Linear Time Algorithms for Happy Vertex Coloring Problems for Trees

N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare(✉)

Department of Computer Science and Engineering Indian Institute of Technology, Hyderabad, India
{aravind,subruk,cs14resch01002}@iith.ac.in

Abstract. Given an undirected graph $G = (V, E)$ with $|V| = n$ and a vertex coloring, a vertex v is *happy* if v and all its neighbors have the same color. An edge is *happy* if its end vertices have the same color. Given a partial coloring of the vertices of the graph using k colors, the *Maximum Happy Vertices* (also called k-MHV) problem asks to color the remaining vertices such that the number of happy vertices is maximized. The *Maximum Happy Edges* (also called k-MHE) problem asks to color the remaining vertices such that the number of happy edges is maximized. For arbitrary graphs, k-MHV and k-MHE are NP-Hard for $k \geq 3$. In this paper we study these problems for trees. For a fixed k we present linear time algorithms for both the problems. In general, for any k the proposed algorithms take $O(nk \log k)$ and $O(nk)$ time respectively.

Keywords: Happy vertex · Happy edge · Graph coloring · Coloring trees

1 Introduction

Graph coloring problems are well studied in literature. The traditional vertex coloring problem asks to color the vertices of the graph using minimum number of colors such that the adjacent vertices get different colors. There are many variants of coloring problems. Recently, Zhang and Li [10] studied a coloring problem in which adjacent vertices are allowed to get same color. The proposed problems have applications related to homophyly in networks (see Chapter 4 of [4]).

Given an undirected graph $G = (V, E)$ and a vertex coloring, a vertex is *happy* if the vertex and all its adjacent vertices have the same color and *unhappy* otherwise. An edge is *happy* if its end vertices have the same color and *unhappy* otherwise.

For $S \subseteq V$, let $c_p : S \to \{1, 2, \ldots, k\}$ be a partial vertex coloring. A coloring $c_f : V \to \{1, 2, \ldots, k\}$ is an *extended full coloring* for c_p, if $c_f(v) = c_p(v), \forall v \in S$.

A.S. Kare—Faculty member of University of Hyderabad. This work is carried out as part of his PhD program at IIT Hyderabad.

V. Mäkinen et al. (Eds.): IWOCA 2016, LNCS 9843, pp. 281–292, 2016.
DOI: 10.1007/978-3-319-44543-4_22

Given an $S \subseteq V$ and a partial coloring c_p, *Maximum Happy Vertices* (MHV) (respectively, *Maximum Happy Edges* (MHE)) problem asks to find an extended full coloring c such that the number of happy vertices (respectively, edges) is maximized. As k is also an input parameter, the problem is also referred to as k-MHV (respectively, k-MHE).

Definition 1. *Multiway-Cut*

(Instance) We are given an undirected graph $G = (V, E)$ and a terminal set $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$.

(Goal) Find a set of edges $C \subseteq E$ with minimum cardinality whose removal disconnects all the terminals from each other.

Definition 2. *Multiway-Uncut*

(Instance) We are given an undirected graph $G = (V, E)$ and a terminal set $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$.

(Goal) Find a partition $\{V_1, V_2, \ldots, V_k\}$ of V such that each partition contains exactly one terminal and the number of edges not cut by the partition is maximized.

The k-MHE problem is a generalization of the Multiway Uncut problem [7] which is the complement of Multiway Cut problem [1,2]. The Multiway Uncut problem is a special case of k-MHE problem in which there is exactly one pre-colored vertex (terminal) for each color.

