Happy Set Problems on Cubic Graphs and Convex Bipartite Graphs

Yuichi Asahiro[1], Hiroshi Eto[2], Guohui Lin[3], Eiji Miyano[2], and Yudai Oka[2]

[1] Kyushu Sangyo University, Fukuoka, Japan (asahiro@is.kyusan-u.ac.jp)
[2] Kyushu Institute of Technology, Iizuka, Japan (eto@ai.kyutech.ac.jp, miyano@ai.kyutech.ac.jp, oka.yudai550@mail.kytech.jp)
[3] University of Alberta, Edmonton, Canada (guohui@ualberta.ca)

Abstract. In this paper, we study the approximability and the computational complexity of the Maximum Happy Set problem (MaxHS for short) on graph classes: For an undirected graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, a vertex v is *happy* if v and all its neighbors are in S; otherwise *unhappy*. Given a graph G and an integer k, the goal of MaxHS is to find a subset S of k vertices such that the number of happy vertices is maximized. MaxHS is known to be NP-hard even for bipartite graphs and cubic (i.e., 3-regular) graphs. As for the approximability, it is known that there is a polynomial-time $(2\Delta + 1)$-approximation algorithm for MaxHS on graphs with maximum degree Δ, and furthermore the approximation ratio can be improved to Δ if Δ is a constant. In this paper, we first design a polynomial-time 2-approximation algorithm for MaxHS on cubic graphs. We then design an exact algorithm for MaxHS on n-vertex convex bipartite graphs, which runs in $O(n^2 + k^3 n)$ time.

Keywords: Happy set problem· Regular graph · Convex bipartite graph

1 Introduction

The *homophily* is the fundamental law that people are more likely to connect with people sharing similar interests with them in social networks [15]. Motivated by the homophily law, Zhang and Li [21] introduced the concept of *happy* vertices in terms of graph coloring problems, and subsequently the concepts have attracted many researchers [1, 2, 9, 16, 22]. Later, Asahiro et al. [4] formulated the Maximum Happy Set Problem (MaxHS) in terms of graph subset problems: For an undirected graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, we say that a vertex $v \in V$ is *happy* if v and all its neighbors are in S, and *unhappy*, otherwise. Given an undirected graph $G = (V, E)$ and an integer k, the goal of MaxHS is to find a subset $S \subseteq V$ of k vertices such that the number of happy vertices is maximized. MaxHS is NP-hard even for bipartite graphs [3], co-bipartite graphs [11], cubic graphs [3], and split graphs [4]. On the other hand, fortunately, MaxHS can be solved in $O(k^2 n)$ time for block graphs [3], $O(n^2 + k^3 n)$ time for interval graphs [11], $O(kn \log k + m)$ time for proper interval graphs [5],

$O(n^2 + k^3 n)$ time for permutation graphs, and $O(n^2 + d^2(k+1)^{3d}n)$ time for d-trapezoid graphs [11], where n and m are the numbers of vertices and edges in the input graph, respectively. (See, e.g., [7] for details on graph classes.) As for the approximability, it is known [3] that there is a polynomial-time $(2\Delta+1)$-approximation algorithm for MaxHS on graphs with maximum degree Δ, and furthermore the approximation ratio can be improved to Δ if Δ is a constant. On the other hand, there are no known inapproximability results for MaxHS as long as the authors know, while its NP-hardness is known as mentioned above.

In this paper, we study the approximability of MaxHS on *cubic graphs*, and the computational complexity of MaxHS on *convex bipartite graphs*. A graph G is *cubic* if the degree of every vertex in G is exactly three. Regular graphs appear in various fields due to their uniform structures, e.g., sensor network topologies and game theory. Especially, since even for cubic graphs, many combinatorial problems become hard to solve in polynomial time, many researchers are devoted to studying cubic graphs (see, e.g., [13]). A graph G is *bipartite* if its vertex set can be partitioned into two disjoint subsets, called *partite sets*, V_U and V_W so that every edge connects a vertex in V_U to one in V_W, and it is often represented by $G = (V_U \cup V_W, E)$. A bipartite graph $G = (V_U \cup V_W, E)$ is *convex* if the vertices in V_U can be ordered in such a way that, for each $w \in V_W$, the neighborhood $N(w)$ of w are consecutive in V_U. That is, there exists an ordering $\sigma : V_U \to \{1, 2, \ldots, |V_U|\}$ such that for any pair of two vertices $u, u' \in N(w) \subseteq V_U$ and any $u'' \in V_U$ with $\sigma(u) < \sigma(u'') < \sigma(u')$, it holds $u'' \in N(w)$. The ordering σ of the vertices in V_U is said to be *convex*, and G is said to be *convex with respect to* V_U. The convex bipartite graphs naturally arise in several scheduling and industrial applications, and thus many efficient algorithms have been designed on convex bipartite graphs for popular graph optimization problems such as Maximum Matching [8, 12, 20], Maximum Independent Set [19], and Minimum Feedback Vertex Set [17]. Our contributions are summarized as follows (but, due to space limitations, we omit many details and proofs from the paper):

1. In Section 2, we first design a polynomial-time 2-approximation algorithm for MaxHS on cubic graphs that improves the best known approximation ratio 3 in [3].
2. In Section 3, we show that MaxHS can be solved in $O(n^2 + k^3 n)$ time if the input graph is restricted to convex bipartite graphs, while MaxHS on bipartite graphs is NP-hard as shown in [3].

Related work. MaxHS is also studied from the point of view of parameterized complexity. MaxHS is W[1]-hard when parameterized by k even on split graphs [4]. By contrast, MaxHS admits fixed-parameter algorithms when parameterized by structural parameters of the graphs, such as tree-width [4], neighborhood diversity [4], twin-cover number [4], modular-width [18], and clique-width [18] of a graph.

Preliminaries. Let $G = (V, E)$ be an undirected graph, where V and E denote the set of vertices and the set of edges, respectively. Throughout the paper, let

$n = |V|$. for any given graph. We denote an edge with the endpoints u and v by $\{u, v\}$. A graph H is a subgraph of a graph $G = (V, E)$ if $V(H) \subseteq V$ and $E(H) \subseteq E$. For a subset of vertices $U \subseteq V$, let $G[U]$ be the subgraph of G induced by U. The set of vertices adjacent to a vertex v in G, i.e., the open neighborhood of v is denoted by $N(v) = \{u \in V \mid \{u, v\} \in E\}$. Similarly, let $N(S) = \{u \in V \setminus S \mid v \in S, \{u, v\} \in E\}$ be the open neighborhood of a subset S of vertices. The closed neighborhood of v (S, resp.) is denoted by $N[v]$ ($N[S]$, resp.), i.e., $N[v] = \{v\} \cup N(v)$ ($N[S] = S \cup N(S)$, resp.). For a proper subset $S \subset V$ of the vertices, the *cut* $(S, V \setminus S)$ is the set of all edges in G with one endpoint in S and the other in $V \setminus S$.

For an undirected graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, a vertex v is *happy* if $N[v] \subseteq S$; otherwise, i.e., if $N[v] \not\subseteq S$, then v is *unhappy*. Let $\#h(S)$ and $\#u(S)$ denote the number of happy and unhappy vertices in a subset S of vertices, respectively. By definition, $|S| = \#h(S) + \#u(S)$ holds. An algorithm `ALG` is called an α-approximation algorithm and `ALG`'s approximation ratio is α if $OPT(G)/ALG(G) \leq \alpha$ holds for every input G, where $ALG(G)$ and $OPT(G)$ are the numbers of happy vertices obtained by applying `ALG` and an optimal algorithm to G, respectively.

2 Cubic graphs

In this section, we propose a polynomial-time 2-approximation algorithm, named `CUBIC`, for MaxHS on cubic graphs. From now on, without loss of generality, we can assume that $4 \leq k \leq |V| - 1$ for cubic graphs since the size of any happy set in every connected component is zero if $1 \leq k \leq 3$; and the size is trivially $|V|$ if $k = |V|$. If $4 \leq k \leq 7$, then the proposed algorithm `CUBIC` exhaustively finds an optimal solution. Otherwise, `CUBIC` executes our main procedure `ALG`, which behaves as follows: (1) `ALG` initially selects a special subset of vertices into a partial solution set S' in **Step 1**. (2) In **Steps 2** through **4**, `ALG` enlarges S' by putting the neighbor vertices in $N(S')$ into S'. (3) `ALG` eventually finds a solution S of k vertices, and outputs S in **Steps 5** and **6**. In the following, Section 2.1 first describes our main procedure `ALG`. Next, a *potential method* is used to calculate the *amortized* number of happy vertices obtained by the main part (**Steps 2** through **4**) of `ALG` in Section 2.2. Then, Section 2.3 observes the initial setup **Step 1** of `ALG` to make the potential sufficiently large at the beginning stage. Finally, we describe the whole algorithm `CUBIC`, estimate its approximation ratio by using the potential method, and bound its running time in Section 2.4.

