Tel Aviv University
Raymond and Beverly Sackler
Faculty of Exact Sciences
School of Computer Science

Algorithmic Problems in Graph Theory and Molecular Biology

Thesis submitted for the degree of "Doctor of Philosophy"
by
Dekel Tsur

The work on this thesis was carried out under the supervision of

Prof. Ron Shamir

Submitted to the Senate of Tel-Aviv University
2002

Abstract

We study several problems in graph theory and molecular biology. We first deal with problems on trees. We give fast polynomial algorithms to the subtree isomorphism problem. We give an exact algorithm and a polynomial time approximation scheme for the maximum subforest problem. We then study the clustering problem under a random graph model, and present fast algorithms that correctly find the clusters in the graph with high probability. These algorithms operate under a wide parameter range, and in particular, they handle non-constant number of clusters. Finally, we study Sequencing By Hybridization. We analyze a technique that allows reconstruction of long sequences under the presence of hybridization errors.

Acknowledgements

I wish to thank my advisor Prof. Ron Shamir for his tremendous help during my research.

I also thank Noga Alon, Roded Sharan, and Chaim Linhart for helpful discussions.

Contents

Chapter 1

Introduction

1.1 Summary of thesis results

We study several problems in graph theory and molecular biology.

In Chapter 2 we study the subtree isomorphism problem which is as follows: Given trees H and G, find a subtree of G which is isomorphic to H or decide that there is no such subtree. We give an $O((k^{1.5}/\log k)n)$-time algorithm for this problem, where k and n are the number of vertices in H and G, respectively. This improves over the $O(k^{1.5}n)$ algorithms of Chung and Matula [35, 95]. We also give a randomized (Las Vegas) $O(k^{1.376}n)$-time algorithm for the decision problem. These results were published in [116] and [118].

In Chapter 3 we study the following generalization of the subtree isomorphism problem, which we call the *maximum subforest problem* (MSP): Given a tree G and a set of trees $\mathcal{H}$, find a subgraph G' of G such that G' does not contain any subtree isomorphic to a tree from $\mathcal{H}$, and the number of edges in G' is maximum. We show that the maximum subforest problem is NP-hard. It is known that MSP can be solved in $2^{2^{O(k)}}n$ time, where n is the number of vertices in G, and k is the total number of vertices in $\mathcal{H}$ [127]. We give an algorithm for MSP whose time complexity is $2^{O(k^2/\log k)}n$. We also give a polynomial time approximation scheme for MSP. Moreover, we show that the corresponding edge deletion problem (namely, given $G = (V, E)$ and $\mathcal{H}$, find a set of edges $F \subseteq E$ with minimum size such that $G - F$ does not contain a subtree isomorphic to a tree from $\mathcal{H}$) is hard to approximate under various restrictions. Most of these results were published in [117].

In Chapter 4 we deal with clustering. The following probabilistic process models the generation of noisy clustering data: Clusters correspond to disjoint sets of vertices in a graph. Each two vertices from the same set are connected by an edge with probability p, and each two vertices from different sets are connected by an edge with probability $r < p$. The goal of the clustering problem is to reconstruct the clusters from the graph. We give algorithms that solve this prob-

lem with high probability. Compared to previous studies, our algorithms have lower time complexity and wider parameter range of applicability. In particular, our algorithms can handle $O(\sqrt{n}/\log n)$ clusters in an n-vertex graph, while all previously published algorithms require that the number of clusters is constant (e.g., [17, 28, 39]). A single exception is [97], which was published after we obtained and presented our results [119]. These results were published in [121]. We also study a formulation of clustering as an optimization problem and prove that it is NP-hard.

Chapter 5 deals with Sequencing by Hybridization. Sequencing by Hybridization is a method for reconstructing a DNA sequence based on its k-mer content. This content, called the *spectrum* of the sequence, can be obtained from hybridization with a universal DNA chip. However, even with a sequencing chip containing all 4^9 9-mers and assuming no hybridization errors, only about 400 bases-long sequences can be reconstructed unambiguously. Drmanac et al. [46] suggested sequencing long DNA targets by obtaining spectra of many short overlapping fragments of the target, inferring their relative positions along the target and then computing spectra of subfragments that are short enough to be uniquely recoverable. Drmanac et al. do not treat the realistic case of errors in the hybridization process. Here we study the effect of such errors. We show that the probability of ambiguous reconstruction in the presence of false negative errors is close to the probability in the errorless case. More precisely, the ratio between these probabilities is $1 + O(p/(1-p)^4 \cdot 1/d)$ where d is the average length of subfragments, and p is the probability of a false negative.

