




# Exact Algorithms for Maximum Weighted Independent Set on Sparse Graphs (Extended Abstract)

Sen Huang<sup>1</sup>, Mingyu Xiao<sup>1</sup> , and Xiaoyu Chen<sup>2</sup>

<sup>1</sup> University of Electronic Science and Technology of China, Chengdu, China

<sup>2</sup> Nanjing University, Nanjing, China

**Abstract.** The maximum independent set problem is one of the most important problems in graph algorithms and has been extensively studied in the line of research on the worst-case analysis of exact algorithms for NP-hard problems. In the weighted version, each vertex in the graph is associated with a weight and we are going to find an independent set of maximum total vertex weight. In this paper, we design several reduction rules and a fast exact algorithm for the maximum weighted independent set problem, and use the measure-and-conquer technique to analyze the running time bound of the algorithm. Our algorithm works on general weighted graphs and it has a good running time bound on sparse graphs. If the graph has an average degree at most 3, our algorithm runs in  $O^*(1.1443^n)$  time and polynomial space, improving previous running time bounds for the problem in cubic graphs using polynomial space.

**Keywords:** Maximum weighted independent set · Exact algorithms · Measure-and-Conquer · Graph algorithms · Reduction rules

## 1 Introduction

The MAXIMUM INDEPENDENT SET problem on unweighted graphs belongs to the first batch of 21 NP-hard problems proved by Karp [12]. In the line of research on the worst-case analysis of exact algorithms for NP-hard problems, MAXIMUM INDEPENDENT SET, as one of the most fundamental problems, is used to test the efficiency of new techniques of exact algorithms. There is a long list of contributions to exact algorithms for MAXIMUM INDEPENDENT SET in unweighted graphs [2, 8, 11, 13, 16, 17]. Now it can be solved in  $O^*(1.1996^n)$  time and polynomial space [21]. If the maximum degree of the graph is 3, the running time bound can be improved to  $O^*(1.0836^n)$  [20].

In this paper, we will consider the weighted version of MAXIMUM INDEPENDENT SET, called MAXIMUM WEIGHTED INDEPENDENT SET, where each vertex in the graph has a nonnegative weight and we are asked to find an independent

---

The work is supported by the National Natural Science Foundation of China, under grant 61972070.

© Springer Nature Switzerland AG 2021

C.-Y. Chen et al. (Eds.): COCOON 2021, LNCS 13025, pp. 617–628, 2021.

[https://doi.org/10.1007/978-3-030-89543-3\\_51](https://doi.org/10.1007/978-3-030-89543-3_51)

set with the maximum total vertex weight. It has many applications in various real-world problems. For example, the dynamic map labeling problem [1, 15] can be naturally encoded as MAXIMUM WEIGHTED INDEPENDENT SET. Some experimental algorithms, such as the algorithms in [14, 19] have been developed to solve instances from real world and known benchmarks. These algorithms run fast even on large scale sparse instances but lack running time analysis. For running time bounds, most known results were obtained via two counting problems: COUNTING MAXIMUM WEIGHTED INDEPENDENT SET and COUNTING WEIGHTED 2-SAT. Most of these counting algorithms can also list out all independent sets and then we can find a maximum one by increasing only a polynomial factor. Dahllöf et al. [4] presented an  $O^*(1.3247^n)$ -time algorithm for COUNTING MAXIMUM WEIGHTED INDEPENDENT SET. Later, the running time bound was improved to  $O^*(1.2431^n)$  by Fomin et al. [6]. COUNTING MAXIMUM WEIGHTED INDEPENDENT SET can also be reduced to COUNTING WEIGHTED 2-SAT, preserving the exponential part of the running time. For COUNTING WEIGHTED 2-SAT, the running time bound was improved from  $O^*(1.2561^n)$  [5] to  $O^*(1.2461^n)$  [9] and then to  $O^*(1.2377^n)$  [18]. Wahlström [18] also showed that the running time bound could be further improved to  $O^*(1.1499^n)$  and  $O^*(1.2117^n)$  if the maximum degree of the variables or the vertices in the graph is bounded by 3 and 4, respectively. Most of the above algorithms use only polynomial space. If exponential space is allowed, dynamic programming algorithms based on tree decompositions, by using the treewidth bound on degree-3 graphs in [7], may achieve a better running time bound  $O^*(1.1225^n)$ .

In this paper, we will focus on exact algorithms specifying for MAXIMUM WEIGHTED INDEPENDENT SET. We develop structural properties and design reduction rules for the problem, and then design a fast exact algorithm based on them. By using the measure-and-conquer technique, we can prove that the algorithm runs in  $O^*(1.1443^{(0.624x-0.872)n})$  time and polynomial space, where  $x$  is the average degree of the graph. For some sparse graphs, our result beats the known bounds. For example, the running time bound of our algorithm in graphs with the average degree at most three is  $O^*(1.1443^n)$ , which improves the previously known bound of  $O^*(1.1499^n)$  using polynomial space [18]. For graphs with the average degree at most 3.68, the running time of our algorithm is strictly better than the running time bound  $O^*(1.2117^n)$  for MAXIMUM WEIGHTED INDEPENDENT SET in degree-4 graphs [18].