2.1 Main procedure `ALG`

The following algorithm `ALG` is the main procedure applied to the case $k \geq 8$ to design the 2-approximation algorithm `CUBIC`. First, to make the basic strategies of `ALG` clear, see an example illustrated in Fig. 1. Remark that if an optimal algorithm `OPT` and `ALG` (or `CUBIC`) find h and (at least) $h/2$ happy vertices, respectively, then the approximation ratio of `ALG` is 2. (i) Suppose that the input

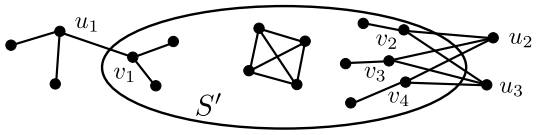

Fig. 1. Basic strategies of ALG

cubic graph G includes a connected component of four vertices in the middle. If ALG selects all the four vertices in the component, then it "optimally" obtains four happy vertices, which can be regarded that ALG gets four gains (new happy vertices), or it gets two excesses since two happy vertices are enough to achieve 2-approximation. (ii) Suppose that $\{v_1, v_2, v_3, v_4\} \subseteq S'$ and $\{u_1, u_2, u_3\} \subseteq V \setminus S'$ as shown in Fig. 1. For example, if u_1 is selected into S', then v_1 becomes happy, i.e., ALG gets one gain at the cost of one (vertex). As another example, if u_2 is selected into S', then only u_2 becomes happy, i.e., again ALG gets one gain at the cost of one; however, if two vertices u_2 and u_3 are selected, then five vertices u_2, u_3, v_2, v_3, and v_4 become happy, i.e., ALG gets five gains at the cost of two, i.e., 2.5 gains per cost one. That is, the latter vertex-selection is more efficient. To estimate the gain, the excess, and the efficiency of vertex-selections, we introduce two potential functions in Section 2.2. For a vertex set S' used in ALG and a vertex $v \in S'$, let $N_{out}(v)$ denote $N(v) \setminus S'$ for short. **Steps 1** and **5** in ALG are executed only once as an initial setup and a final treatment, respectively, and ALG iteratively executes **Steps 2** through **4** as the main loop.

Algorithm ALG

Input: A cubic graph $G = (V, E)$ and an integer k such that $8 \leq k \leq |V| - 1$.
Output: A solution set S of exactly k vertices.

Step 1. Set $S' = \emptyset$. If there is a proper subset $S_1 \subset V$ of vertices such that $|(S_1, V \setminus S_1)| \leq 3$ and $7 \leq |S_1| \leq k$, then execute **(a)**; otherwise execute **(b)**:
- **(a)** Choose a largest subset S_1 satisfying the above condition, then set $S' = S_1$ and $k' = k - |S_1|$.
- **(b)** Set $S' = S_1 = N[v]$ and $k' = k - 4$ for an arbitrary chosen vertex $v \in V$.

Step 2. If there is a vertex $v \in S'$ satisfying two conditions, $|N_{out}(v)| \in \{1, 2\}$ and $|N_{out}(v)| \leq k'$, then set $S' = S' \cup N_{out}(v)$ and $k' = k' - |N_{out}(v)|$. Repeat **Step 2** until there are no such vertices.

Step 3. If $|N_{out}(v)| = 3$ for every $v \in S'$, and $k' \geq 1$, then **(i)** choose any pair of two vertices $v \in S'$ and $u \in N_{out}(v)$, **(ii)** set $S' = S' \cup \{u\}$ and $k' = k' - 1$, and then **(iii)** go back to **Step 2**.

Step 4. If $|N_{out}(v)| = 0$ for every $v \in S'$ (i.e., $G[S']$ is a connected component), and $k' \geq 1$, then **(i)** choose an arbitrary vertex $u \in V \setminus S'$, **(ii)** set $S' = S' \cup \{u\}$ and $k' = k' - 1$, and then **(iii)** go back to **Step 3**.

Step 5. If $|N_{out}(v)| = 2$ for every $v \in S'$, and $k' = 1$, then choose any vertex $u \in V \setminus S'$, and set $S' = S' \cup \{u\}$.

Step 6. Output $S = S'$.

2.2 Steps 2 through 4 in ALG

We closely look at the main loop. Let $u_1, u_2, \ldots, u_\ell$ be the ℓ vertices added into S' in this order by **Steps 2** through **5**. The number ℓ of added vertices depends on the result of **Step 1**, i.e., $\ell = k - |S_1|$ if **Step 1(a)** is executed; otherwise, $\ell = k - 4$. When one iteration of **Step 2** adds two vertices in $N_{out}(v)$ into S' simultaneously, we consider the order of those two vertices is arbitrary.

We now introduce two potential functions f and g. Intuitively, for $1 \leq i \leq \ell$, $f(i)$ means the increased number of happy vertices by adding $u_1, u_2, \ldots, u_i$ into S_1, where S_1 is the set of vertices chosen by **Step 1**, i.e., $f(i) = \#h(S_1 \cup \{u_1, u_2, \ldots, u_i\}) - \#h(S_1)$. Then, we define $g(i) = f(i) - i/2$, representing an excess of the happy vertices to achieve 2-approximation. Now suppose that **Step 2** adds two vertices u_j and u_{j+1} into S', and $h \geq 1$ vertices newly become happy by adding them. Then, we carry out the following special treatment for **Step 2**: We distribute h to $f(j)$ and $f(j+1)$ in a such way that $f(j) = f(j-1) + h/2$ and $f(j+1) = f(j) + h/2$ by allowing $f(i)$ to have a fractional value.

As initial values, we define that $f(0)$ is the number of happy vertices in S_1 after **Step 1**, and then $g(0) = f(0) - |S_1|/2$ (later, we will explain these initial values more carefully). In the following, we observe how each step of ALG changes the values of f and g. For convenience, let $f'(i,j) = f(i) - f(j)$ for $i \geq j$, i.e., the increment difference of happy vertices during additions from u_{j+1} through u_i into S'. One sees that $f'(i,i) = 0$ and $f'(i,0) = f(i)$ hold for any $i \geq 0$. Later, we will show that, for each i, $f(i) \geq i/2$, $f'(i, i-1) \geq 0$, $g(i) \geq 0$, and/or essentially the same inequalities as these three inequalities hold, by which we can bound the approximation ratio of CUBIC. Let us take a closer look at **Step 2**; by simple calculations on the increased number of happy vertices in **Step 2**, we obtain the following inequalities for $f(i)$, $f'(i, i-1)$, and $g(i)$:

Lemma 1. *Suppose that **Step 2** adds only one vertex u_i into S' for $i \geq 1$ such that $N_{out}(v) = \{u_i\}$ for some $v \in S'$ and $N[u_i] \subseteq S' \cup \{u_i\}$. Then, it holds that*

- *$f(i) \geq f(i-1) + 2$, $f'(i, i-1) \geq 2$, $g(i) \geq g(i-1) + 3/2$, and*
- *if a vertex (u_{i+1}) will be added next into S', then it is by either **Step 2**, **3**, **4**, or **5***

Lemma 2. *Suppose that **Step 2** adds only one vertex u_i into S' for $i \geq 1$ such that $N_{out}(v) = \{u_i\}$ for some $v \in S'$ and $N[u_i] \not\subseteq S' \cup \{u_i\}$. Then, it holds that*

- *$f(i) \geq f(i-1) + 1$, $f'(i, i-1) \geq 1$, $g(i) \geq g(i-1) + 1/2$, and*
- *if a vertex (u_{i+1}) will be added next into S', then it is by either **Step 2** or **5**.*

Lemma 3. *Suppose that **Step 2** adds two vertices u_{i-1} and u_i into S' for $i \geq 2$ such that $N_{out}(v) = \{u_{i-1}, u_i\}$ for some $v \in S'$ and $N[u_{i-1}] \cup N[u_i] \subseteq S' \cup \{u_{i-1}, u_i\}$. Then, it holds that*

- *$f(i-1) \geq f(i-2) + 3/2$, $f'(i-1, i-2) \geq 3/2$, $g(i-1) \geq g(i-2) + 1$,*
- *$f(i) \geq f(i-1) + 3/2$, $f'(i, i-1) \geq 3/2$, $g(i) \geq g(i-1) + 1$, and*
- *if a vertex (u_{i+1}) will be added next into S', then it is by either **Step 2**, **3**, **4**, or **5**.*

Lemma 4. *Suppose that **Step 2** adds two vertices u_{i-1} and u_i into S' for $i \geq 2$ such that $N_{out}(v) = \{u_{i-1}, u_i\}$ for some $v \in S'$ and $N[u_{i-1}] \cup N[u_i] \not\subseteq S' \cup \{u_{i-1}, u_i\}$ Then, it holds that*

- *$f(i-1) \geq f(i-2) + 1/2$, $f'(i-1, i-2) \geq 1/2$, $g(i-1) \geq g(i-2)$,*
- *$f(i) \geq f(i-1) + 1/2$, $f'(i, i-1) \geq 1/2$, $g(i) \geq g(i-1)$, and*
- *if a vertex (u_{i+1}) will be added next into S', then it is by either **Step 2** or **5**.*

We next consider **Step 4** before **Step 3**. We can estimate $f(i)$, $f'(i, i-1)$, and $g(i)$ as follows after **Step 4**:

Lemma 5. *Suppose that a vertex u_i for $i \geq 1$ is added into S' in **Step 4**. Then, $f(i) = f(i-1)$, $f'(i, i-1) = 0$, and $g(i) \geq 3/2$ hold. If a vertex (u_{i+1}) will be added next into S', then it is by **Step 3**.*

The next lemma considers the case that u_i for $i \geq 2$ is added into S' in **Step 3**. The case where u_1 is added in **Step 3** will be discussed later.

