Corrigendum to "Complexity and approximability of the happy set problem" [Theoret. Comput. Sci. 866(2021) 123–144]

Yuichi Asahiro* Hiroshi Eto† Tesshu Hanaka‡
Guohui Lin§ Eiji Miyano† Ippei Terabaru†

August 2, 2023

Abstract

For a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, a vertex is happy if all its neighbor vertices in G are contained in S. Given a connected undirected graph and an integer k, the Maximum Happy Set Problem (MaxHS) asks to find a set S of k vertices which maximizes the number of happy vertices in S (note that all happy vertices in V belong to S). We proposed an algorithm for MaxHS on proper interval graphs in Theoretical Computer Science, 866(2021), 123–144. However, due to a wrong observation made by the authors, it works only on proper interval graphs obeying the observation. In this corrigendum, we propose a new algorithm which runs in $O(k|V| \log k + |E|)$ time for proper interval graphs.

Keywords happy set, proper interval graph, dynamic programming

1 Introduction

In this section, we recall several definitions in [1] for completeness of this corrigendum, and then explain the error contained in [1].

1.1 Definitions

Let $G = (V, E)$ be a connected undirected graph, where V and E denote the set of vertices and the set of edges, respectively. $V(G)$ and $E(G)$ also denote the vertex set and the edge set of G. Throughout this corrigendum, let $n = |V|$ and $m = |E|$ for any given graph. We denote an edge with endpoints u and v by $\{u, v\}$. The closed neighborhood of a vertex v in G is denoted by $N[v] = \{v\} \cup \{u \in V \mid \{u, v\} \in E\}$. For a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, a vertex v is *happy* (*with respect to* S) if $N[v] \subseteq S$.

*Corresponding author. Department of Information Science, Kyushu Sangyo University, 2-3-1 Matsukadai, Higashi-ku, Fukuoka 813-8503, Japan. asahiro@is.kyusan-u.ac.jp.
†Kyushu Institute of Technology, Fukuoka, Japan
‡Kyushu University, Fukuoka, Japan
§University of Alberta, Edmonton, Canada

The Maximum Happy Set problem (MaxHS) is defined as follows.

Maximum Happy Set problem (MaxHS)

Input: An undirected graph $G = (V, E)$ and an integer k

Goal: Find a subset $S \subseteq V$ of k vertices such that the number $\#h(S)$ of happy vertices is maximized.

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. For a subset C of $V(G)$, if every pair of vertices in C are adjacent in $G[C]$, then $G[C]$ or C is called a *clique*. A clique is *maximal* if it is not contained in any other clique.

Given an undirected graph $G = (V, E)$ and two non-adjacent vertices u and v, a subset $S \subset V$ is a (u, v)-*separator* if the removal of S separates u and v in distinct connected components. If no proper subset of S is a (u, v)-separator, then S is a *minimal* (u, v)-*separator*. Conversely, if there exists a pair of non-adjacent vertices u and v such that $S' \subset V$ is a (minimal) (u, v)-separator, then S' is just called a *(minimal) separator*.

For an interval I of real line, $l(I)$ and $r(I)$ are the left endpoint and the right endpoint of I, respectively. A graph G is an *interval graph* if there exists an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ such that there is an edge $\{i, j\}$ between two vertices i and j in G if and only if I_i and I_j intersect, i.e., $I_i \cap I_j \neq \emptyset$. An interval graph G with interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ is a *proper interval graph* if $I_i \not\subseteq I_j$ and $I_j \not\subseteq I_i$ for any $1 \leq i < j \leq n$. For simplicity, we use l_j and r_j instead of $l(I_j)$ and $r(I_j)$ when an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ is given.

1.2 The error

The error in [1] is related to a clique tree of an input graph. Let $G = (V, E)$ be a proper interval graph. The clique graph of G, denoted by $G_c = (V_c, E_c, w)$, with $w : E_c \to \{1, 2, \ldots, n\}$ is defined as follows [3][1]:

1. The vertex set V_c is the set of maximal cliques of G, i.e., a vertex c_i in G_c corresponds to a maximal clique C_i in G.

2. For two vertices $c_i, c_j \in V_c$, the edge $\{c_i, c_j\}$ belongs to E_c if and only if the intersection $V(C_i) \cap V(C_j)$ is a minimal (u, v)-separator for every pair of $u \in (V(C_i) \setminus V(C_j))$ and $v \in (V(C_j) \setminus V(C_i))$.