Due to the limited space, the proofs of lemmas marked with (\*) were omitted, which can be found in the full version of this paper [10].

## 2 Preliminaries

Let  $G = (V, E, w)$  denote an undirected vertex-weighted graph with  $|V| = n$  vertices and  $|E| = m$  edges, where each vertex  $v \in V$  is associated with a positive weight  $w(v)$ . Although our graphs are undirected, we may use an arc to denote the relation of the weights of the two endpoints of an edge. An arc  $\vec{uv}$  from vertex  $u$  to vertex  $v$  means that there is an edge between  $u$  and  $v$  and it holds that  $w(u) \geq w(v)$ .

Let  $V' \subseteq V$  be a vertex subset. We let  $w(V') = \sum_{v \in V'} w(v)$ , and  $N(V')$  denote the set of vertices not in  $V'$  but adjacent to at least one vertex in  $V'$ . We also denote  $d(V') = |N(V')|$  and  $N[V'] = N(V') \cup V'$ . We use  $G[V']$  to denote the subgraph of  $G$  induced by  $V'$  and use  $G - V'$  to denote  $G[V \setminus V']$ . For a graph  $G'$ , we use  $\mathcal{C}(G')$  to denote the set of connected components of  $G'$ . A *chain* is an induced path such that the degree of each vertex except the two endpoints of the path is exactly 2. One vertex is a *chain-neighbor* of another vertex if they are connected by a chain. For a vertex-weighted graph, a *maximum weighted independent set* is an independent set  $S$  such that  $w(S)$  is maximized among all independent sets in the graph. We use  $S(G)$  to denote a maximum weighted independent set in graph  $G$  and  $\alpha(G)$  to denote the total vertex weight of  $S(G)$ . Our problem is defined below.

**MAXIMUM WEIGHTED INDEPENDENT SET (MWIS)**

**Input:** An undirected vertex-weighted graph  $G = (V, E, w)$ .

**Output:** the weight of a maximum weighted independent set in  $G$ , i.e.,  $\alpha(G)$ .

**2.1 Measure-and-Conquer**

Our algorithm is a branch-and-search algorithm. We will use a measure to evaluate the time complexity. For a branching operation, if the measure decreases by at least  $a_i$  in the  $i$ -th substance, then we say the *branching vector* of the operation is  $[a_1, a_2, \dots, a_l]$ . The largest root of the function  $f(x) = 1 - \sum_{i=1}^l x^{-a_i}$  is called the *branching factor* of the recurrence.

The measure-and-conquer technique, introduced in [8], is a powerful tool to analyze branch-and-search algorithms. The main idea is to use a non-traditional measure to evaluate the running time. Let  $n_i$  denote the number of vertices of degree  $i$  in the graph. We associate a cost  $\delta_i \geq 0$  for each degree- $i$  vertex in the graph. Our measure  $p$  is defined as follows:

$$p := \sum_{i=0}^n n_i \delta_i. \tag{1}$$

The cost  $\delta_i$  in this paper is given by

$$\delta_i = \begin{cases} 0 & \text{if } i \leq 1 \\ 0.376 & \text{if } i = 2 \\ 1 & \text{if } i = 3 \\ 1 + 0.624(i - 3) & \text{if } i \geq 4. \end{cases} \tag{2}$$

We also define  $\delta_i^{<-k>} := \delta_i - \delta_{i-k}$  for each integer  $k \geq 0$ . In our analysis, we may use the following inequalities and equalities to simplify some arguments:  $\delta_i^{<-1>} = \delta_3^{<-1>}$  for  $i \geq 4$ ;  $\delta_3 \geq 2.5\delta_2$ ;  $3\delta_2 \geq \delta_3$ .

With the above setting, we know that when  $p \leq 0$ , the instance contains only degree-0 and degree-1 vertices and can be solved directly. We will design

an algorithm with running time bound  $O^*(c^p)$  for some constant  $c$ . If the initial graph has degree at most 3, then we have that  $p \leq n$  and then the running time bound of the algorithm is  $O^*(c^n)$ . In general, if we have  $p \leq f(n)$  for some function  $f$  on  $n$ , then we can get a running time bound of  $O^*(c^{f(n)})$ . We have the following lemma for the relation between  $p$  and  $n$ .

**Lemma 1.** (\*) *For a graph of  $n$  vertices, if the average degree of the graph is at most  $x$ , then the measure  $p$  of the graph is at most  $(0.624x - 0.872)n$ .*

### 3 Reduction Rules

We first introduce reduction rules that will be applied to reduce the instance directly by eliminating some local structures of the graph. Some reduction rules may include a set  $S$  of vertices in the solution set directly. We use  $M_c$  to store the weight of the vertices that have been included in the solution set. When a set  $S$  of vertices is included in the solution set, we will remove  $N[S]$  from the graph and update  $M_c$  by adding  $w(S)$ .

#### 3.1 General Reductions for Some Special Structures

We use several reduction rules based on unconfined vertices, twins, vertices with a clique neighborhood, and heavy vertices. Some of these reduction rules were introduced in [14] and [19].