Lemma 6. *Assume the followings (I) and (II): (I) A vertex u_i for $i \geq 2$ is added into S' in **Step 3**. (II) For the previous step, either of the following three conditions (1), (2), or (3) holds: (1) Two vertices u_{i-2} and u_{i-1} for $i \geq 3$ are added into S' in **Step 2**, and $g(i-2) \geq -2$ holds. (2) A vertex u_{i-1} is added into S' in **Step 2**, and $g(i-1) \geq -3/2$. (3) A vertex u_{i-1} is added into S' in **Step 4**. Then, $f(i) = f(i-1)$, $f'(i, i-1) = 0$, and $g(i) \geq 0$ hold. Moreover, if a vertex (u_{i+1}) will be added next into S', then it is by either **Step 2** or **5**.*

In the above lemmas, we gave recurrence formulas on $f(i)$ and $g(i)$. Among them, the assumptions in Lemma 6 are crucial to show $g(i) \geq 0$ holds for all i's, by which $f(i) \geq i/2$ holds for all i's, achieving the approximation ratio 2 of `CUBIC`. This will be discussed with an analysis on **Step 1** in the next subsection.

2.3 Step 1 in `ALG`

We first look at the case that **Step 1(a)** is executed. Let z be the last index that either **Step 2**, **3**, or **4** adds a vertex into S', i.e., $z = k - |S_1|$ if **Step 5** does not add any vertex into S', or $z = k - |S_1| - 1$ otherwise.

Lemma 7. *If **Step 1(a)** is executed, then, it holds that $f(i) \geq i/2$ for each $1 \leq i \leq z$.*

We next consider the case that **Step 1(b)** is executed. Since $k \geq 8$, it holds $k' \geq 4$ after **Step 1(b)**. Hence, for some $u_0 \in S'$, where u_0 is a vertex adjacent to v chosen in **Step 1(b)**, **Step 2** can add one or two vertices in $N(u_0) \setminus S'$ into S' as the next step, i.e., u_1 must be added into S' by **Step 2**. We further divide the case into two sub-cases, (i) **Step 3** or **4** adds at least one vertex u_j for some $j \geq 2$ into S', and (ii) only **Step 2** adds vertices into S' for each $1 \leq i \leq z$.

Lemma 8. *Suppose that **Step 1(b)** is executed, and Step 3 or 4 adds at least one vertex u_j for $j \geq 2$. Then, it holds $f(i) \geq i/2$ for $j \leq i \leq z$.*

The last case to consider is that only **Step 2** adds vertices into S' after **Step 1(b)** (before executing **Step 5**). Lemma 2 gives the following corollary.

Corollary 1. *Suppose that Step 2 adds $u_1, \ldots, u_i$ into S' for $i \geq 1$. Then $f(i) = \sum_{j=1}^{i} f'(j, j-1) \geq i/2$ holds.*

We end this section by calculating the optimal number of happy vertices for the case that **Step 1(b)** is executed.

Lemma 9. *Suppose that **Step 1(b)** is executed and S^* is an optimal solution such that $|S^*| \geq 8$. Then, $\#h(S^*) \leq k - 3$,*

2.4 The whole algorithm

The next two lemmas calculate the number of happy vertices in a solution obtained by `ALG`. As the worst case, we assume that **Step 5** adds one vertex into a solution, but does not obtain any new happy vertex in the following. Let ALG denote the number of happy vertices in the solution obtained by `ALG`.

Lemma 10. *If **Step 1(a)** in `ALG` is executed, then $ALG \geq k/2$ holds.*

Lemma 11. *If **Step 1(b)** in `ALG` is executed, then $ALG \geq (k-3)/2$ holds.*

We are now ready to describe the whole algorithm `CUBIC`:

Algorithm `CUBIC`

Input: A cubic graph $G = (V, E)$ and an integer k such that $4 \leq k \leq |V| - 1$.
Output: A set S of k vertices.

Step 1. If $4 \leq k \leq 7$, then let $\mathcal{U}$ be the set of all (proper) subsets $U \subset V$ such that $1 \leq |U| \leq 4$.
- **(1)** For each $U \in \mathcal{U}$, let $S_U = N[U]$, and then compute $\#h(S_U)$.
- **(2)** Let $S' = \arg\max_{U \in \mathcal{U}} \{\#h(N[U]) \mid |N[U]| \leq k\}$.
- **(3)** Select an arbitrary subset S'' of $k - |S'|$ vertices from $V \setminus S'$, then output $S = S' \cup S''$, and terminate.

Step 2. (Note that now $k \geq 8$ holds.) Execute `ALG` and terminate.

The approximation ratio and the running time of the above algorithm are given in the next theorem:

Theorem 1. *The algorithm `CUBIC` is an $O(n^4)$-time 2-approximation algorithm for MaxHS on cubic graphs.*

Proof. We first estimate the approximation ratio. Let S^* and OPT be an optimal solution and the number of happy vertices in S^*, respectively.

First, consider the case $4 \leq k \leq 7$. **Step 1** in `CUBIC` investigates whether $|N[U]| \leq k$ or not for every set $U \subset V$ such that $|U| \leq 4$, i.e., whether the set $N[U]$ is a feasible solution which makes all the vertices in U happy. Thus, if $OPT \leq 4$, then `CUBIC` finds an optimal solution. If $5 \leq OPT \leq 7$, then there must exist at least one set U of four vertices corresponding to four happy vertices in S^* such that $N[U] \subseteq S^*$, since $OPT = \#h(S^*) \geq 5$ Therefore, `CUBIC` exhaustively searches such U and selects all the vertices of $N[U]$ as a part of a solution. Now, we obtain (at least) four happy vertices in U. Hence the approximation ratio is at most $OPT/4 \leq 7/4 \leq 2$ for this case.

Next, consider the case $k \geq 8$, i.e., `ALG` is executed. When **Step 1(a)** in `ALG` is executed, $OPT/ALG \leq 2$ is satisfied since $ALG \geq k/2$ from Lemma 10 and clearly $OPT \leq k$. When **Step 1(b)** in `ALG` is executed, $OPT/ALG \leq 2$ is satisfied again since $ALG \geq (k-3)/2$ from Lemma 11 and $OPT \leq k-3$ from Lemma 9. Therefore, the approximation ratio of `CUBIC` is 2.

Now, we bound the running time of `CUBIC`. As a preprocessing, we construct $N[v]$ for every vertex v by scanning all edges. Since the input graph is cubic and has $O(n)$ edges, $|N[v]| \leq 4$ holds for any v and this preprocessing can be done in $O(n)$ time. Based on this preprocessing, **Step 1** of `CUBIC` can be done in $O(n^4)$ time: First, $\mathcal{U}$ is constructed by enumerating all U's in $O(n^4)$ time. Then, **Step 1(1)** obtains S_U for each $U \in \mathcal{U}$ in constant time by merging $N[v]$'s for at most four vertices v's in U, since $|N[v]| \leq 4$. Thus, **Step 1(1)** takes $O(n^4)$ time in total. In **Step 1(2)**, checking whether $|S_U| \leq k$, and (when it is true) computing $\#h(S_U)$ for each U by scanning constant number of edges, respectively take constant time. Finally, finding a maximum value among $O(n^4)$ values needs $O(n^4)$ time. **Step 1(3)** is clearly done in $O(n)$ time, since $k - |S'| < k < n$.