Both k-MHV and k-MHE problems are NP-Hard [10] for $k \geq 3$ for arbitrary graphs. In [10], $O(mn^7 \log n)$ and $O(\min\{n^{\frac{2}{3}}m, m^{\frac{3}{2}}\})$ time algorithms are presented for 2-MHV and 2-MHE respectively. Towards this end, the authors of [10] used techniques such as minimizing sub modular functions (2-MHV) [6] and max-flow algorithms (2-MHE) [5]. Zhang and Li [10] presented approximation algorithms with approximation ratios $\max\{\frac{1}{k}, \Omega(\Delta^{-3})\}$ and $\frac{1}{2}$ for k-MHV and k-MHE respectively. Here, Δ is the maximum degree of the graph. Later, Zhang et al. [9] presented approximation algorithms with approximation ratios $\frac{1}{\Delta+1}$ and $(\frac{1}{2} + \frac{\sqrt{2}}{4}f(k)) \geq 0.8535$ for k-MHV and k-MHE respectively.

1.1 Our Results

Apart from the results in [9,10], the MHV and MHE problems does not seem to be addressed for any class of graphs. In this paper, we study these problems for trees. We propose dynamic programming based algorithms for both k-MHV and k-MHE. For an arbitrary k, the proposed algorithms take $O(nk \log k)$ and $O(nk)$ time respectively. When k is fixed, the algorithms run in linear time. We also extend our algorithms to generate all the optimal colorings of the tree. Generating each optimal coloring takes polynomial time.

Using the result from [2] we observe that, for an arbitrary k, the k-MHE problem is NP-Hard for planar graphs. Using the result from [3] we infer that, when the number of pre-colored vertices is bounded, the k-MHE problem can be solved in linear time for graphs with bounded branch width.

The rest of the paper is organized as follows: In Sect. 2 we discuss the algorithm for the k-MHV problem, in Sect. 3 we discuss the algorithm for the k-MHE problem and the related observations. We conclude with Sect. 4. Throughout the paper we assume that the input graph is a tree (T). We use integers from 1 to k to denote the colors.

2 Algorithm for k-MHV Problem

We root the tree at an arbitrary vertex. Let T_v denotes the subtree rooted at a vertex v. Before presenting the algorithm we give a simple reduction rule, which can be executed in linear time.

Rule 1: If a leaf vertex is uncolored, remove it and count the leaf vertex as happy.

We can give the color of its parent to the uncolored leaf to make it happy. Hence, without loss of generality we can assume that all the leaves are colored.

We process the vertices of the rooted tree according to post order traversal. At each vertex v, we maintain a list of $2k$ integer values. The maximum value of these $2k$ values gives the maximum number of happy vertices in T_v, the sub tree rooted at v. The maximum value of the $2k$ values associated with the root gives us the maximum number of happy vertices of the tree. The corresponding optimal coloring can also be traced back in reverse direction. The list of $2k$ values defined as follows, for $1 \leq i \leq k$:

- $T_v[i, H]$: The maximum number of happy vertices in the subtree T_v, when v is colored i and is happy in T_v. That is, when v and all its children are colored i. Note that, here we focus on v being happy in the subtree T_v. The vertex v can become unhappy in the tree T because its parent gets another color.
- $T_v[i, U]$: The maximum number of happy vertices in T_v, when v is colored i and is unhappy in T_v. That is, when one or more children of v are colored with a color other than i.

Note that, if a vertex or some of its children are already colored, then some of the $2k$ values are invalid. We use -1 to denote an invalid value. We keep these $2k$ values in an array to access any specific item in constant time. The values are indexed in the order, $T_v[1, H]$, $T_v[1, U]$, $T_v[2, H]$, $T_v[2, U]$, $\ldots$, $T_v[k, H]$, $T_v[k, U]$.