3. The edge $\{c_i, c_j\} \in E_c$ is weighted by the number of vertices of the corresponding minimal separator $U_{i,j}$, i.e., $w(c_i, c_j) = |U_{i,j}|$.

A clique tree of a proper interval graph G is a maximum-weight spanning tree of the clique graph of G [3]. In addition, if G is a proper interval graph, then G has a clique tree which is a simple path [6]. Such a clique tree is called a *clique path*. Thus, in [1], we assumed that the input graph G consists of t maximal cliques C_1 through C_t, and G has a clique path $P = (V_P, E_P)$ with $V_P = \{c_1, c_2, \ldots, c_t\}$ and $E_P = \{\{c_1, c_2\}, \{c_2, c_3\}, \ldots, \{c_{t-1}, c_t\}\}$, where

[1] Precisely, the clique graph is defined on chordal graphs containing (proper) interval graphs as a subset. However, since this corrigendum considers proper interval graphs only, the definitions here and in the later part are only described for (proper) interval graphs.

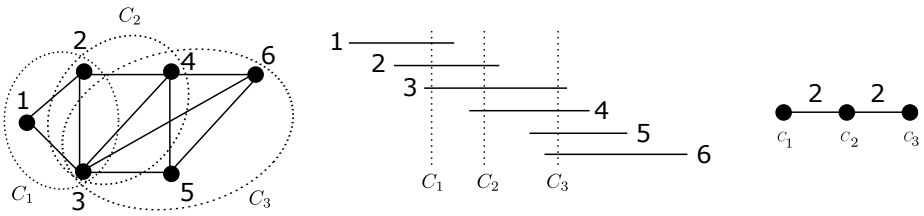

Figure 1: A proper interval graph G with vertex set $\{1, 2, 3, 4, 5, 6\}$ (left), an interval representation of G (middle), and a clique path of G (right). There are three maximal cliques C_1, C_2, and C_3 on $\{1, 2, 3\}$, $\{2, 3, 4\}$, and $\{3, 4, 5, 6\}$, respectively. The separators are $S_1 = \{2, 3\}$ and $S_2 = \{3, 4\}$, and hence $S_1 \cap S_2 = \{3\} \neq \emptyset$.

the vertex c_i corresponds to the maximal clique C_i for $1 \leq i \leq t$. Let S_i be a minimal separator of two cliques C_i and C_{i+1} for $1 \leq i \leq t-1$, that is $S_i = V(C_i) \cap V(C_{i+1})$.

The proposed algorithm for proper interval graphs in [1] was based on the clique path and minimal separators of the input. In [1], we made the following observation:

Error 1 (line 3 on page 139 in [1]) *$S_i \cap S_{i+1} = \emptyset$ holds for every i,*

However, this was incorrect. See Fig. 1 for an example, in which this observation does not hold. Since the algorithm for proper interval graphs was designed based on this observation, i.e., it was incorrect, we would like to propose a new algorithm which runs in $O(kn \log k + m)$ time for proper interval graphs in this corrigendum.

1.3 MaxHS on interval representation

The set of intervals that intersect an interval I_i is denoted by $N[I_i]$. For a set S of intervals, we say that an interval I_i is *happy* (*with respect to* S) if $N[I_i] \subseteq S$. On the contrary, an interval I is *unhappy* (*with respect to* S) if I is not happy (with respect to S). Using these notations, the goal of MaxHS on proper interval graphs can be interpreted as finding a set S of k intervals from an interval representation of a given proper interval graph such that the number $\#h(S)$ of happy intervals is maximized. Thus we first obtain an interval representation from an input proper interval graph, and then find an optimal set of intervals, although the original paper [1] tried to design an algorithm on the input graphs without using interval representations.

1.4 Organization

In Section 2, we first give two basic properties on interval representations with several notations. The proposed algorithm and its correctness are described in Section 3. Then, Section 4 analyzes the time complexity of the proposed algorithm. Finally, we conclude this corrigendum in Section 5.

2 Preliminaries

Given a proper interval graph, the proposed algorithm works on its interval representation. The following proposition states the time complexity of obtaining an interval representation from a proper interval graph.