**Unconfined Vertices.** A vertex  $v$  in  $G$  is called *removable* if  $\alpha(G) = \alpha(G - v)$ , i.e., there is a maximum weighted independent set in  $G$  that does not contain  $v$ . We can say that a vertex  $v$  is removable if a contradiction is obtained from the assumption that every maximum weighted independent set in  $G$  contains  $v$ . A sufficient condition for a vertex to be removable in unweighted graphs has been studied in [20]. We extend this concept to weighted graphs.

For an independent set  $S$  of  $G$ , a vertex  $u \in N(S)$  is called a *child* of  $S$  if  $w(u) \geq w(S \cap N(u))$ . A child  $u$  is called an *extending child* if it holds that  $|N(u) \setminus N[S]| = 1$ , and the only vertex  $v \in N(u) \setminus N[S]$  is called a *satellite* of  $S$ .

**Lemma 2.** (\*) *Let  $S$  be an independent set that is contained in any maximum weighted independent set in  $G$ . Then every maximum weighted independent set contains at least one vertex in  $N(u) \setminus N[S]$  for each child  $u$  of  $S$ .*

We introduce a method based on Lemma 2 to find possible removable vertices. Let  $v$  be an arbitrary vertex in the graph. After starting with  $S := \{v\}$ , we repeat (1) until (2) or (3) holds:

- (1) If  $S$  has some extending child in  $N(S)$ , then let  $S'$  be the set of satellites. Update  $S$  by letting  $S := S \cup S'$ .
- (2) If  $S$  is not an independent set or there is a child  $u$  such that  $N(u) \setminus N[S] = \emptyset$ , then halt and conclude that  $v$  is *unconfined*.
- (3) If  $|N(u) \setminus N[S]| \geq 2$  for all children  $u \in N(S)$ , then halt and return  $S_v = S$ .

Obviously, the procedure can be executed in polynomial time for any starting set  $S$  of a vertex. If the procedure halts in (2), we say vertex  $v$  *unconfined*. If the procedure halts in (3), then we say that the set  $S_v$  returned in (3) *confines* vertex  $v$  and vertex  $v$  is also called *confined*. Note that the set  $S_v$  confining  $v$  is uniquely determined by the procedure with starting set  $S := \{v\}$ . It is easy to observe the following lemma.

**Lemma 3.** *(\*) Any unconfined vertex is removable.*

**Reduction Rule 1 (R1).** *If a vertex  $v$  is unconfined, remove  $v$  from  $G$ .*

**Twins.** A set  $A = \{u, v\}$  of two non-adjacent vertices is called a *twin* if they have the same neighbor set, i.e.,  $N(u) = N(v)$ .

**Reduction Rule 2 (R2)** [14]. *If there is a twin  $A = \{u, v\}$ , delete  $v$  and update the weight of  $u$  by letting  $w(u) := w(u) + w(v)$ .*

**Clique Neighborhood.** A vertex  $v$  has a *clique neighborhood* if the graph  $G[N(v)]$  induced by the open neighbor set of  $v$  is a clique, which was introduced as isolated vertices in [14].

**Reduction Rule 3 (R3)** [14]. *If there is a vertex  $v$  having a clique neighborhood and  $w(v) < w(u)$  holds for all  $u \in N(v)$ , then remove  $v$  from the graph, update the weight  $w(u) := w(u) - w(v)$  for all  $u \in N_G(v)$ , and add  $w(v)$  to  $M_c$ .*

**Heavy Vertices.** A vertex  $v$  is called a *heavy vertex* if its weight is not less the weight of the maximum weighted independent set in subgraph induced by the open neighborhood of it, i.e.,  $w(v) \geq \alpha(G[N(v)])$ .

**Reduction Rule 4 (R4).** *If there is a heavy vertex  $v$  of degree at most 5, then delete  $N[v]$  from the graph and add  $w(v)$  to  $M_c$ .*

It is an effective rule that has been used in some experimental algorithms [14, 19]. In this paper, we will only check heavy vertices of degree bounded by 5 and then it can be done in polynomial time. Note that degree-0 vertices will be reduced as heavy vertices in this step.

### 3.2 Reductions Based on Degree-2 Vertices

For unweighted graphs, we have good reduction rules to deal with all degree-2 vertices (see the reduction rule in [3]). However, for weighted graphs, it becomes much more complicated. The following R5 is generalized from the concept of folding degree-2 vertices in unweighted graphs in [3], which has been also used in some experimental algorithms [14, 19]. We also consider more reduction rules for degree-2 vertices in some complicated structures.