The following expressions are defined to simplify some of the equations:

- $T_v[i, *]$: The maximum number of happy vertices in the subtree T_v, when v is colored i. v may be happy or unhappy. That is:

$$T_v[i, *] = \max\{T_v[i, H], T_v[i, U]\}. \tag{1}$$

- $T_v[i, -]$: The maximum number of happy vertices in T_v excluding v, when v is colored i.

$$T_v[i, -] = \max\{T_v[i, H] - 1, T_v[i, U]\}. \tag{2}$$

– $T_v[\bar{i}, *]$: The maximum number of happy vertices in the subtree T_v, when v is colored with color other than i.

$$T_v[\bar{i}, *] = \max_{r \neq i}\{T_v[r, *]\}. \tag{3}$$

– $T_v[\bar{i}, -]$: The maximum number of happy vertices in the subtree T_v excluding v, when v is colored with color other than i.

$$T_v[\bar{i}, -] = \max_{r \neq i}\{T_v[r, -]\}. \tag{4}$$

– $T_v[*, *]$: The maximum number of happy vertices in T_v. That is:

$$T_v[*, *] = \max\{T_v[1, *], T_v[2, *], \ldots, T_v[k, *]\}. \tag{5}$$

Now we explain the process to compute these $2k$ values at each vertex. As a leaf vertex is pre-colored, it is always happy alone as a subtree with a single vertex. Only one out of $2k$ values is valid. Suppose the color of the leaf is i, then the only valid value is $T_v[i, H] = 1$.

The following subsections consider the case when v is a non leaf vertex. Let $v_1, v_2, \ldots, v_d$ be the children of v. The values $T_v[i, H]$ and $T_v[i, U]$ are invalid, if v is pre-colored with a color $r \neq i$. Otherwise, we compute $T_v[i, H]$ and $T_v[i, U]$ as follows:

2.1 Computing $T_v[i, H]$

Computing $T_v[i, H]$ has two cases:

Algorithm 1. Computing $T_v[i, H]$

```
1: procedure ComputeTvH(v, i)
2:     if ∀v_j, T_{v_j}[i, *] ≠ −1 then
3:         return (1 + Σ_{v_j} T_{v_j}[i, *])          ▷ Case 2
4:     else
5:         return −1                                   ▷ Case 1
6:     end if
7: end procedure
```

Case 1: For some child v_j, $T_{v_j}[i, *] = -1$.
This means that the child v_j is pre colored with a color other than i. In this case, v becomes unhappy when it gets color i. So $T_v[i, H]$ is invalid.
Case 2: For every child v_j, $T_{v_j}[i, *] > -1$.
In this case, we use the following equation to compute $T_v[i, H]$.

$$T_v[i, H] = 1 + \sum_{v_j} T_{v_j}[i, *]. \tag{6}$$

Algorithm 2. Computing $T_v[i, U]$

procedure ComputeTvU(v, i)
  if every child v_j is pre-colored with color i **then**
    return -1 ▷ Case 1
  else if $\exists v_{j'}$ child of v such that $T_{v_{j'}}[*, *] \neq T_{v_{j'}}[i, *]$ **then**
    return $(\sum_{v_j} \max\{T_{v_j}[1, -], \ldots, T_{v_j}[i, *], \ldots, T_{v_j}[k, -]\})$ ▷ Case 2
  else ▷ Case 3
    for each child v_j **do**
      $\text{diff}(v_j, i) \leftarrow T_{v_j}[i, *] - T_{v_j}[\bar{i}, -]$
    end for
    $v_\ell \leftarrow \text{argmin}_{v_j} \text{diff}(v_j, i)$
    $q \leftarrow \text{argmax}_{r \neq i} T_{v_\ell}[r, -]$
    return $(T_{v_\ell}[q, -] + \sum_{v_j \neq v_\ell} T_{v_j}[i, *])$
  end if
end procedure

2.2 Computing $T_v[i, U]$

Computing $T_v[i, U]$ has three cases:

Case 1: Every child v_j is pre colored with color i.
In this case, we cannot make v unhappy by giving color i to v. Hence $T_v[i, U]$ is invalid.