Proposition 1 ([5]) *For a proper interval graph G, an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of G can be computed in $O(n + m)$ time. Moreover, the obtained interval representation $\mathcal{I}$ satisfies the following condition: $l_i < l_j$ for $1 \leq i < j \leq n$.*

The next proposition describes a property related to the right endpoints of intervals in the interval representation obtained by Proposition 1.

Proposition 2 *The interval representation $\mathcal{I}$ obtained in Proposition 1 also satisfies the following condition: $r_i < r_j$ for $1 \leq i < j \leq n$.*

Proof Assume for contradiction that $r_i \geq r_j$. Since $l_i < l_j$ from Proposition 1, $I_j \subseteq I_i$. This contradicts the assumption that G is a proper interval graph. Thus, $r_i < r_j$ holds. □

From the above two propositions, we assume that an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of a proper interval graph satisfies $l_i < l_j$ and $r_i < r_j$ for $1 \leq i < j \leq n$. For an interval I_i in an interval representation $\mathcal{I}$, $L(i)$ and $R(i)$ respectively denote the indices of the leftmost and the rightmost intervals intersecting I_i. Namely, intervals $I_{L(i)}, \ldots, I_{R(i)}$ intersect I_i, where $L(i) \leq i \leq R(i)$ holds. Note that $L(i)$ and/or $R(i)$ may be i. Then $N[I_i]$ is the set of intervals $\{I_{L(i)}, \ldots, I_{R(i)}\}$. By Propositions 1 and 2, $L(R(x)) = x$ and $R(L(x)) = x$ hold.

3 Proposed algorithm

3.1 Overview

The overview of the proposed algorithm is as follows.

Step 1. Obtain an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ from an input proper interval graph with n vertices and m edges.

Step 2. Compute $L(i)$ and $R(i)$ for $1 \leq i \leq n$.

Step 3. The maximum number of happy intervals is computed by a dynamic programming method.

Step 1 can be done in $O(n+m)$ time by Proposition 1. The details of Steps 2 and 3 will be given in the following subsections. We will present a procedure for Step 3, which only computes the maximum number of happy intervals for simplicity. An optimal set of intervals and hence an optimal set of vertices that corresponds to the maximum value can be found by adding a trace-back step without increasing the procedure's time complexity.

3.2 Step 2

We show the following lemma, whose proof is constructive and gives a procedure for Step 2.

Lemma 1 *Step 2 can be done in $O(n)$ time.*

Proof First set $L(1) = 1$ in $O(1)$ time. Then, run the following procedure.

Step 2-1. Let $i = 1$ and $j = 2$.

Step 2-2. If $l_j \leq r_i$, then set $L(j) = i$ and increment j. Otherwise (i.e., in the case $l_j > r_i$), set $R(i) = j - 1$ and increment i.

Step 2-3. If $j \leq n$, go to Step 2-2.

Step 2-4. Set $R(i') = n$ for $i \leq i' \leq n$

The two indices i and j always satisfy $i \leq j$ during the execution of the above procedure; if $i = j$, the condition $l_j \leq r_i = r_j$ holds and j is incremented in Step 2-2. By Step 2-2 and the condition of Step 2-3, when Step 2-4 starts, all of $L(1), \ldots, L(n)$ have been determined. On the other hand, only $R(1), \ldots, R(i-1)$ have been determined in Step 2-2, and hence we prepare Step 2-4. The correctness of the values $L(i)$ and $R(i)$ for every i is obvious from Propositions 1 and 2.

Let us estimate the running time of the above procedure. Each of Steps 2-1, 2-2, and 2-3 can be done in $O(1)$ time. One execution of Step 2-2, increments i or j. Hence Step 2-2 is repeated at most $2n = O(n)$ times. Thus the total time spent by Steps 2-2 and 2-3 is $O(n)$. Since Step 2-4 also takes $O(n)$ time, the total running time of the above procedure is $O(n)$. □

3.3 Step 3

We describe the dynamic programming method used in Step 3 of the proposed algorithm.

3.3.1 Notation

Given an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, define $\mathcal{I}_i = \bigcup_{j=1}^{i} I_j$. For a set S of intervals, let $S_i = S \setminus \{I_{i+1}, \ldots, I_n\}$. With respect to S_i, we represent the status of an interval I_i by three letters $\overline{s}$, u, and s as follows.

