] Z. Z. Chen, M. Fellows, B. Fu, H. Jiang, Y. Liu, L. Wand, and B. Zhu, A linear kernel for co-path/cycle packing, *Proceedings of AAIM 2010*, LNCS 6124, pp. 90–102.

[13] D. Eppstein, Quasiconvex analysis of backtracking algorithms, *Proceedings of SODA 2004*, pp. 781–790, 2004.

[14] D. Eppstein, Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms, *ACM Transactions on Algorithms* **2** (2006), pp. 492–509.

[15] M. R. Fellows, J. Guo, H. Moser, and R. Niedermeier, A generalization of Nemhauser and Trotter's local optimization theorem, *Journal of Computer and System Sciences* **77** (2011), pp. 1141–1158.

[16] F. V. Fomin, F. Grandoni, and D. Kratsch, Measure and conquer: domination – a case study, *Proceedings of ICALP 2005*, LNCS 3580 (2005), pp. 191–203.

[17] F. V. Fomin, F. Grandoni, and D. Kratsch, Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm, *Proceedings of SODA 2006*, pp. 18–25, 2006.

[18] F. V. Fomin, F. Grandoni, and D. Kratsch, A measure & conquer approach for the analysis of exact algorithms, *Journal of the ACM* **56** (2009) Article No. 25.

[19] F. Fomin and D. Kratsch, *Exact Exponential Algorithms*, Springer, 2010.

[20] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann, A more relaxed model for graph-based data clustering: s-plex cluster editing, *SIAM Journal on Discrete Mathematics* **24** (2010), pp. 1662–1683.

[21] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier, Isolation concepts for efficiently enumerating dense subgraphs, *Theoretical Computer Science* **410** (2009), pp. 3640–3654.

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

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

[24] N. Nishmura, P. Ragde, and D. M. Thilikos, Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover, *Discrete Applied Mathematics* **152** (2005), pp. 229–245.

[25] J. M. Robson, Algorithms for maximum independent sets, *Journal of Algorithms* **7** (1986), pp. 425–440.

[26] S. B. Seidman and B. L. Foster, A graph-theoretic generalization of the clique concept, *The Journal of Mathematical Sociology* **6** (1978), pp. 139–154.

[27] S. Trukhannov, *Novel approaches for solving large-scale optimization problems on graphs*, PhD Thesis, A&M University, Texas, 2008.

[28] B. Wu and X. Pei, A parallel algorithm for enumerating all the maximal k-plexes, *Proceedings of PAKDD 2007*, LNAI 4819 (2007), pp. 476–483.