Case 2: For some child $v_{j'}$, $T_{v_{j'}}[*, *] \neq T_{v_{j'}}[i, *]$.
That is, the child $v_{j'}$ has color $r \neq i$ in the optimal coloring of $T_{v_{j'}}$. When v is colored i and $v_{j'}$ is colored r, irrespective of the colors of the other children, v will certainly be unhappy. In this case, we use the following expression to compute $T_v[i, U]$.

$$T_v[i, U] = T_{v_{j'}}[r, -] + \sum_{\substack{v_j \text{ child of } v, \\ v_j \neq v_{j'}}} \max\{T_{v_j}[1, -], \ldots, T_{v_j}[i, *], \ldots, T_{v_j}[k, -]\} \tag{7}$$

$$= \sum_{v_j \text{ child of } v} \max\{T_{v_j}[1, -], \ldots, T_{v_j}[i, *], \ldots, T_{v_j}[k, -]\}. \tag{8}$$

Case 3: For every child v_j, $T_{v_j}[*, *] = T_{v_j}[i, *]$.
For each v_j, if we pick $T_{v_j}[i, *]$, v will become happy, but we need v to be unhappy. To avoid this situation, for some child we pick a value with color other than i as follows:

For each v_j, we define $\text{diff}(v_j, i)$ as follows:

$$\text{diff}(v_j, i) = T_{v_j}[i, *] - T_{v_j}[\bar{i}, -]. \tag{9}$$

We pick the child (say v_ℓ) with minimum diff(v_j, i) value. Suppose, $T_{v_\ell}[\bar{i}, -] = T_{v_\ell}[q, -]$, we replace $T_{v_\ell}[i, *]$ with $T_{v_\ell}[q, -]$. The new expression is:

$$T_v[i, U] = T_{v_\ell}[q, -] + \sum_{v_j \neq v_\ell} T_{v_j}[i, *]. \tag{10}$$

Algorithm 3. Algorithm for MHV problem

```
 1: for each v ∈ V in post order do
 2:     for i = 1 to k do
 3:         if v is a leaf then
 4:             if color(v) = i then
 5:                 T_v[i, H] ← 1
 6:                 T_v[i, U] ← −1
 7:             else
 8:                 T_v[i, H] ← −1
 9:                 T_v[i, U] ← −1
10:             end if
11:         else
12:             if v is pre-colored and color(v) ≠ i then
13:                 T_v[i, H] ← −1
14:                 T_v[i, U] ← −1
15:             else
16:                 T_v[i, H] ← ComputeTvH(v, i)
17:                 T_v[i, U] ← ComputeTvU(v, i)
18:             end if
19:         end if
20:     end for
21: end for
```

Theorem 1. *There is an $O(nk \log k)$ time algorithm for the k-MHV problem for trees.*

Proof. We evaluate the time spent at a particular vertex v to compute $T_v[i, H]$ and $T_v[i, U]$, for $1 \leq i \leq k$. Let $v_1, v_2, \ldots, v_d$ be the children of v.

Computing $T_v[i, H]$: The $T_{v_j}[i, H]$ and $T_{v_j}[i, U]$ values are accessible in constant time for each child v_j. Time to compute $T_v[i, H]$, $\forall 1 \leq i \leq k$ is:

$$\sum_{1 \leq i \leq k} O(d) = O(kd). \tag{11}$$

Computing $T_v[i, U]$: We sort the $2k$ values in descending order. For any child v_j, $T_{v_j}[i, *]$ is available in constant time from the original array. From the sorted array $T_{v_j}[*, *]$ and $T_{v_j}[\bar{i}, *]$ are available in constant time. Hence $T_v[i, U]$, $\forall 1 \leq i \leq k$ can be computed in:

$$O(dk \log k) + \sum_{1 \leq i \leq k} O(d) = O(dk \log k). \tag{12}$$

Hence the total time is:

$$\sum_v dk + dk \log k \leq \sum_v 2dk \log k = 2k \log k \sum_v d = O(nk \log k). \quad (13)$$

□

The correctness of the value $T_v[*,*]$ for every vertex v implies the correctness of the algorithm. The correctness of the value $T_v[*,*]$ follows from the correctness of the $2k$ values $T_v[1,H]$, $T_v[1,U]$, $T_v[2,H]$, $T_v[2,U]$, ..., $T_v[k,H]$, $T_v[k,U]$ associated with v.