- $\overline{s}$: $I_i \notin S_i$, and hence I_i is unhappy with respect to S_i.
- u: $I_i \in S_i$, however I_i is unhappy with respect to S_i
- s: $I_i \in S_i$. Whether I_i is happy or not with respect to S could not be determined only by S_i , i.e., it will be determined based on how $I_{i+1}, \ldots, I_n$ are selected into S. The important property in this case is that $N[I_i] \cap \mathcal{I}_i \subseteq S_i$ should hold. As an exception, when we decide $I_n \in S_i$, whether I_n is happy or not with respect to S is determined, since $S_n = S$.

Then, we simply say that I_i *is* c *(with respect to* S_i*)* for $c \in \{\overline{s}, u, s\}$ if the condition of I_i is represented by c with respect to S_i. The condition $\overline{s}$, u, or s of an interval I_i with respect to S_i is denoted by $c(I_i)$.

For $1 \leq i \leq n$, $0 \leq \ell \leq k$, and $c \in \{\overline{s}, u, s\}$, we define $H(i, \ell, c)$ to be the maximum number of happy intervals with respect to S_i under the conditions $|S_i| = \ell$ and I_i is c. For $i \leq 0$ or $\ell < 0$, we define $H(i, \ell, c) = -\infty$ for every $c \in \{\overline{s}, u, s\}$. Step 3 of the proposed algorithm computes

$$\max_{c \in \{\overline{s}, u, s\}} H(n, k, c) \tag{1}$$

in a dynamic programming manner.

Along with $c(I_i)$, let $h_i(I_j)$ for $1 \leq j \leq i \leq n$ represent the happiness of I_j with respect to S, by three letters: u, h, or y.

- u: I_j is already determined **u**nhappy with respect to S, only by S_i. This happens when $c(I_j) = \overline{s}$ or $c(I_j) = u$.
- h: I_j is already determined **h**appy with respect to S, only by S_i. This happens only when $c(I_j) = s$.
- y: Whether I_j is happy or not with respect to S has not been determined **y**et, only by S_i. This happens only when $c(I_j) = s$.

This $h_i(I_j)$ is only used for explanation. Observe that $h_n(I_j) = h_{R(j)}(I_j)$ for any $1 \leq j \leq n$, since I_j overlaps $I_{L(i)}, \ldots, I_{R(i)}$. Thus, in order to express that I_j is determined happy with respect to S, we will increment $H(R(j), \ell, s)$ and $H(R(j), \ell, u)$ when every $c(I_{L(j)}), \ldots, c(I_{R(j)})$ is s or u.

3.3.2 Initialization: $i = 1$

Consider $\mathcal{I}_1 = \{I_1\}$. For any candidate solution S, I_1 is $\overline{s}$ or s with respect to S_i, since all the neighbors of I_1 is in $S \setminus S_i$. Thus, we set 0 for these valid cases, and $-\infty$ for the other cases, to each corresponding entry of the table H:

$$\begin{aligned}
H(1, 0, \overline{s}) &= 0, \\
H(1, 1, \overline{s}) &= -\infty, \\
H(1, 1, s) &= 0, \\
H(1, 0, s) &= -\infty, \\
H(1, 0, u) &= -\infty, \\
H(1, 1, u) &= -\infty, \text{ and} \\
H(1, \ell, c) &= -\infty \text{ for } 2 \leq \ell \leq k \text{ and } c \in \{\overline{s}, u, s\}.
\end{aligned}$$

For each pair of ℓ and c, the above can be done in $O(1)$ time. Since the number of pairs of ℓ and c is $3(k+1)$, we have the following lemma.

Lemma 2 *It takes* $O(k)$ *time to fill* $H(1, \ell, c)$ *for* $0 \leq \ell \leq k$ *and* $c \in \{\overline{s}, u, s\}$.

3.3.3 $i \geq 2$ and $c(I_i) = \overline{s}$

Consider the case that $c(I_i) = \overline{s}$. Since the intervals $I_1, \ldots, I_{L(i)-1}$ do not overlap I_i, $h_i(I_j)$ is independent of $c(I_i)$ for $1 \leq j \leq L(i) - 1$. As for $L(i) \leq j \leq$

$i-1$, $h_i(I_j) = u$, regardless of $h_{i-1}(I_j)$: if $h_{i-1}(I_j) = y$, $h_i(I_j)$ turns to be u, and, if $h_{i-1}(I_j) = u$, then clearly $h_i(I_j) = u$. Moreover, $h_{i-1}(I_j)$ must be u or y, since whether $I_{R(j)}$ is included in S or not is unclear when considering S_{i-1}, and hence we can not conclude that I_j is happy. In summary, the number of happy vertices does not increase if $c(I_i) = \overline{s}$. Hence, we set as follows.

$$H(i, \ell, \overline{s}) = \max\{H(i-1, \ell, \overline{s}),\ H(i-1, \ell, u),\ H(i-1, \ell, s) \tag{2}$$

For each pair of i and ℓ, the above can be computed in $O(1)$ time. Since the number of pairs of i and ℓ is $n(k+1)$, we have the following lemma.