As for **Step 2** of `CUBIC`, we bound the running time of `ALG`. **Step 1** of `ALG` can be done in $O(n^4)$ time, by enumerating $O(n^3)$ combinations of at most three edges in the input cubic graph having n vertices and $O(n)$ edges, and investigating whether or not removal of each such set disconnects the input cubic graph in $O(n)$ time. For **Steps 2** through **5** of `ALG`, we need to update N_{out} by scanning all edges at the beginning of each step, which can be done in $O(n)$ time. **Step 2** finds a vertex to be processed in linear time from S', i.e., the total time to execute **Step 2** once is $O(n)$. **Steps 3** through **5** can be executed similarly to **Step 2**, that is, each of those steps takes $O(n)$ time. Since, the number of vertices added into a solution by **Steps 2**, **3**, and **4** is at most k, and hence the total number of executions of these steps is $O(k)$. Therefore, **Steps 2** through **4** spend $O(nk)$ time in total. **Step 5** is executed only once, and takes $O(n)$ time. Therefore, the most time-consuming part of `ALG` is **Step 1**, and hence the running time of `ALG` is $O(n^4)$.

As a result, the total running time of `CUBIC` is $O(n^4)$. □

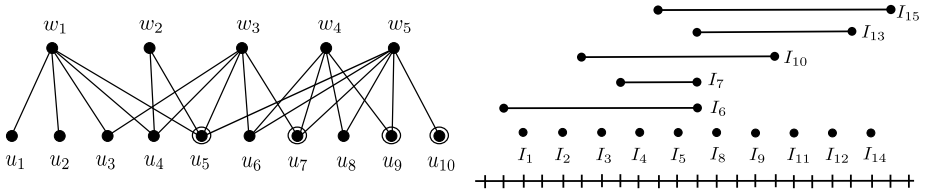

Fig. 2. (Left) Convex bipartite graph and (Right) its interval representation

3 Convex bipartite graphs

In this section, we design a polynomial-time algorithm for MaxHS on convex bipartite graphs. The following is our main theorem in this section.

Theorem 2. *Given an n-vertex convex bipartite graph G and an integer k, MaxHS can be solved in $O(n^2 + k^3 n)$ time.*

3.1 Preliminaries for convex bipartite graphs

Let $G = (V_U \cup V_W, E)$ be a convex bipartite graph with respect to V_U. Suppose that $|V_U| = n_U$, $|V_W| = n_W$, $V_U = \{u_1, u_2, \ldots, u_{n_U}\}$, $V_W = \{w_1, w_2, \ldots, w_{n_W}\}$, and $|V| = n_U + n_W = n$. Without loss of generality, there exists a convex ordering σ which satisfies $\sigma(u_1) < \sigma(u_2) < \cdots < \sigma(u_{n_U})$. Let u_{ℓ_i} and u_{r_i} be the leftmost and the rightmost vertices in $N(w_i)$, respectively. Assume that n_W vertices in $V_W = \{w_1, \ldots, w_{n_W}\}$ are sorted such that $\sigma(u_{r_1}) \leq \cdots \leq \sigma(u_{r_{n_W}})$ holds by the convex ordering σ, with ties broken arbitrarily. The ordering can be computed in linear time [6, 14]. See Fig. 2-(Left). For example, the neighborhood of $w_3 \in V_W$ is $N(w_3) = \{u_3, u_4, u_5, u_6, u_7\}$ which contains five consecutive vertices in V_U. One sees that $u_{r_1} = u_{r_2} = u_5$, $u_{r_3} = u_7$, $u_{r_4} = u_9$, and $u_{r_5} = u_{10}$. Also, $u_{\ell_1} = u_1$, $u_{\ell_2} = u_4$, $u_{\ell_3} = u_3$, $u_{\ell_4} = u_6$, and $u_{\ell_5} = u_5$.

We map all the vertices V_U and V_W to intervals of integers as follows: (i) Each $u_i \in V_U$ is mapped to the interval $int(u_i) = [2i, 2i]$, i.e., one even integer $2i$. (ii) Each w_j is mapped to the interval $int(w_j) = [2\ell_j - 1, 2r_j + 1]$. Note that this mapping of vertices to intervals is different from that for interval graphs. It is possible that an interval corresponding to a vertex in V_W intersects with another interval corresponding to another vertex in V_W, but there is no edge between these two vertices. Although our main strategy to design the algorithm is following and extending the ideas for the interval graphs in [11], this difference requires us to prove several lemmas similar to those in [11]; we cannot just use the results in [11], and/or we cannot give straightforward corollaries of them.

Let $\mathcal{I} = \{int(v) \mid v \in V_U \cup V_W\}$ be a set of the n intervals, corresponding to the n vertices in $V = V_U \cup V_W$. If an interval $I = [i_\ell, i_r]$, then we define $left(I) = i_\ell$ and $right(I) = i_r$. We sort those n intervals by the rightmost values with ties broken arbitrarily, and define the sorted n intervals as I_1, I_2, ..., I_n from now on, i.e., $right(I_i) < right(I_j)$ holds for $1 \leq i < j \leq n$. See the convex

bipartite graph in Fig. 2-(Left) again. For example, $I_1 = int(u_1)$, $I_2 = int(u_2)$, $I_6 = int(w_1)$, $I_7 = int(w_2)$, and so on. Also, for $\mathcal{I}' \subseteq \mathcal{I}$, let $U(\mathcal{I}') = \{I \in \mathcal{I}' \mid ver(I) \in V_U\}$, where $ver(I)$ represents a vertex corresponding to interval I, i.e., $ver(I) = int^{-1}(I)$. Analogously, for $\mathcal{I}' \subseteq \mathcal{I}$, let $W(\mathcal{I}') = \{I \in \mathcal{I}' \mid ver(I) \in V_W\}$. One can verify that for any two interval sets $\mathcal{I}', \mathcal{I}'' \subseteq \mathcal{I}$, $U(\mathcal{I}' \setminus \mathcal{I}'') = U(\mathcal{I}') \setminus \mathcal{I}''$ and $W(\mathcal{I}' \setminus \mathcal{I}'') = W(\mathcal{I}') \setminus \mathcal{I}''$ hold, which we will repeatedly use in the following. In Fig. 2-(Right), $U(\mathcal{I}) = \{I_1, I_2, I_3, I_4, I_5, I_8, I_9, I_{11}, I_{12}, I_{14}\}$ and $W(\mathcal{I}) = \{I_6, I_7, I_{10}, I_{13}, I_{15}\}$. We call $\mathcal{I} = \{I_1, \ldots, I_n\}$ the *interval representation* of the convex bipartite graph G.

Consider the interval representation $\mathcal{I} = \{I_1, \ldots, I_n\}$ of a convex bipartite graph $G = (V_U \cup V_W, E)$. Let $S \subseteq V_U \cup V_W$ be a subset of vertices in G and $\mathcal{S} = \{int(v) \mid v \in S\}$ be a subset of intervals in $\mathcal{I}$. For the subset $\mathcal{S} = U(\mathcal{S}) \cup W(\mathcal{S})$, we say that an interval $I \in U(\mathcal{S})$ (respectively, $I \in W(\mathcal{S})$) is *happy with respect to* $\mathcal{S}$ if $I \cap I' = \emptyset$ for any interval $I' \in W(\mathcal{I}) \setminus \mathcal{S}$ (respectively, $I' \in U(\mathcal{I}) \setminus \mathcal{S}$). Hence, a vertex $v \in V_U \cup V_W$ is happy with respect to S if and only if $int(v)$ is happy with respect to $\mathcal{S}$. Recall that an interval $int(w_i)$ of a vertex w_i in V_W may intersect with another interval $int(w_j)$ of another vertex w_j in V_W even if w_i is happy, whereas every interval of a vertex in V_U does not intersect with another interval of a vertex in V_U by the construction of the intervals. Various parts of the discussions in the following trace arguments in [11], and then we obtain lemmas whose statements are the same as ones in [11]. However, the above difference and the convexity of the input graph cause complications in proving several properties/statements compared to [11]; roughly speaking, we need to prove each statement, separately for each of the two sets $U(\mathcal{I})$ and $W(\mathcal{I})$ of intervals, (sometimes, by different arguments).