Theorem 2. *Algorithm 3 correctly computes the values $T_v[i,H]$ and $T_v[i,U]$ for every v and $1 \leq i \leq k$.*

Proof. We prove the theorem by using induction on the size of the subtrees. For a leaf vertex v, the algorithm correctly computes the values $T_v[i,H]$ and $T_v[i,U]$ for $1 \leq i \leq k$. Since the leaf vertices are pre-colored, each leaf vertex has only one valid value (this value being 1).

For a non-leaf vertex v, let $v_1, v_2, \ldots, v_d$ be the children of v. By induction on the size of the sub-trees, all the $2k$ values associated with each child v_j of v are correctly computed. Let x be the value computed by the algorithm for $T_v[i,H]$ (or $T_v[i,U]$) for any color i. If x is not the optimal value, it will contradict the optimality of at least one value of a child of v. Hence the algorithm correctly computes the values $T_v[i,H]$ and $T_v[i,U]$ for every v and $1 \leq i \leq k$. □

2.3 Generating All Optimal Happy Vertex Colorings

Our algorithm can also be extended to generate all the optimal happy vertex colorings of the tree. Among the $2k$ values associated with a vertex v, there may be multiple values equal to the optimal value. So, while generating optimal happy vertex coloring, we can chose any of these values to generate a different optimal coloring. For example, let $T_v[i,H]$ be an optimal value for the vertex v. Let v_j be a child of v with both $T_{v_j}[i,H]$ and $T_{v_j}[i,U]$ are optimal. So, we can generate one optimal coloring by picking $T_{v_j}[i,H]$ and another optimal coloring by picking $T_{v_j}[i,U]$. There may be exponentially many optimal colorings, but, generating each optimal coloring takes polynomial time (linear time for fixed k).

3 Algorithm for k-MHE Problem

Before presenting the algorithm we give simple reduction rules, which can be executed in linear time.

Rule 2: Let v be a pre-colored vertex with degree more than 1. Let $v_1, v_2, \ldots, v_d$ be the neighbours of v in T. We can divide T into d edge disjoint subtrees $T_1, T_2, \ldots, T_d$ and all these trees share only the vertex v.

$$k\text{-MHE}(T) = k\text{-MHE}(T_1) + k\text{-MHE}(T_2) + \cdots + k\text{-MHE}(T_d). \quad (14)$$

With the application of Rule 2, without loss of generality we can assume that T does not have a pre-colored vertex with degree more than 1.

Now, we root the tree at an arbitrary vertex with degree more than 1.

Rule 3: (Similar to Rule 1 in Sect. 2) If a leaf vertex is uncolored, remove it and count the edge connecting the leaf vertex as happy.

With Rule 2 and Rule 3, without loss of generality, all the leaves of the rooted tree T are pre-colored and no non-leaf vertex is pre-colored.

Our algorithm for k-MHE problem has two phases. In the first phase, we visit the vertices according to post order traversal and populate a list of tentative colors for each vertex. In the second phase we visit the vertices according to pre-order traversal and assign a color for each vertex.

Algorithm 4. Phase 1 of the algorithm

```
1: procedure PopulateTentativeColors(T)
2:     for each v ∈ V in post order do
3:         if v is a leaf then
4:             L(v) ← color(v)
5:         else                                  ▷ Let v_1, v_2, ..., v_d be the children of v
6:             frequency[1..k] ← {0}
7:             for each child v_j of v do
8:                 for each color c ∈ L(v) do
9:                     frequency[c] ← frequency[c] + 1
10:                end for
11:            end for
12:            max ← 0
13:            for i = 1 to k do
14:                if frequency[i] > max then
15:                    max ← frequency[i]
16:                end if
17:            end for
18:            for i = 1 to k do
19:                if frequency[i] = max then
20:                    L(v) ← L(v) ∪ {i}
21:                end if
22:            end for
23:        end if
24:    end for
25: end procedure
```

Phase 1: We visit the vertices according to post order traversal. At each vertex v, we keep a list of tentative colors to assign to the vertex v in the optimal solution. The size of this list is at most k. Let $L(v)$ denote the list of tentative colors associated with the vertex v.