Lemma 3 *It takes $O(kn)$ time to fill $H(i, \ell, \overline{s})$ for $1 \leq i \leq n$ and $0 \leq \ell \leq k$.*

3.3.4 $i \geq 2$ and $c(I_i) = u$

As in the case $c(I_i) = \overline{s}$, since the intervals $I_1, \ldots, I_{L(i)-1}$ do not overlap I_i, $h_i(I_j)$ is independent of $c(I_i)$ for $1 \leq j \leq L(i) - 1$.

Considering I_i, the case $c(I_i) = u$ can happen only if (at least) one of $I_{L(i)}, \cdots, I_{i-1}$ is not included in S_i. Let $j = \max\{j' \mid I_{j'} \notin S_i,\ L(i) \leq j' < i\}$. Namely, I_j is the rightmost interval which is not in S_i and overlaps I_i, and then $I_{j+1}, \ldots, I_i$ are included in S_i. Since $I_j \notin S_i$ and $I_j \cap I_{i'} \neq \emptyset$ for $L(i) \leq i' \leq i-1$, $h_i(I_{L(i)}) = \cdots = h_i(L_{i-1}) = u$. Thus, by selecting I_i into the solution S (or S_i), the number of happy vertices does not increase. (Note that the purpose of selecting $I_{j+1}, \ldots, I_i$ into the solution is to make some intervals from $I_{i+1}, \ldots, I_{R(j)}$ happy.) Therefore, $H(i, \ell, u)$ is estimated by $H(j, \ell - (i-j), \overline{s})$, where $i - j$ is the number of intervals $I_{j+1}, \ldots, I_i$.

Since $\ell \leq k$, we only need to consider the index j satisfying $i - j \leq k$, i.e., $i - k \leq j$. Here, $i-k$ might be negative, and hence $\max\{i-k, 1, L(i)\} \leq j \leq i-1$ is the range of j to be considered. By letting $i' = \max\{i-k, 1, L(i)\}$, the recursive formula is as follows.

$$H(i, \ell, u) = \max_{i' \leq j \leq i-1} \{H(j, \ell - (i-j), \overline{s})\} \tag{3}$$

Since the number of candidates for j is $O(k)$, we can compute the above in $O(k)$ time for each pair of i and ℓ.

Lemma 4 *It takes $O(k^2 n)$ time to fill $H(i, \ell, u)$ for $1 \leq i \leq n$ and $0 \leq \ell \leq k$.*

3.3.5 $i \geq 2$ and $c(I_i) = s$

Similar to the above two subsections, since the intervals $I_1, \ldots, I_{L(i)-1}$ do not overlap I_i, $h_i(I_j)$ is independent of $c(I_i)$ for $1 \leq j \leq L(i) - 1$.

The case $c(I_i) = s$ happens only when all of $I_{L(i)}, \ldots, I_{i-1}$ are selected into a solution S_i. Let $j = \max\{j' \mid I_{j'} \notin S_i,\ j' < L(i)\}$, i.e., I_j is the rightmost interval which is not in S_i. Namely, $i - j$ intervals $I_{j+1}, \ldots, I_i$ are included in S_i. Note that $R(j) < L(i)$ holds. Here, it is sufficient to consider j satisfying $i - j \leq k$, i.e., $i - k \leq j$, since $i - j$ intervals $I_{j+1}, \ldots, I_i$ are included in S_i. To cope with the case that $i - k$ is negative, the range of j to be considered is $i' \leq j \leq L(i) - 1$, where $i' = \max\{i - k, 1\}$.