**Reduction Rule 5 (R5).** *If there is a degree-2 vertex  $v$  with two neighbors  $\{u_1, u_2\}$  such that  $w(u_1) + w(u_2) > w(v) \geq \max\{w(u_1), w(u_2)\}$ , then delete  $\{v, u_1, u_2\}$  from the graph  $G$ , introduce a new vertex  $v'$  adjacent to  $N_G(\{v, u_1, u_2\})$  with weight  $w(v') := w(u_1) + w(u_2) - w(v)$ , and add  $w(v)$  to  $M_c$ .*

**Reduction Rule 6 (R6)** ([19]). *If there is a path  $v_1v_2v_3v_4$  such that  $d_G(v_2) = d_G(v_3) = 2$  and  $w(v_1) \geq w(v_2) \geq w(v_3) \geq w(v_4)$ , then remove  $v_2$  and  $v_3$  from the graph, add an edge  $v_1v_4$  if it does not exist, update the weight of  $v_1$  by letting  $w(v_1) := w(v_1) + w(v_3) - w(v_2)$ , and add  $w(v_2)$  to  $M_c$ .*

**Reduction Rule 7 (R7)** ([19]). *If there is a 4-cycle  $v_1v_2v_3v_4$  such that  $d_G(v_2) = d_G(v_3) = 2$  and  $w(v_1) \geq w(v_2) \geq w(v_3)$ , then remove  $v_2$  and  $v_3$ , update the weight of  $v_1$  by letting  $w(v_1) := w(v_1) + w(v_3) - w(v_2)$ , and add  $w(v_2)$  to  $M_c$ .*

**Reduction Rule 8 (R8).** *If there is a 4-path  $v_1v_2v_3v_4v_5$  such that  $d_G(v_2) = d_G(v_3) = d_G(v_4) = 2$  and  $w(v_1) \geq w(v_2) \geq w(v_3) \leq w(v_4) \leq w(v_5)$ , then remove  $v_2$  and  $v_4$ , add edges  $v_1v_3$  and  $v_3v_5$ , update the weight of  $v_1$  by letting  $w(v_1) := w(v_1) + w(v_3) - w(v_2)$  and the weight of  $v_5$  by letting  $w(v_5) := w(v_5) + w(v_3) - w(v_4)$ , and add  $w(v_2) + w(v_4) - w(v_3)$  to  $M_c$ .*

**Reduction Rule 9 (R9).** *For a 5-cycle  $v_1v_2v_3v_4v_5$  such that  $d_G(v_2) = d_G(v_3) = d_G(v_5) = 2$ ,  $\min\{d(v_1), d(v_4)\} \geq 3$ , and  $w(v_1) \geq w(v_2) \geq w(v_3) \leq w(v_4)$ ,*

- (1) *if  $w(v_3) > w(v_5)$ , then remove  $v_5$ , update the weight of  $v_i$  by letting  $w(v_i) := w(v_i) - w(v_5)$  for  $i = 1, 2, 3, 4$ , and add  $2w(v_5)$  to  $M_c$ .*
- (2) *if  $w(v_3) \leq w(v_5)$ , then remove  $v_2$  and  $v_3$ , update the weight of  $v_1$  by letting  $w(v_1) := w(v_1) - w(v_2)$ , the weight of  $v_4$  by letting  $w(v_4) := w(v_4) - w(v_3)$  and the weight of  $v_5$  by letting  $w(v_5) := w(v_5) - w(v_3)$ , and add  $w(v_2) + w(v_3)$  to  $M_c$ .*

**Reduction Rule 10 (R10).** *For a 6-cycle  $v_1v_2v_3v_4v_5v_6$  such that  $d_G(v_2) = d_G(v_3) = d_G(v_5) = d_G(v_6) = 2$ ,  $w(v_1) \geq \max\{w(v_2), w(v_6)\}$ ,  $w(v_4) \geq \max\{w(v_3), w(v_5)\}$ , and  $w(v_6) \geq w(v_5)$ ,*

- (1) *if  $w(v_2) \geq w(v_3)$ , then remove  $v_5$  and  $v_6$ , and update the weight of  $v_2$  by letting  $w(v_2) := w(v_2) + w(v_6)$  and the weight of  $v_3$  by letting  $w(v_3) := w(v_3) + w(v_5)$ ;*
- (2) *if  $w(v_2) < w(v_3)$ , then remove  $v_6$ , add edge  $v_1v_5$ , and update the weight of  $v_2$  by letting  $w(v_2) := w(v_2) + w(v_6)$ , the weight of  $v_3$  by letting  $w(v_3) := w(v_3) + w(v_5)$ , and the weight of  $v_5$  by letting  $w(v_5) := w(v_6) + w(v_3) - \max\{w(v_2) + w(v_6), w(v_3) + w(v_5)\}$ .*

### 3.3 Reductions Based on Small Cuts

We also have some reduction rules to deal with vertex-cuts of size one or two, which can even be used to design a polynomial-time divide-and-conquer algorithm. However, a graph may not always have vertex-cuts of small size.

**Reduction Rule 11 (R11).** For a vertex-cut  $\{u\}$  with a connected component  $G^*$  in  $G - u$  such that  $2\delta_3 - \delta_2 \leq \sum_{v \in G^*} \delta_{d_G(v)} \leq 10$ ,

- (1) if  $w(u) + \alpha(G^* - N[u]) \leq \alpha(G^*)$ , then remove  $G^*$  and  $\{u\}$  from  $G$  and add  $\alpha(G^*)$  to  $M_c$ ;
- (2) if  $w(u) + \alpha(G^* - N[u]) > \alpha(G^*)$ , then remove  $G^*$  from  $G$ , update the weight of  $u$  by letting  $w(u) := w(u) + \alpha(G^* - N[u]) - \alpha(G^*)$ , and add  $\alpha(G^*)$  to  $M_c$ .