If the vertex v is a leaf, as the leaf vertex is pre-colored, we add that pre-color to $L(v)$. Otherwise, let $v_1, v_2, \ldots, v_d$ be the children of v. The list of tentative colors $L(v_j)$ for each vertex v_j are already computed. For each child v_j, we traverse the list $L(v_j)$ and compute the frequency of occurrences of each color in the multiset that is union of the lists. Let frequency(i) denote the frequency of color i. We add all the colors with maximum frequency to $L(v)$. The process is captured in Algorithm 4.

Algorithm 5. Phase 2 of the algorithm

```
1: procedure AttachColors(T, L)                    ▷ Fixing color to vertices
2:     for each v ∈ V in pre order do
3:         if |L(v)| = 1 then
4:             color(v) ← Only element of L(v)
5:         else if color(parent(v)) ∈ L(v) then
6:             color(v) ← color(parent(v))
7:         else
8:             color(v) ← Any element of L(v)
9:         end if
10:    end for
11: end procedure
```

Phase 2: We visit the vertices according to pre-order traversal to assign a color to each vertex. Let v be the vertex in pre-order. If $|L(v)| = 1$, then we fix the color of v to the only color in $L(v)$. Otherwise, we check if the color of the parent of v is present in $L(v)$, and assign it to v if present. Otherwise, we pick any arbitrary color from $L(v)$ and assign it to v. The process is captured in Algorithm 5.

Theorem 3. *There is an $O(nk)$ time algorithm for the k-MHE problem for trees.*

Proof. At each vertex with degree d, we perform $O(kd)$ time in the Phase 1 and $O(k)$ time in the Phase 2. The time complexity is:

$$\sum_v O(kd) = O(nk). \tag{15}$$

□

The correctness of the algorithm can be proved using induction on the size of the sub-tree similar to Theorem 2.

3.1 Generating All Optimal Happy Edge Colorings

Our algorithm can be extended to generate all the optimal happy edge colorings. We keep a list of tentative colors at each vertex. At a vertex v, if the color(parent(v)) is present in $L(v)$, then, we assign the color(parent(v)) to v in the optimal coloring. Otherwise, we can generate a different optimal coloring for each color in $L(v)$. Here we point out that, this scheme may miss out some optimal colorings when color(parent(v)) is not present in $L(v)$ but present in the set of colors with frequency one less than the maximum frequency. In this case, we can assign the color(parent(v)) to v even though the color(parent(v)) is not present in $L(v)$. A special case of this scenario is when there is a vertex v where all its children have distinct colors (the maximum frequency being 1). Even though the color(parent(v)) not present in $L(v)$, we can assign the color(parent(v)) to v as it has zero frequency at v.

There may be exponentially many optimal happy edge colorings. Generating each optimal coloring takes polynomial time (linear time for fixed k).

3.2 k-MHE for Planar Graphs and Graphs with Bounded Branch Width

The Multiway-Cut problem is NP-Hard for planar graphs [2] when k, the number of terminals, is not fixed. This implies the following theorem on hardness of k-MHE for planar graphs for an arbitrary k.

Theorem 4. *For an arbitrary k, the k-MHE problem is NP-Hard for planar graphs.*

In [8], Robertson and Seymour introduced the notions of tree width and branch width. They showed that these two quantities are always within a constant factor of each other. Many graph problems that are NP-Hard for general graphs have been shown to be solvable in polynomial time for graphs with bounded tree width or equivalently bounded branch width. For more formal definitions of branch width and tree width we refer the readers to [8].