Since $I_j \notin S_i$, $h_i(I_{j+1}) = \cdots = h_i(I_{R(j)}) = u$. If $R(j) < L(i+1)$ $(\leq i)$, $h_i(I_{R(j)+1}) = \cdots = h_i(I_{L(i+1)-1}) = h$, since $I_{L(R(j)+1)}, \ldots, I_{R(L(i+1)-1)}$ are

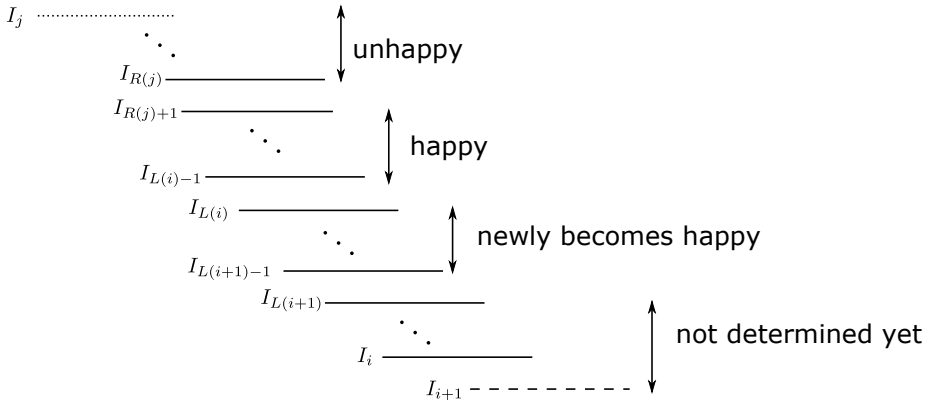

Figure 2: The case $R(j) < L(i+1)$. Selecting I_i makes the intervals $I_{L(i)}, \ldots, I_{L(i+1)-1}$ happy under the condition that $I_j \notin S_i$ and whether I_j is included into the solution has been determined.

included in S_i, where $j < L(R(j)+1)$ and $R(L(i+1)-1) \leq i$. Among them, $h_i(I_{R(j)+1}) = \cdots = h_i(I_{L(i)-1})$ have been already determined until when considering S_{i-1}, and $I_{L(i)}, \ldots, I_{L(i+1)-1}$ newly become happy by selecting I_i into the solution, Then, $h_i(I_{L(i+1)}) = \cdots = h_i(I_i) = y$. See Figure 2. If $R(j) \geq L(i+1)$, $h_i(I_{R(j)+1}) = \cdots = h_i(I_i) = y$ (whether they are happy or not will be determined by I_{i+1} and later).

Let us introduce a variable $x_{i,j}$ for each pair of i and j such that $x_{i,j} = 1$ if $R(j) < L(i+1)$, and $x_{i,j} = 0$ otherwise. Using this $x_{i,j}$, the recursive formula is as follows.

$$H(i,\ell,s) = \max_{i' \leq j \leq L(i)-1} \{H(i,\ell-(i-j),\overline{s}) + x_{i,j} \cdot (L(i+1) - L(i) - 1)\} \quad (4)$$

Since $L(i+1)$ and $R(j)$ have already been computed in Step 2 of the algorithm, $x_{i,j}$ can be computed in $O(1)$ time. Then, since the number of candidates for j is $O(k)$, the above can be computed in $O(k)$ time for each pair of i and ℓ.

Lemma 5 *It takes $O(k^2 n)$ time to fill $H(i,\ell,s)$ for $1 \leq i \leq n$ and $0 \leq \ell \leq k$.*

4 Running time

The number of entries in the table H is $O(kn)$, and each entry stores one integer at most k (the possible maximum number of happy vertices). Then, from Proposition 1 and Lemmas 1, 2, 3, 4, and 5, we can show in a straightforward way that the proposed algorithm runs in $O(k^2 n + m)$ time.

We improve the time complexity by a careful implementation of Steps 2 and 3. Concretely, we reduce the $O(k)$-time computing the formulas (3) and (4) for each triple (i,ℓ,c), where $c \in \{u,s\}$, to $O(\log k)$-time as follows. We compute (2) before computing (3) and (4), and then apply the following.

The formula (3): First suppose that $i \geq k+1$. To compute $H(i,\ell,u)$, we need $H(j,\ell-(i-j),\overline{s})$ for $i-k \leq j \leq i-1$, i.e., $H(i-1,\ell-1,\overline{s}), H(i-2,\ell-2,\overline{s}), \ldots, H(i-k,\ell-k,\overline{s})$. Similarly, computing $H(i+1,\ell+1,u)$, we

need $H(i,\ell,\overline{s}), H(i-1,\ell-1,\overline{s}), \ldots, H(i-k+1,\ell-k+1,\overline{s})$. That is, $H(i-1,\ell-1,\overline{s}), \ldots, H(i-k+1,\ell-k+1,\overline{s})$ are used for computing $H(i,\ell,u)$ and $H(i+1,\ell+1,u)$.