Let $H(\mathcal{I}'; \mathcal{S})$ be a set of happy intervals in $\mathcal{I}' \subseteq \mathcal{I}$ with respect to $\mathcal{S}$. In the following, we provide a polynomial-time algorithm which can find a subset $\mathcal{S} \subseteq \mathcal{I}$ such that the number of happy vertices in $H(\mathcal{I}; \mathcal{S})$ is maximized for the interval representation $\mathcal{I}$ of the input convex bipartite graph G. Applying a similar argument to one in the proof of Lemma 3 in [11], for two subsets $\mathcal{I}'$ and $\mathcal{S}$ of $\mathcal{I}$, we can show the following (1) inclusion, and moreover (2) equivalence if $\mathcal{I}' \subseteq \mathcal{S}$; the following lemma plays important roles to design our polynomial-time algorithm for MaxHS, and indeed it is frequently used in the remaining:

Lemma 12. *Consider the interval representation $\mathcal{I} = \{I_1, \ldots, I_n\}$ of a convex bipartite graph G. Suppose that $\mathcal{I}'$ and $\mathcal{S}$ are subsets of $\mathcal{I}$. Then, (1) $H(\mathcal{I}; \mathcal{S}) \setminus \mathcal{I}' \subseteq H(\mathcal{I} \setminus \mathcal{I}'; \mathcal{S} \setminus \mathcal{I}')$ holds. (2) Furthermore, if $\mathcal{I}' \subseteq \mathcal{S}$, then $H(\mathcal{I}; \mathcal{S}) \setminus \mathcal{I}' = H(\mathcal{I} \setminus \mathcal{I}'; \mathcal{S} \setminus \mathcal{I}')$ holds.*

Here we introduce additional notation to prove the following lemmas, where we shall use the same notation as in [11]. Suppose that an integer k and the interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of a convex bipartite graph G are given. Then, we add dummy intervals $I_0 = [0, 0]$ and $I_{n+1} = [2n_U + 2, 2n_U + 2]$ into $U(\mathcal{I})$. In the following, we assume that $\mathcal{I}$ has those two dummy intervals I_0 and I_{n+1}, and $\{I_0, I_{n+1}\} \subseteq U(\mathcal{I})$. For an integer $i \in \{0, 1, \ldots, n+1\}$, let $\mathcal{I}_i = \{I_0, I_1, \ldots, I_i\}$ be a subset of the first $i + 1$ intervals and $\overline{\mathcal{I}_i} = \mathcal{I} \setminus \mathcal{I}_i =$

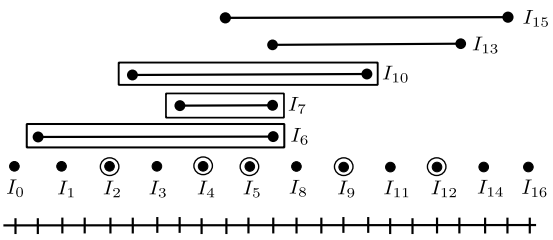

Fig. 3. Let $\mathcal{S} = \{I_2, I_4, I_5, I_6, I_7, I_9, I_{10}, I_{12}\}$ be a solution. Then, when $\mathcal{I}_7$ is observed, I_7 can be decided to be happy, but, the (un)happiness of I_5 cannot be decided yet.

$\{I_{i+1}, I_{i+2}, \ldots, I_{n+1}\}$. Let $\mathcal{I}^-$ be the original interval representation of the input convex bipartite graph G without the two dummy intervals, i.e., $\mathcal{I}^- = \mathcal{I} \setminus \{I_0, I_{n+1}\}$. Let $\mathcal{S}^*$ be a subset of $\mathcal{I}^-$ such that $|H(\mathcal{I}; \mathcal{S}^*)|$ is maximized among all subsets of $\mathcal{I}^-$ of size k. Since both I_0 and I_{n+1} have no intersection with other intervals of $\mathcal{I}$, $\mathcal{S}^*$ is an optimal solution of the original interval representation $\mathcal{I}^-$ of G. Our algorithm is based on the dynamic programming (DP) method. Our basic strategies are as follows: To obtain an optimal solution $\mathcal{S}^*$, we compute a partial solution $\mathcal{S}_i = \mathcal{S}^* \cap \mathcal{I}_i$ for each $i \in \{0, 1, \ldots, n+1\}$, and finally obtain $\mathcal{S}_{n+1} = \mathcal{S}^*$. Here, $\mathcal{S}_i$ is contained in an optimal solution $\mathcal{S}^*$, but it does not mean that $\mathcal{S}_i$ is an optimal set for the (sub)graph whose interval representation is $\mathcal{I}_i$.

Observation 1 *See the interval representation $\mathcal{I}$ of G in Fig. 3, including two dummy intervals I_0 and I_{16}. Suppose for example that a (probably non-optimal) solution $\mathcal{S}$ has $\{I_2, I_4, I_5, I_6, I_7, I_9, I_{10}, I_{12}\}$ of $k = 8$ intervals. (i) Consider a subset $\mathcal{I}_7$ of the eight intervals I_0 through I_7. One sees that we can decide that I_7 must be happy at this time since all the intervals I_4 and I_5 in $U(\mathcal{I})$, which have intersections with I_7 are in $\mathcal{S}$ and furthermore all the intervals in $U(\overline{\mathcal{I}_7}) = \{I_8, I_9, I_{11}, I_{12}, I_{14}, I_{16}\}$ have no intersection with I_7 . Also, for example, I_6 can be decided to be unhappy by checking whether or not there is* at least one *interval in $U(\mathcal{I}_7) \setminus \mathcal{S}$ which intersects with I_7* from the right to the left. *(ii) Consider a larger subset $\mathcal{I}_{10}$ of* 11 *intervals. At this time, the (un)happiness of all intervals in $W(\mathcal{I}_{10})$ can be decided. On the other hand, the (un)happiness of I_9 cannot be decided yet since it depends on $W(\overline{\mathcal{I}_{10}}) = \{I_{13}, I_{15}\}$. Even if I_{13} is in $\mathcal{S}$, I_9 is unhappy if I_{15} is not in $\mathcal{S}$. These observations suggest to us that the left endpoints of intervals in $W(\overline{\mathcal{I}_{10}})$ play important roles.*

From the above observation, in order to correctly compute $\mathcal{S}_i$ for each $i \in \{0, 1, \ldots, n+1\}$ by observing only $\mathcal{I}_i$ in our DP-based algorithm, we keep integers r, ℓ, and k' that satisfy the following three conditions for $\mathcal{S}^*$: (i) The interval $I_r \in U(\mathcal{I})$ has the largest right endpoint among all intervals in $U(\mathcal{I}_i) \setminus \mathcal{S}^*$, that is, $I_r \notin \mathcal{S}^*$ and $right(I_j) \leq right(I_r)$ for any $I_j \in U(\mathcal{I}_i) \setminus \mathcal{S}^*$[4]. (ii) The interval $I_\ell \in W(\mathcal{I})$ has the smallest left endpoint among all intervals in $W(\overline{\mathcal{I}_i}) \setminus \mathcal{S}^*$,

[4] Recall that $left(I) = right(I)$ for $I \in U(\mathcal{I})$.

that is, $I_\ell \notin \mathcal{S}^*$ and $left(I_\ell) \leq left(I_j)$ for any $I_j \in W(\overline{\mathcal{I}_i}) \setminus \mathcal{S}^*$. (iii) $|\mathcal{S}_i| = k'$. In the condition (i), we did not define $I_r \in \mathcal{I}$, but $I_r \in U(\mathcal{I})$. As will be seen later, we need to check whether such an interval I_r intersects with an interval in $W(\mathcal{I})$. Since, by definition of the happiness of an interval, we ignore intersections between two intervals in $W(\mathcal{I})$. Thus, in order to be specific, we define $I_r \in U(\mathcal{I})$. The reason why we define $I_\ell \in W(\mathcal{I})$ in the condition (ii) is similar: We need to check whether I_ℓ intersects with an interval in $U(\mathcal{I})$. Since two intervals in $U(\mathcal{I})$ do not intersect, I_ℓ is defined to be in $W(\mathcal{I})$. The roles of I_r and I_ℓ will get clearer below.

We say that a quadruple (i, r, ℓ, k') of integers is *compatible* with $\mathcal{S}^*$ if i, r, ℓ, and k' satisfy the above three conditions (i), (ii), and (iii). For the three integers i, r, and ℓ, let $\overrightarrow{\mathcal{I}_r} = \{I_{i'} \in I_i \mid right(I_r) < right(I_{i'})\}$ and $\mathcal{I}_{i,\ell} = \mathcal{I}_i \cup \{I_\ell\}$ for short. Again, see Fig. 3 as an example. For $\mathcal{I}_i$ and $\mathcal{S}^* = \{I_2, I_4, I_5, I_6, I_7, I_9, I_{10}, I_{12}\}$, $\ell = 15$ and hence $\mathcal{I}_{7,15} = \{I_0, I_1, \ldots, I_7, I_{15}\}$ since $left(I_{15}) < left(I_{13})$.