**Lemma 4.** (\*) Let  $\{u, u'\}$  be a vertex-cut of size two in  $G$  and  $G^*$  be a connected component in  $G - \{u, u'\}$ , where we assume w.l.o.g. that  $\alpha(G^* - N[u]) \geq \alpha(G^* - N[u'])$ . We construct a new graph  $G'$  from  $G$  as follows: remove  $G^*$ ; add three new vertices  $\{v_1, v_2, v_3\}$  with weight  $w(v_1) = \alpha(G^* - N[u']) - \alpha(G^* - N[\{u, u'\}])$ ,  $w(v_2) = \alpha(G^* - N[u]) - \alpha(G^* - N[\{u, u'\}])$  and  $w(v_3) = \alpha(G^*) - \alpha(G^* - N[u])$ , and add five new edges  $uv_1, v_1v_2, v_2u', uv_3$  and  $u'v_3$ . It holds that

$$\alpha(G) = \alpha(G') + \alpha(G^* - N[\{u, u'\}]).$$

**Reduction Rule 12 (R12).** For a vertex-cut  $\{u, u'\}$  of size two with a connected component  $G^*$  in  $G - \{u, u'\}$  such that  $2\delta_3 + \delta_2 \leq \sum_{v \in G^*} \delta_{d_G(v)} \leq 10$ , we construct the graph  $G'$  in Lemma 4, replace  $G$  with  $G'$ , and add  $\alpha(G^* - N[\{u, u'\}])$  to  $M_c$ .

### 3.4 Analyzing Reduction Rules

It is easy to see that each application of our reduction rules can be executed in polynomial time. We also show that

**Lemma 5.** The measure  $p$  will not increase after applying any reduction rule.

**Definition 1.** An instance is reduced, if no reduction rule can be applied.

**Lemma 6.** (\*) In a reduced instance, any two degree-2 vertices in different chains have at most one common chain-neighbor of degree at least 3, and each cycle contains at least three vertices of degree  $\geq 3$ .

**Lemma 7.** (\*) For a triangle  $C$  in a reduced instance, each vertex in  $C$  is a vertex of degree  $\geq 3$  and it has a chain-neighbor of degree at least 3 not in  $C$ .

## 4 Branching Rules

### 4.1 Two Branching Rules

We have two branching rules. The first branching rule is to branch on a vertex  $v$  by considering two cases: (i) there is a maximum weighted independent set in  $G$  which does not contain  $v$ ; (ii) every maximum weighted independent set in  $G$  contains  $v$ . For the former case, we simply delete  $v$  from the graph. For the latter case, by Lemma 2 we know that we can include the set  $S_v$  confining  $v$  in the independent set. So we delete  $N[S_v]$  from the graph.

**Branching Rule 1 (Branching on a vertex).** *Branch on a vertex  $v$  to generate two sub instances by either deleting  $v$  from the graph or deleting  $N[S_v]$  from the graph and adding  $w(S_v)$  to  $M_c$ .*

Since each independent set contains at most two vertices in each 4-cycle, we have the second rule.

**Branching Rule 2 (Branching on a 4-cycle).** *Branch on a 4-cycle  $v_1v_2v_3v_4$  to generate two sub instances by deleting either  $\{v_1, v_3\}$  or  $\{v_2, v_4\}$  from  $G$ .*

### 4.2 The Analysis and Some Properties

The hardest part is to analyze how much we can decrease the measure in each sub-branch of a branching operation. Usually, we need to deeply analyze the local graph structure and use case-analysis. Here we try to summarize some common properties. The following notations will be frequently used in the whole paper.

Let  $S$  be a vertex subset in a reduced graph  $G$ . We use  $G_{-S}$  to denote the graph after deleting  $S$  from  $G$  and iteratively applying R1 to R4 until none of them can be applied. We use  $R_S$  to denote the set of deleted vertices during applying R1 to R4 on  $G - S$ . Then  $G_{-S} = G - (S \cup R_S)$ . We also use  $e_S$  to denote the number of edges between  $S \cup R_S$  and  $V \setminus (S \cup R_S)$  in  $G$ . We have the following lemmas for some bounds on  $p(G) - p(G_{-S})$ . Note that  $G_{-S}$  may not be a reduced graph because of reduction rules from R5 to R12 and we may further apply reduction rules to further decrease the measure  $p$ .

**Lemma 8.** (\*) *It holds that*

$$p(G) - p(G_{-S}) \geq \sum_{u \in S \cup R_S} \delta_{d_G(u)} + e_S \delta_3^{<-1>}. \tag{3}$$

In some cases, we can not use the bound in (3) directly, since we may not know the vertex set  $R_S$ . So we also consider some special cases and relaxed bounds.