Based on the above observation, for each pair (i,ℓ), we prepare (maintain) a red-black tree [4] storing (at most) k values of $H(i-1,\ell-1,\overline{s}), H(i-2,\ell-2,\overline{s}), \ldots, H(i-k,\ell-k,\overline{s})$. To compute (3), we pick the maximum value in the red-black tree. Then, as maintenance of the red-black tree, we delete $H(i-k,\ell-k,\overline{s})$ from the red-black tree, and then insert $H(i,\ell,\overline{s})$ into the red-black tree, both of which can be done in $O(\log k)$ time. Note that two or more same values as $H(i-k,\ell-k,\overline{s})$ may exist in the red-black tree, but we can remove any one of them. In order to compute $H(i+1,\ell+1,u)$, we use this updated red-black tree, similarly.

Consider the next case that $i \leq k$. In this case, during the maintenance of the red-black tree, only insertion of $H(i,\ell,\overline{s})$ is done, while we do not delete $H(i-k,\ell-k,\overline{s})$ (which does not exist since $i \leq k$).

To fill the table $H(i,\ell,s)$ for $1 \leq i \leq n$ and $0 \leq \ell \leq k$, we first construct one red-black tree for each pair of i and ℓ for $i = 1$ or $\ell = 1$, i.e., $(i,\ell) = (1,1),(1,2),\ldots,(1,k),(2,1),(3,1),\ldots,(n,1)$ ($O(k+n) = O(n)$ trees in total), and maintain them for pairs $(1+1,1+1) = (2,2)$, $(1+1,2+1) = (2,3)$, and so on. Thus, we can compute (3) in $O(\log k)$ time. Since the number of nodes and each stored value are both $O(k)$ in these red-black trees, the total additional space needed is $O(kn\log k)$, which equals to the required space for the table H having $O(kn)$ entries of value (at most) k.

The formula (4): We just store the values of $H(i,\ell-(i-j),\overline{s}) + x_{i,j} \cdot (L(i+1)-L(i)-1)$ in red-black trees, instead of $H(i,\ell-(i-j),\overline{s})$. By a similar argument to the above, (4) can be computed in $O(\log k)$ time.

In summary, both of (3) and (4) can be computed in $O(\log k)$ time for each pair of i and ℓ, and then we have the following theorem.

Theorem 1 *The proposed algorithm runs in $O(kn\log k + m)$ time.*

5 Conclusion

We proposed an $O(kn\log k + m)$-time algorithm for MaxHS on proper interval graphs, correcting an error in [1]. As for interval graphs, we proposed an $O(kn^8)$-time algorithm in [1]. Very recently, Eto, Ito, Miyano, Suzuki, and Tamura designed an $O(k^3n + n^2)$-time algorithm for interval graphs [2].

Declaration of competing interest

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

Acknowledgments

This work is supported in part by JSPS KAKENHI Grant Numbers JP17K00024, JP21H05852, JP21K11755, JP21K17707, JP22H00513, JP22K11915, and JP23H04388.

References

[1] Y. Asahiro, H. Eto, T. Hanaka, G. Lin, E. Miyano, I. Terabaru. Complexity and approximability of the happy set problem, Theoretical Computer Science, **866**(2021), 123–144.

[2] H. Eto, T. Ito, E. Miyano, A. Suzuki, Y. Tamura. Happy set problem on subclasses of co-comparability graphs, Algorithmica (2022). https://doi.org/10.1007/s00453-022-01081-0.

[3] P. Bernstein, N. Goodman. Power of natural semijoins. SIAM J. Comput. **10**(4)(1981), 751–771.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Chapter 13, Red-black trees, in *Algorithm Introduction, fourth edition*, The MIT Press, 2022

[5] D.G. Corneil, H. Kim, S. Natarajan, S. Olariu, A.P. Sprague. Simple linear time recognition of unit interval graphs, *IPL*, **55**(1995), 99–104.

[6] P. Gilmore, A. Hoffman. A characterization of comparability graphs and of interval graphs. Can. J. Math, **16**(1964), 539–548.