Consider a convex bipartite graph G and its interval representation $\mathcal{I}$. Let $\mathcal{S}^*$ be an optimal solution of the original interval representation $\mathcal{I}^-$ such that $|\mathcal{S}^*| = k$. Suppose that a quadruple (i, r, ℓ, k') of integers is compatible with $\mathcal{S}^*$. Note that $H(\mathcal{I}; \mathcal{S}^*) \cap \mathcal{I}_{i,\ell} = H(\mathcal{I}; \mathcal{S}^*) \setminus (\mathcal{I} \setminus \mathcal{I}_{i,\ell})$ holds. Also, $\mathcal{S}_i = S^* \cap \mathcal{I}_i = \mathcal{S}^* \cap \mathcal{I}_{i,\ell}$ holds since $I_\ell \notin \mathcal{S}^*$. Therefore, by using Lemma 12(1) we can obtain the following lemma on $H(\mathcal{I}; \mathcal{S}^*)$:

Lemma 13. *$H(\mathcal{I}; \mathcal{S}^*) \cap \mathcal{I}_{i,\ell} \subseteq H(\mathcal{I}_{i,\ell}; \mathcal{S}_i)$.*

Now, suppose that $\mathcal{S}_i^\star$ is an optimal solution of $\mathcal{I}_{i,\ell}$ such that $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}_i^\star$, $I_0, I_r, I_\ell \notin \mathcal{S}_i^\star$, and $|\mathcal{S}_i^\star| = k'$. Let $S^\star = (\mathcal{S}^* \setminus \mathcal{S}_i) \cup S_i^\star$. Then, we can show that if an interval I is in $H(\mathcal{I}_{i,\ell}; \mathcal{S}_i^\star)$, then it must be in $H(\mathcal{I}; \mathcal{S}^\star) \cap \mathcal{I}_{i,\ell}$, i.e., we have:

Lemma 14. *$H(\mathcal{I}_{i,\ell}; \mathcal{S}_i^\star) \subseteq H(\mathcal{I}; \mathcal{S}^\star) \cap \mathcal{I}_{i,\ell}$ holds.*

Similarly, we obtain the following inclusion:

Lemma 15. *$H(\mathcal{I}; \mathcal{S}^*) \setminus \mathcal{I}_{i,\ell} \subseteq H(\mathcal{I}; \mathcal{S}^\star) \setminus \mathcal{I}_{i,\ell}$ holds.*

From Lemmas 13, 14, and 15, we obtain the following lemma:

Lemma 16. *For the interval representation $\mathcal{I}$ of a convex bipartite graph G and integer k, let $\mathcal{S}^*$ be an optimal solution of the original interval representation $\mathcal{I}^-$ such that $|\mathcal{S}^*| = k$. Also, let $\mathcal{S}_i = \mathcal{S}^* \cap \mathcal{I}_i$. Suppose that a quadruple (i, r, ℓ, k') of integers is compatible with $\mathcal{S}^*$, and $\mathcal{S}_i^\star$ is an optimal solution of $\mathcal{I}_{i,\ell}$ such that $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}_i^\star \subseteq \mathcal{I}_{i,\ell} \setminus \{I_0, I_r, I_\ell\}$ and $|\mathcal{S}_i^\star| = k'$. Then, there is an optimal solution $\mathcal{S}^\star = (\mathcal{S}^* \setminus \mathcal{S}_i) \cup \mathcal{S}_i^\star$ of $\mathcal{I}^-$ (where $\mathcal{S}^\star$ could be the same as $\mathcal{S}^*$).*

The above Lemma 16 enables us to maintain a partial solution for each I_i, $0 \leq i, \leq n$, in the proposed DP-algorithm: we first guess a quadruple (i, r, ℓ, k') of integers that is compatible with $\mathcal{S}^*$, and then find a partial solution $\mathcal{S}$ satisfying three conditions (i) $|\mathcal{S}| = k'$, (ii) $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}$, and (iii) $|H(\mathcal{I}_{i,\ell}; \mathcal{S})|$ is the maximum among all subsets of $\mathcal{I}_{i,\ell} \setminus \{I_0, I_r, I_\ell\}$. The details will be given in the next subsection.

3.2 A dynamic programming algorithm

Our algorithm uses the following three functions $h_{in}(\mathcal{I}_{i,\ell}, r, k')$, $h_{out}(\mathcal{I}_{i,\ell}, r, k')$ and $h_{max}(\mathcal{I}_{i,\ell}, k')$, where i, r, ℓ, and k' are integers such that $0 \leq r \leq i < \ell \leq n+1$ and $0 \leq k' \leq k$: (i) $h_{in}(\mathcal{I}_{i,\ell}, r, k')$ returns the maximum of $|H(\mathcal{I}_{i,\ell}, \mathcal{S})|$ among all subsets $\mathcal{S} \subseteq \mathcal{I}_{i,\ell} \setminus \{\mathcal{I}_0, \mathcal{I}_r, \mathcal{I}_\ell\}$ such that $I_i \in \mathcal{S}$, $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}$, and $|\mathcal{S}| = k'$. (ii) $h_{out}(\mathcal{I}_{i,\ell}, r, k')$ returns the maximum of $|H(\mathcal{I}_{i,\ell}, \mathcal{S})|$ among all subsets $\mathcal{S} \subseteq \mathcal{I}_{i,\ell} \setminus \{\mathcal{I}_0, \mathcal{I}_r, \mathcal{I}_\ell\}$ such that $I_i \notin \mathcal{S}$, $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}$, and $|\mathcal{S}| = k'$. (iii) $h_{max}(\mathcal{I}_{i,\ell}, k')$ returns the maximum of $|H(\mathcal{I}_{i,\ell}, \mathcal{S})|$ among all subsets $\mathcal{S} \subseteq \mathcal{I}_{i,\ell} \setminus \{\mathcal{I}_0, \mathcal{I}_\ell\}$ such that $|\mathcal{S}| = k'$. Remark that the difference between h_{in} and h_{out} is that h_{in} and h_{out} include $I_i \in \mathcal{S}$ and $I_i \notin \mathcal{S}$ in their conditions on the ith interval I_i, respectively. [5]

Now we focus on n_U intervals in $U(\mathcal{I})$; let $U(\mathcal{I}) = \{I_{\mu_1}, I_{\mu_2}, \ldots, I_{\mu_{n_U}}\}$ such that $\mu_j < \mu_{j+1}$ for $1 \leq j \leq n_U - 1$. For each i, $R(i)$ denotes the largest index such that $I_{\mu_{R(i)}} \in U(\mathcal{I}_i)$, i.e., $I_{\mu_{R(i)}} = I_i$ if $I_i \in U(\mathcal{I})$, and $I_{\mu_{R(i)}} \in U(\mathcal{I}_i)$ but $I_{\mu_{R(i)+1}} \notin U(\mathcal{I}_i)$ if $I_i \in W(\mathcal{I})$. Then, r must be (at least) in $\{\mu_{R(i)} - k, \mu_{R(i)} - k + 1, \ldots, \mu_{R(i)}\}$ if $\mu_{R(i)} - k > 0$, and in $\{0, \mu_1, \mu_2, \ldots, \mu_{R(i)}\}$ if $\mu_{R(i)} - k \leq 0$ (where I_0 is the leftmost dummy interval). Otherwise, it implies $|\mathcal{S}| > k'$, which violates the condition $|\mathcal{S}| = k'$ when we compute $h_{in}(\mathcal{I}_{i,\ell}, r, k')$, $h_{out}(\mathcal{I}_{i,\ell}, r, k')$, or $h_{max}(\mathcal{I}_{i,\ell}, k')$. For simplicity, let $L(i)$ be $\max\{0, R(i) - k\}$.

Let $h_{in}(\mathcal{I}_{i,\ell}, r, k') = -\infty$, $h_{out}(\mathcal{I}_{i,\ell}, r, k') = -\infty$, and $h_{max}(\mathcal{I}_{i,\ell}, k') = -\infty$ if there exists no subset $\mathcal{S}$ that satisfies all the prescribed conditions for h_{in}, h_{out} and h_{max}, respectively. We compute values $h_{in}(\mathcal{I}_{i,\ell}, r, k')$, $h_{out}(\mathcal{I}_{i,\ell}, r, k')$, and $h_{max}(\mathcal{I}_{i,\ell}, k')$ by the DP method on the lexicographic order of a quadruple (i, r, ℓ, k'). Finally, we obtain the value $h_{max}(\mathcal{I}_{n,(n+1)}, k)$, which is the maximum size of $H(\mathcal{I}; \mathcal{S})$ such that $\mathcal{S} \subseteq \mathcal{I}^-$ and $|\mathcal{S}| = k$.

For each triple (i, ℓ, k') of integers, $h_{max}(\mathcal{I}_{i,\ell}, k')$ is obtained as follows:

$$h_{max}(\mathcal{I}_{i,\ell}, k') = \max_{r \in \{\mu_{L(i)}, \ldots, \mu_{R(i)}\}} \Big\{ h_{in}(\mathcal{I}_{i,\ell}, r, k'), h_{out}(\mathcal{I}_{i,\ell}, r, k') \Big\}.$$

Below we describe how to compute the three functions h_{in}, h_{out}, and h_{max}. Some parts of the following are very similar to corresponding arguments in [11] for interval graphs. However, we do not omit such arguments for completeness of the description of the proposed algorithm in this paper.

(IN) We first consider the recursive computation of $h_{in}(\mathcal{I}_{i,\ell}, r, k')$, by distinguishing two cases **(Case 1-1)** $I_i \in U(\mathcal{I})$ and **(Case 1-2)** $I_i \in W(\mathcal{I})$. Note that if $k' = 0$, then we set $h_{in}(\mathcal{I}_{i,\ell}, r, 0) = -\infty$ for any i, ℓ, and r since there is no subset $\mathcal{S}$ such that $I_i \in \mathcal{S}$ and $|\mathcal{S}| = 0$.