**Lemma 9.** (\*) *Let  $S = \{v\}$  be a set of a vertex of degree  $\geq 3$ . We have that*

$$p(G) - p(G_{-S}) \geq \delta_{d(v)} + \sum_{u \in N(v)} \delta_{d(u)}^{<-1>} + q_2 \delta_3^{<-1>},$$

where  $q_2$  is the number of degree-2 vertices in  $N(v)$ .

**Lemma 10.** (\*) *If  $S \cup R_S$  contains  $N[v]$  for some vertex  $v$  of degree  $\geq 3$ , then we have that*

$$p(G) - p(G_{-S}) \geq \sum_{u \in N[v]} \delta_{d(u)} + q_2 \delta_3^{<-1>},$$

where  $q_2$  is the number of degree-2 vertices in  $N(v)$ .

Recall that we use  $\mathcal{C}(G')$  to denote the set of connected components of the graph  $G'$ . We can easily observe the following lemma, which will be used to prove several bounds on  $p(G) - p(G_{-S})$ .



**Lemma 11.** *Let  $S$  be a vertex subset. Let  $S'$  be a subset of  $S \cup R_S$  and  $R' = S \cup R_S \setminus S'$ . The number of edges between  $S \cup R_S$  and  $V \setminus (S \cup R_S)$  is  $e_S$ , and the number of edges between  $S'$  and  $V \setminus S'$  is  $k$ . For any component  $H \in \mathcal{C}(G[R'])$ , the number of edges between  $S'$  and  $H$  is  $l_H$  and the number of edges between  $H$  and  $N(S \cup R_S)$  is  $r_H$ . We have that*

$$k - e_S = \sum_{H \in \mathcal{C}(G[R'])} (l_H - r_H).$$

Furthermore, for any component  $H \in \mathcal{C}(G[R'])$  containing only degree-2 vertices, it holds that  $l_H - r_H = 0$  or 2.

**Lemma 12.** (\*) *For any subset  $S' \subseteq S \cup R_S$  with  $k$  edges between  $S'$  and  $V \setminus S'$ , it holds that*

$$p(G) - p(G_{-S}) \geq \sum_{u \in S'} \delta_{d_G(u)} + e_S \delta_3^{<-1>} + \begin{cases} 0, & k - e_S \leq 0 \\ \delta_3, & k - e_S = 1 \\ \delta_2, & k - e_S = 2 \\ \delta_3, & k - e_S = 3 \\ 2\delta_2, & k - e_S > 3. \end{cases}$$

**Lemma 13.** (\*) *Assume that a reduced graph  $G$  has a maximum degree 3 and has no 3 or 4-cycles. For any subset  $S' \subseteq S \cup R_S$  with  $k$  edges between  $S'$  and  $V \setminus S'$ , if the diameter of the induced graph  $G[S']$  is 2, then it holds that either  $p(G) - p(G_{-S}) > 10$  or*

$$p(G) - p(G_{-S}) \geq \sum_{u \in S'} \delta_{d_G(u)} + 3\delta_3^{<-1>} + \begin{cases} 0, & k \leq 3 \\ \delta_3^{<-1>}, & k = 4 \\ 2\delta_2, & k = 5 \\ \delta_2 + \delta_3, & k = 6. \end{cases}$$

**Lemma 14.** (\*) *Assume that a reduced graph  $G$  has a maximum degree 3, and each cycle  $C$  in it contains at least five vertices, where at least four vertices are degree-3 vertices. For any subset  $S' \subseteq S \cup R_S$  with  $k$  edges between  $S'$  and  $V \setminus S'$ , if each path  $P$  in the induced graph  $G[S']$  contains either at most three vertices or at most two degree-3 vertices, then it holds either  $p(G) - p(G_{-S}) > 10$  or*

$$p(G) - p(G_{-S}) \geq \sum_{u \in S'} \delta_{d_G(u)} + \begin{cases} k\delta_3^{<-1>}, & k \leq 5 \\ \delta_3 + 2\delta_2 + 3\delta_3^{<-1>}, & k = 6. \end{cases}$$

### 5 The Algorithm

Now we describe the main steps of the algorithm. When the algorithm executes one step, we assume that all previous steps can not be applied.

**Step 1 (Applying Reductions).** *If the instance is not reduced, iteratively apply reduction rules in order, i.e., when one reduction rule is applied, no reduction rule with a smaller index can be applied on the graph.*

**Step 2 (Solving Small Components).** *If there is a connected component  $G^*$  of  $G$  such that  $p(G^*) \leq 10$ , solve the component  $G^*$  directly and return  $\alpha(G - G^*) + \alpha(G^*)$ .*

**Step 3 (Branching on Vertices of Degree  $\geq 5$ ).** *If there is a vertex  $v$  with degree  $d(v) \geq 5$ , then branch on  $v$  with Branching Rule 1 by either excluding  $v$  from the independent set or including  $S_v$  in the independent set.*

**Lemma 15.** *(\*) Step 3 followed by applications of reduction rules creates a branching vector covered by [5.368, 7.248].*