Definition 3. *Multi-Multiway Cut*

(Instance) We are given an undirected graph $G = (V, E)$ and c sets of vertices $S_1, S_2, \ldots, S_c$.

(Goal) Find a set of edges $C \subseteq E$ with minimum cardinality whose removal disconnects every pair of vertices in each set S_i.

When $c = 1$, the Multi-Multiway Cut problem is equivalent to Multiway Cut problem. The k-MHE problem can also be formulated as a Multi-Multiway Cut problem, by creating vertex sets with every pair of pre-colored vertices with different colors. In [3], Deng et al. studied the Multi-Multiway Cut problem for graphs with bounded branch width and presented an $O(b^{2b+2}.2^{2bc}.|G|)$ time algorithm, where b is the branch width of the graph and c is the number of vertex sets. The algorithm runs in linear time when the branch width and the number of vertex sets are fixed.

Theorem 5. *When the branch width of the graph and the number of pre-colored vertices are bounded, there is a linear time algorithm for the k-MHE problem.*

Proof. Let the number of pre-colored vertices be p and the branch width be b. For this instance of k-MHE, we can formulate a Multi-Multiway Cut problem with at most p^2 vertex sets. Hence, the k-MHE problem can be solved in time $O(b^{2b+2}.2^{2bp^2}.|G|)$. Hence, when both the number of pre-colored vertices and the branch width are constants, the k-MHE problem can be solved in linear time. □

4 Conclusions

In this paper, we study the Maximum Happy Vertices (k-MHV) and Maximum Happy Edges (k-MHE) problems for trees. We have presented $O(nk \log k)$ and $O(nk)$ time algorithms for k-MHV and k-MHE problems respectively. Our algorithms run in linear time when k is fixed. Our algorithms can be extended to generate all the optimal colorings of the tree.

As a future direction, it is interesting to study the hardness of the k-MHV problem for planar graphs. For fixed k, the Multiway Cut problem has a polynomial time algorithm for planar graphs [2]. So, for planar graphs and when k is fixed, polynomial time algorithms might be possible for k-MHV and k-MHE. Finding a linear time algorithm for graphs with bounded tree width (branch width) without the constraint on the number of pre-colored vertices is another direction.

Acknowledgement. We thank the anonymous reviewers for their detailed reviews and suggestions.

References

1. Chopra, S., Rao, M.R.: On the multiway cut polyhedron. Networks **21**(1), 51–89 (1991)
2. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiway cuts (extended abstract). In: Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing, STOC 1992, pp. 241–251 (1992)
3. Deng, X., Lin, B., Zhang, C.: Multi-multiway cut problem on graphs of bounded branch width. In: Fellows, M., Tan, X., Zhu, B. (eds.) FAW-AAIM 2013. LNCS, vol. 7924, pp. 315–324. Springer, Heidelberg (2013)
4. Easley, D., Kleinberg, J.: Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press, New York (2010)
5. Even, S., Tarjan, R.E.: Network flow and testing graph connectivity. SIAM J. Comput. **4**(4), 507–518 (1975)
6. Iwata, S., Fleischer, L., Fujishige, S.: A combinatorial strongly polynomial algorithm for minimizing submodular functions. J. ACM **48**(4), 761–777 (2001)
7. Langberg, M., Rabani, Y., Swamy, C.: Approximation algorithms for graph homomorphism problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 176–187. Springer, Heidelberg (2006)

8. Robertson, N., Seymour, P.: Graph minors. X. Obstructions to tree-decomposition. J. Comb. Theory, Ser. B **52**(2), 153–190 (1991)
9. Zhang, P., Jiang, T., Li, A.: Improved approximation algorithms for the maximum happy vertices and edges problems. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 159–170. Springer, Heidelberg (2015)
10. Zhang, P., Li, A.: Algorithmic aspects of homophyly of networks. Theor. Comput. Sci. **593**, 117–131 (2015)