(Case 1-1) Suppose that $I_i \in \mathcal{S}$ and $I_i \in U(\mathcal{I})$. When $i = 0$, the leftmost dummy interval I_0 must not be in $\mathcal{S}$, and r must be 0, since $r \leq i$. Therefore, for any $k' \geq 0$ and any $\ell > 0$, we set $h_{in}(\mathcal{I}_{0,\ell}, 0, k') = -\infty$. Moreover, $h_{in}(\mathcal{I}_{(n+1),\ell}, r, k')$ is undefined for any ℓ, r, and k', since an imposed condition $(n+1 =)\ i < \ell \leq n+1$ cannot be satisfied. Thus, we here assume that $1 \leq i \leq n$.

[5] The dynamic programming algorithm for interval graphs in [11] uses two functions, while we need three functions for convex bipartite graphs in the paper.

Recall that for every interval $I \in \mathcal{I}_{i-1}$, $right(I) < left(I_i)$ if $I_i \in U(\mathcal{I})$. Hence the (un)happiness of I_i depends only on I_ℓ. That is, if I_i intersects with I_ℓ, then I_i is unhappy; otherwise, I_i is happy. Here we do not care about I_r since $I_r \in U(\mathcal{I})$ and hence I_r does not intersect with I_i. Therefore, we compute $h_{in}(\mathcal{I}_{i,\ell}, r, k')$ from $h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'-1)$ and $h_{out}(\mathcal{I}_{(i-1),\ell}, r, k'-1)$ by deciding whether I_i is happy or not:

$$h_{in}(\mathcal{I}_{i,\ell}, r, k') = \begin{cases} \max\left\{h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'-1), h_{out}(\mathcal{I}_{(i-1),\ell}, r, k'-1)\right\} & \text{if } I_i \cap I_\ell \neq \emptyset; \\ \max\left\{h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'-1), h_{out}(\mathcal{I}_{(i-1),\ell}, r, k'-1)\right\} + 1 & \text{otherwise.} \end{cases}$$

(Case 1-2) Suppose that $I_i \in \mathcal{S}$ and $I_i \in W(\mathcal{I})$. Recall that two dummy intervals I_0 and I_{n+1} are in $U(\mathcal{I})$, and I_1 is always in $U(\mathcal{I})$ by the construction of the interval representation. Therefore, we can assume $2 \leq i \leq n$ in this case. Let $\mathcal{S}$ be a subset of $\mathcal{I}^- \setminus \{I_r\}$ such that $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}$, $|\mathcal{S}| = k'$, and $|H(\mathcal{I}_{i,\ell}; \mathcal{S})|$ is maximized. Since $H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_i\} = H(\mathcal{I}_{(i-1),\ell}; \mathcal{S}) \setminus \{I_i\}$, we compute $h_{in}(\mathcal{I}_{i,\ell}, r, k')$ from $h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'-1)$ and $h_{out}(\mathcal{I}_{(i-1),\ell}, r, k'-1)$ by deciding whether the ith interval I_i is happy with respect to $\mathcal{S}$. If $I_i \in W(\mathcal{I})$ intersects with $I_r \notin \mathcal{S}$, then I_i is unhappy; otherwise, I_i is happy. Here, we do not care about I_ℓ, since $I_\ell \in W(\mathcal{I})$ and hence whether $I_i \cap I_\ell \neq \emptyset$ or not is unrelated to the happiness of I_i. That is, we have:

$$h_{in}(\mathcal{I}_{i,\ell}, r, k') = \begin{cases} \max\left\{h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'-1), h_{out}(\mathcal{I}_{(i-1),\ell}, r, k'-1)\right\} & \text{if } I_i \cap I_r \neq \emptyset; \\ \max\left\{h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'-1), h_{out}(\mathcal{I}_{(i-1),\ell}, r, k'-1)\right\} + 1 & \text{otherwise.} \end{cases}$$

Remark that the above formula is very similar to the one in **(Case 1-1)**, but the condition $I_i \cap I_r \neq \emptyset$ is different from the one $I_i \cap I_\ell \neq \emptyset$ in **(Case 1-1)**.

(OUT) We next consider the recursive computation of $h_{out}(\mathcal{I}_{i,\ell}, r, k')$, by distinguishing two cases **(Case 2-1)** $I_i \in U(\mathcal{I})$ and **(Case 2-2)** $I_i \in W(\mathcal{I})$. In the following, let $\mathcal{S}$ be a subset of $\mathcal{I}_{i,\ell} \setminus \{I_0, I_r, I_i, I_\ell\}$ such that $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}$, $|\mathcal{S}| = k'$, and $|H(\mathcal{I}_{i,\ell}; \mathcal{S})|$ is maximized.

(Case 2-1) Suppose that $I_i \notin \mathcal{S}$ and $I_i \in U(\mathcal{I})$. First we show $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i-1,\ell}; \mathcal{S})$. Since $I_i \notin \mathcal{S}$, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_i\}$ holds. Lemma 12(1) gives $H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_i\} \subseteq H(\mathcal{I}_{i,\ell} \setminus \{I_i\}; \mathcal{S} \setminus \{I_i\}) = H(\mathcal{I}_{i-1,\ell}; \mathcal{S})$. Namely, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) \subseteq H(\mathcal{I}_{i-1,\ell}; \mathcal{S})$ holds. Since I_i does not intersect with any interval in $\mathcal{I}_{i-1}$, $H(\mathcal{I}_{i,\ell}; \mathcal{S})$ includes every interval in $H(\mathcal{I}_{i-1,\ell}; \mathcal{S})$, i.e., $H(\mathcal{I}_{i-1,\ell}; \mathcal{S}) \subseteq H(\mathcal{I}_{i,\ell}; \mathcal{S})$ holds. As a result, we have $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i-1,\ell}; \mathcal{S})$.

Observe that r in $h_{out}(\mathcal{I}_{i,\ell}, r, k')$ must be i by the assumption $U(\overrightarrow{\mathcal{I}_r}) \subseteq \mathcal{S}$ of h_{out} and $I_i \notin \mathcal{S}$. Therefore, we can calculate $h_{out}(\mathcal{I}_{i,\ell}, r, k')$ as follows:

$$h_{out}(\mathcal{I}_{i,\ell}, r, k') = \begin{cases} h_{max}(\mathcal{I}_{(i-1),\ell}, k') & \text{if } r = i; \\ -\infty & \text{otherwise.} \end{cases}$$

(Case 2-2) Suppose that $I_i \notin \mathcal{S}$ and $I_i \in W(\mathcal{I})$. Again, we can assume that $2 \leq i \leq n$. We further consider the following two cases: (i) $left(I_i) < left(I_\ell)$, and (ii) $left(I_i) \geq left(I_\ell)$.

(i) Suppose that $left(I_i) < left(I_\ell)$. We show $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i-1,i}; \mathcal{S})$ holds. First, we show the following inclusion $H(\mathcal{I}_{i,\ell}; \mathcal{S}) \subseteq H(\mathcal{I}_{i-1,i}; \mathcal{S})$. From the definition $I_\ell \notin \mathcal{S}$, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_\ell\}$ holds. Lemma 12(1) gives the following inclusion: $H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_\ell\} \subseteq H(\mathcal{I}_{i,\ell} \setminus \{I_\ell\}; \mathcal{S} \setminus \{I_\ell\}) = H(\mathcal{I}_i; \mathcal{S})$. Since $\mathcal{I}_{(i-1),i} = \mathcal{I}_i$, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) \subseteq H(\mathcal{I}_{i-1,i}; \mathcal{S})$ holds. Next, we show that $H(\mathcal{I}_{i-1,i}; \mathcal{S}) \subseteq H(\mathcal{I}_{i,\ell}; \mathcal{S})$. The assumption $I_i \notin \mathcal{S}$ means that $right(I') < left(I_i)$ for any interval $I' \in H(\mathcal{I}_{i-1,i}; \mathcal{S})$, i.e., I' intersects with neither I_i nor I_ℓ. Therefore, $I' \in H(\mathcal{I}_{i,\ell}; \mathcal{S})$. It follows that $H(\mathcal{I}_{(i-1),i}; \mathcal{S}) \subseteq H(\mathcal{I}_{i,\ell}; \mathcal{S})$. Thus, in this case, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{(i-1),i}; \mathcal{S})$ holds.