**Step 4 (Branching on 4-Cycles with Chords).** *If there is a 4-cycle  $C = v_1v_2v_3v_4$  with a chord  $v_1v_3 \in E$ , then branch on the 4-cycle with Branching Rule 2 by excluding either  $\{v_1, v_3\}$  or  $\{v_2, v_4\}$  from the independent set.*

**Lemma 16.** *(\*) Step 4 followed by applications of reduction rules creates a branching vector covered by one of  $[3\delta_4 + \delta_3^{<-1>}, 4\delta_4 + 2\delta_3^{<-1>}] = [5.496, 7.744]$  and  $[4\delta_4, 2\delta_4 + 2\delta_3 + 2\delta_2] = [6.496, 6]$ .*

**Step 5 (Branching on Degree-4 Vertices).** *If there is a degree-4 vertex  $v$ , then branch on it with Branching Rule 1 by either excluding  $v$  from the independent set or including  $S_v$  in the independent set.*

**Lemma 17.** *(\*) Step 5 followed by applications of reduction rules creates a branching vector covered by one of  $[5.624, 5.624]$ ,  $[5.248, 6]$ ,  $[4.872, 6.624]$ ,  $[4.496, 7.248]$ , and  $[4.12, 7.872]$ .*

**Step 6 (Branching on Other 4-Cycles).** *If there is a 4-cycle  $C = v_1v_2v_3v_4$ , then branch on the 4-cycle with Branching Rule 2 by excluding either  $\{v_1, v_3\}$  or  $\{v_2, v_4\}$  from the independent set.*

**Lemma 18.** *(\*) Step 6 followed by applications of reduction rules creates a branching vector covered by  $[6\delta_3 - 2\delta_2, 6\delta_3 - 2\delta_2] = [5.248, 5.248]$ .*

**Step 7 (Branching on Triangles).** *If there is a triangle  $C = v_1v_2v_3$ , where we assume without loss of generality that  $w(v_1) \geq \max\{w(v_2), w(v_3)\}$  and  $v_1$  is chain-adjacent to a degree-3 vertex  $u \neq v_2, v_3$ , then branch on  $u$  with Branching Rule 1.*

**Lemma 19.** *(\*) Step 7 followed by applications of reduction rules creates a branching vector covered by one of  $[6\delta_3 - 3\delta_2, 7\delta_3 + \delta_2] = [4.872, 7.376]$  and  $[6\delta_3 - 2\delta_2, 5\delta_3 + 2\delta_2] = [5.248, 5.752]$ .*

**Step 8 (Branching on Cycles Containing Three Degree-3 Vertices).** *If there is a cycle  $C$  containing exactly three degree-3 vertices  $\{v_1, v_2, v_3\}$ , where we assume without loss of generality that  $v_1$  is chain-adjacent to a degree-3 vertex  $u \neq v_2, v_3$ , then branch on  $u$  with Branching Rule 1.*

**Lemma 20.** *(\*) Step 8 followed by applications of reduction rules can create a branching vector covered by one of  $[6\delta_3 - 4\delta_2, 8\delta_3 - 2\delta_2] = [4.496, 7.248]$ ,  $[6\delta_3 - 3\delta_2, 6\delta_3 - \delta_2] = [4.872, 5.624]$ , and  $[6\delta_3 - 2\delta_2, 6\delta_3 - 2\delta_2] = [5.248, 5.248]$ .*

**Step 9 (Branching on Degree-3 Vertices with Two Degree-2 Neighbors).** *If there is degree-3 vertex  $u$  having two degree-2 neighbors and one degree-3 neighbor  $v$ , then branch on  $v$  with Branching Rule 1.*

**Lemma 21.** *(\*) Step 9 followed by applications of reduction rules creates a branching vector covered by one of  $[4\delta_3 - \delta_2, 8\delta_3 - \delta_2] = [3.624, 7.624]$ ,  $[4\delta_3, 8\delta_3 - 4\delta_2] = [4, 6.496]$ , and  $[4\delta_3 + \delta_2, 6\delta_3] = [4.376, 6]$ .*

**Step 10 (Branching on Degree-3 Vertices of a Mixed Case).** *If a degree-3 vertex  $u$  without degree-3 neighbors is chain-adjacent to a degree-3 vertex  $v$  with exactly two degree-3 neighbors, then branch on  $v$  with Branching Rule 1.*

**Lemma 22.** *(\*) Step 10 followed by applications of reduction rules creates a branching vector covered by  $[4\delta_3, 8\delta_3 - 2\delta_2] = [4, 7.248]$ .*

**Step 11 (Branching on Degree-3 Vertices With At Least Two Degree-3 Neighbors).** *If there is a connected component  $H$  containing a degree-3 vertex with at least two degree-3 neighbors, we let  $u$  be the vertex of the maximum weight in  $H$  and let  $v$  be a degree-3 neighbor of  $u$ , and branch on  $v$  with Branching Rule 1.*

**Lemma 23.** *(\*) Step 11 followed by applications of reduction rules creates a branching vector covered by one of  $[4\delta_3 - \delta_2, 8\delta_3 - \delta_2] = [3.624, 7.624]$ , and  $[4\delta_3, 8\delta_3 - 4\delta_2] = [4, 6.496]$ .*

**Step 12 (Branching on Other Degree-3 Vertices).** *Pick up an arbitrary degree-3 vertex  $v$  and branch on it with Branching Rule 1.*

**Lemma 24.** *(\*) Step 12 followed by applications of reduction rules creates a branching vector covered by  $[4\delta_3 + 6\delta_2, 4\delta_3 + 6\delta_2] = [6.256, 6.256]$ .*

It is easy to see that above steps cover all the cases. Among all the branching vectors, the bottleneck ones are  $[4\delta_3, 8\delta_3 - 4\delta_2] = [4, 6.496]$  in Lemma 21,  $[4\delta_3 + \delta_2, 6\delta_3] = [4.376, 6]$  in Lemma 21, and  $[4\delta_3, 8\delta_3 - 4\delta_2] = [4, 6.496]$  in Lemma 23. All of them have a branching factor of 1.14427. So we get that

**Theorem 1.** MAXIMUM WEIGHTED INDEPENDENT SET can be solved in  $O^*(1.1443^p)$  time and polynomial space.

By Lemma 1 and Theorem 1, we get that

**Corollary 1.** MAXIMUM WEIGHTED INDEPENDENT SET in graphs with average degree  $x$  can be solved in  $O^*(1.1443^{(0.624x - 0.872)n})$  time and polynomial space.

Let  $x = 3$  in Lemma 1, we get that  $p \leq n$  and the following result.

**Theorem 2.** MAXIMUM WEIGHTED INDEPENDENT SET in graphs with the average degree at most 3 can be solved in  $O^*(1.1443^n)$  time and polynomial space.

## References

1. Been, K., Daiches, E., Yap, C.K.: Dynamic map labeling. *IEEE Trans. Visual Comput. Graph.* **12**(5), 773–780 (2006)
2. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: Fast algorithms for max independent set. *Algorithmica* **62**(1), 382–415 (2012)
3. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. *J. Algorithms* **41**(2), 280–301 (2001)
4. Dahllöf, V., Jonsson, P.: An algorithm for counting maximum weighted independent sets and its applications. In: *SODA 2002*, pp. 292–298 (2002)
5. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulae. *Theor. Comput. Sci.* **332**(1–3), 265–291 (2005)
6. Fomin, F.V., Gaspers, S., Saurabh, S.: Branching and treewidth based exact algorithms. In: *ISAAC 2006*. LNCS, vol. 4288, pp. 16–25 (2006)
7. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. *Algorithmica* **54**(2), 181–207 (2009)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. *J. ACM* **56**(5), 25:1–25:32 (2009)
9. Fürer, M., Kasiviswanathan, S.P.: Algorithms for counting 2-SAT solutions and colorings with applications. In: *AAIM 2007*, pp. 47–57 (2007)
10. Huang, S., Xiao, M., Chen, X.: Exact algorithms for maximum weighted independent set on sparse graphs. *CoRR* abs/2108.12840 (2021)
11. Jian, T.: An  $O(2^{0.304n})$  algorithm for solving maximum independent set problem. *IEEE Trans. Comput.* **35**(9), 847–851 (1986)
12. Karp, R.M.: Reducibility among combinatorial problems. In: *Proceedings of a Symposium on the Complexity of Computer Computations*, pp. 85–103. The IBM Research Symposia Series (1972)
13. Kneis, J., Langer, A., Rossmanith, P.: A fine-grained analysis of a simple independent set algorithm. In: *Proceedings of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. LIPIcs, vol. 4, pp. 287–298 (2009)
14. Lamm, S., Schulz, C., Strash, D., Williger, R., Zhang, H.: Exactly solving the maximum weight independent set problem on large real-world graphs. In: *Proceedings of Algorithm Engineering and Experiments*, pp. 144–158 (2019)
15. Liao, C.S., Liang, C.W., Poon, S.H.: Approximation algorithms on consistent dynamic map labeling. *Theor. Comput. Sci.* **640**, 84–93 (2016)
16. Robson, J.M.: Algorithms for maximum independent sets. *J. Algorithms* **7**(3), 425–440 (1986)
17. Tarjan, R.E., Trojanowski, A.E.: Finding a maximum independent set. *SIAM J. Comput.* **6**(3), 537–546 (1977)
18. Wahlström, M.: A tighter bound for counting max-weight solutions to 2SAT instances. In: *Proceedings of Third International Workshop on Parameterized and Exact Computation*. LNCS, vol. 5018, pp. 202–213 (2008)
19. Xiao, M., Huang, S., Zhou, Y., Ding, B.: Efficient reductions and a fast algorithm of maximum weighted independent set. In: *WWW 2021: The Web Conference 2021*, pp. 3930–3940 (2021)
20. Xiao, M., Nagamochi, H.: Confining sets and avoiding bottleneck cases: a simple maximum independent set algorithm in degree-3 graphs. *Theor. Comput. Sci.* **469**, 92–104 (2013)
21. Xiao, M., Nagamochi, H.: Exact algorithms for maximum independent set. *Inf. Comput.* **255**, 126–146 (2017)