(ii) Suppose that $left(I_i) \geq left(I_\ell)$. A very similar argument to the above (i) can be made: Since $I_i \notin \mathcal{S}$, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_i\}$ holds. Therefore, by Lemma 12(1), $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{i,\ell}; \mathcal{S}) \setminus \{I_i\} \subseteq H(\mathcal{I}_{i,\ell} \setminus \{I_i\}; \mathcal{S} \setminus \{I_i\}) = H(\mathcal{I}_{(i-1),\ell}; \mathcal{S})$. Next, the assumption $I_\ell \notin \mathcal{S}$ means that $right(I') < left(I_\ell)$ for any interval $I' \in H(\mathcal{I}_{(i-1),\ell}; \mathcal{S})$, i.e., I' intersects with neither I_i nor I_ℓ. Therefore, $I' \in H(\mathcal{I}_{i,\ell}; \mathcal{S})$, i.e., $H(\mathcal{I}_{i-1,i}; \mathcal{S}) \subseteq H(\mathcal{I}_{i,\ell}; \mathcal{S})$ holds. Thus, $H(\mathcal{I}_{i,\ell}; \mathcal{S}) = H(\mathcal{I}_{(i-1),\ell}; \mathcal{S})$ holds for this case too.

From (i) and (ii), we can calculate $h_{out}(\mathcal{I}_{i,\ell}, r, k')$ as follows:

$$h_{out}(\mathcal{I}_{i,\ell}, r, k') = \begin{cases} \max\left\{h_{in}(\mathcal{I}_{(i-1),i}, r, k'), h_{out}(\mathcal{I}_{(i-1),i}, r, k')\right\} & \text{if } left(I_i) < left(I_\ell), \\ \max\left\{h_{in}(\mathcal{I}_{(i-1),\ell}, r, k'), h_{out}(\mathcal{I}_{(i-1),\ell}, r, k')\right\} & \text{if } left(I_i) \geq left(I_\ell). \end{cases}$$

Running time. We bound the running time of our algorithm for convex bipartite graphs. As mentioned before, given a convex bipartite graph G, the convex ordering of G can be computed in $O(n+m)$ time [6, 14]. The interval representation $\mathcal{I}$ of G can be obtained in $O(n+m)$ time. We can sort intervals in $\mathcal{I}$ in increasing order based on their right endpoints by an $O(n)$-time integer sorting algorithm, e.g., [10], since each right endpoint is an integer ranging from 0 to $2n+1$. For each integer $i \in \{0, \ldots, n+1\}$, we construct $\mathcal{I}_i$ and $\overline{\mathcal{I}_i}$ in $O(n^2)$ time.

We then bound the running time to compute three values $h_{in}(\mathcal{I}_{i,\ell}, r, k')$, $h_{out}(\mathcal{I}_{i,\ell}, r, k')$, and $h_{max}(\mathcal{I}_{i,\ell}, k')$ for each integers i, r, ℓ and k'. Recall that $i \in \{0, \ldots, n+1\}$ and $k' \in \{0, \ldots, k\}$. Also, r is in $\{\mu_{L(i)}, \ldots, \mu_{R(i)}\}$ of at most $k+1$ integers. One sees that for each i, $R(i)$ and hence $L(i)$ can be obtained in $O(n)$ time, by looking at $U(\mathcal{I}_i)$ and I_i. Hence, in total, we can obtain all $R(i)$'s and $L(i)$'s in $O(n^2)$ time. Let us consider ℓ. For each i, the interval I_ℓ can be chosen from a set $\mathcal{I}' \subseteq \overline{\mathcal{I}_i}$ of size at most $k+1$, where $\mathcal{I}'$ consists of the first $k+1$ intervals of $\overline{\mathcal{I}_i}$ sorted in increasing order of their left endpoints; otherwise, from the definition of I_ℓ, an optimal solution $\mathcal{S}^*$ of $\mathcal{I}$ has size at least $k+1$, a contradiction.

Let (i, r, ℓ, k') be a quadruple of those integers. For all quadruples (i, r, ℓ, k'), the values $h_{in}(\mathcal{I}_{i,\ell}, r, k')$ and $h_{out}(\mathcal{I}_{i,\ell}, r, k')$ in each case can be computed in

$O(k^3n)$ time. Also, for all triples (i, r, k'), $h_{max}(\mathcal{I}_{i,\ell}, k')$ can be computed in $O(k^3n)$ time. Finally, by taking value $h_{max}(\mathcal{I}_{n,n+1}, k)$ in $O(1)$ time, we obtain the maximum number of happy vertices for the given convex bipartite graph G and integer k. The total running time of our algorithm is $O(n^2 + k^3n)$.

4 Concluding remarks

For MaxHS, we designed a polynomial-time 2-approximation algorithm for cubic graphs and a polynomial-time algorithm for convex bipartite graphs. Some open problems are listed as follows: (i) Are faster approximation/exact algorithms possible for cubic graphs and convex bipartite graphs? (ii) Can we design approximation algorithms with a smaller approxixmation ratio than 2 for cubic graphs and $(2\Delta + 1)$ for general graphs? (iii) Is it possible to design polynomial-time algorithms for other graph classes? (iv) Can we show any inapproximability of MaxHS?

Acknowledgments. The work was partially supported by the NSERC Canada, and JSPS KAKENHI Grant Numbers JP22K11915 and JP24K02902.

References

1. Akanksha Agrawal, N. R. Aravind, Subrahmanyam Kalyanasundaram, Anjeneya Swami Kare, Juho Lauri, Neeldhara Misra, and I. Vinod Reddy. Parameterized complexity of happy coloring problems. *Theor. Comput. Sci.*, 835:58–81, 2020.
2. N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. Linear time algorithms for happy vertex coloring problems for trees. In Veli Mäkinen, Simon J. Puglisi, and Leena Salmela, editors, *IWOCA 2016*, volume 9843 of *LNCS*, pages 281–292. Springer, 2016.
3. Yuichi Asahiro, Hiroshi Eto, Tesshu Hanaka, Guohui Lin, Eiji Miyano, and Ippei Terabaru. Complexity and approximability of the happy set problem. *Theoretical Computer Science*, 866:123–144, 2021.
4. Yuichi Asahiro, Hiroshi Eto, Tesshu Hanaka, Guohui Lin, Eiji Miyano, and Ippei Terabaru. Parameterized algorithms for the happy set problem. *Discrete Applied Mathematics*, 304:32–44, 2021.
5. Yuichi Asahiro, Hiroshi Eto, Tesshu Hanaka, Guohui Lin, Eiji Miyano, and Ippei Terabaru. Corrigendum to "complexity and approximability of the happy set problem" [Theor. Comput. Sci. 866 (2021) 123-144]. *Theor. Comput. Sci.*, 975:114114, 2023.
6. Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
7. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.
8. Gerth Stølting Brodal, Loukas Georgiadis, Kristoffer Arnsfelt Hansen, and Irit Katriel. Dynamic matchings in convex bipartite graphs. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007,*

32nd International Symposium, MFCS 2007, volume 4708 of *LNCS*, pages 406–417. Springer, 2007.

9. Jayesh Choudhari and I. Vinod Reddy. On structural parameterizations of happy coloring, empire coloring and boxicity. In M. Sohel Rahman, Wing-Kin Sung, and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*, volume 10755 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2018.
10. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press, 2022.
11. Hiroshi Eto, Takehiro Ito, Eiji Miyano, Akira Suzuki, and Yuma Tamura. Happy set problem on subclasses of co-comparability graphs. *Algorithmica*, 85(11):3327–3347, 2023.
12. Fred W. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistics Quarterly*, 14:313–316, 1967.
13. Raymond Greenlaw and Rossella Petreschi. Cubic graphs. *ACM Computing Surveys*, 27(4):471–495, 1995.
14. Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000.
15. David Easley Jon Kleinberg. *Networks, Crowds, and Markets ; Reasoning about a Highly Connected World*. Cambridge University Press, 2010.
16. Rhyd Lewis, Dhananjay R. Thiruvady, and Kerri Morgan. Finding happiness: An analysis of the maximum happy vertices problem. *Comput. Oper. Res.*, 103:265–276, 2019.
17. Y. Daniel Liang and Maw-Shang Chang. Minimum feedback vertex sets in co-comparability graphs and convex bipartite graphs. *Acta Informatica*, 34:337–346, 1997.
18. Yosuke Mizutani and Blair D. Sullivan. Parameterized complexity of maximum happy set and densest k-subgraph. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
19. José Soares and Marco Aurelio Stefanes. Algorithms for maximum independent set in convex bipartite graphs. *Algorithmica*, 53(1):35–49, 2009.
20. George Steiner and Julian Scott Yeomans. A linear time algorithm for maximum matchings in convex, bipartite graphs. *Computers & Mathematics With Applications*, 31:91–96, 1996.
21. Peng Zhang and Angsheng Li. Algorithmic aspects of homophyly of networks. *Theoretical Computer Science*, 593:117–131, 2015.
22. Peng Zhang, Yao Xu, Tao Jiang, Angsheng Li, Guohui Lin, and Eiji Miyano. Improved approximation algorithms for the maximum happy vertices and edges problems. *Algorithmica*, 80(5):1412–1438, 2018.