We also obtain lower and upper bounds for the probability of unambiguous reconstruction based on errorless spectrum. For realistic chip sizes, these bounds are tighter than those given by Arratia et al. [12]. Finally, we report results on simulations with real DNA sequences, showing that even in the presence of 50% false negative errors, a target of cosmid length can be recovered with less than 0.1% miscalled bases. These results were published in [120] and [122].

1.2 Preliminaries

In this section we introduce basic graph theoretic definitions and notions that will be used throughout the thesis. For more background see, e.g., [18].

A *graph* is a pair $G = (V, E)$ where V is a set of *vertices* and E is a set of unordered pairs of vertices which are called *edges*. A *directed graph* is a pair (V, E), where V is a set of vertices and E is a set of ordered pairs of vertices, that are called *directed edges*, or just edges. An edge between two vertices v and w is denoted by either (v, w) or vw (for a directed graph, (v, w) denotes an edge that exits from v and enters w). For an edge $e = (v, w)$ we say that e is *incident* on v and w. We also say in that case that v and w are *adjacent*. Unless noted otherwise, we assume that all graphs are simple, namely do not contain *parallel*

edges (i.e., two or more edges between the same pair of vertices) or self loops (a *self loop* is an edge whose endpoints are the same vertex).

A *hypergraph* is a pair (V, E) where V is a set of vertices, and E is a set of subsets of V called *hyperedges*.

A vertex u is a *neighbor* of v if $(u, v) \in E$. The *open neighborhood* of a vertex v is the set of neighbors of v, and is denoted by $N(v)$. The *closed neighborhood* of v is $N[v] = N(v) \cup \{v\}$. For a set of vertices X, let $N(X) = \cup_{x \in X} N(x)$. The *degree* of a vertex v is the number of neighbors of v, and is denoted by $d_G(v)$. We will write $d(v)$ when G is clear from the context. For a set of vertices S, $d_S(v)$ is the number of neighbors of v in S, namely $|N(v) \cap S|$. In a directed graph G, the *in-degree* of a vertex v is the number of edges that enter v, and the *out-degree* is the number of edges that exit from v.

A graph $G' = (V', E')$ is called a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, the *subgraph induced by* S, denoted G_S, is the subgraph of G whose vertex set is S and whose edge set consists of all the edges in E with both endpoints in S.

For graph $G = (V, E)$ and a set of edges $F \subseteq E$, $G - F$ is the subgraph of G that is obtained by deleting the edges in F. For a set of vertices $S \subseteq V$, $G - S$ is the induced subgraph of G that is obtained by deleting the vertices in S, i.e., G_{V-S}.

A *complete graph* is a graph that contains an edge (v, w) for every pair of vertices v, w. An induced subgraph which is a complete graph is called a *clique*.

A *path* in a graph $G = (V, E)$ is a sequence of vertices $[v_1, \ldots, v_k]$ such that $(v_i, v_{i+1}) \in E$ for every $i \leq k - 1$. The vertices v_1 and v_k are called the *endpoints* of the path. The *length* of a path is the number of edges in the path. A *cycle* is a path $[v_1, \ldots, v_k]$ in which $v_1 = v_k$. A path is called *simple* if all its vertices are distinct, and a cycle is called simple if all its vertices are distinct with exception of the endpoints. A *Hamiltonian path (cycle)* is a simple path (cycle) that passes through all the vertices of the graph. An *Eulerian path* is a path that contains all the edges in the graph, where each edge appears in the path exactly once.

The *distance* between two vertices u and v in a graph is the shortest length of a path whose endpoints are u and v. The *diameter* of a graph G is the maximum distance between two vertices in G.

A *connected component* of a graph $G = (V, E)$ is a set $S \subseteq V$ such that there is a path in G between each two vertices in S, and there is no path in G between a vertex in S and a vertex in $V - S$. A graph is called *connected* if it has only one connected component.

A *forest* is a graph without cycles. A *tree* is a connected forest. *A rooted tree* is a tree with one distinguished vertex which is called the *root*. A vertex in a rooted tree is called a *leaf* if its degree is one and it is not the root. A vertex which is not a leaf is called an *internal* vertex. For vertices v and w in a rooted tree G, we say that w is a *descendant* of v if the unique path from w to the root of G passes through v. A vertex w is a *child* of a vertex v if w is a descendent

of v and (v, w) is an edge in G. In that case we also say that v is the *parent* of w. A *postorder* of a rooted tree is an order on the vertices in which each vertex appears after its children.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijective mapping $f : V \to V'$ such that $(v, w) \in E$ iff $(f(v), f(w)) \in E$ for every $v, w \in V$. The mapping f is called an *isomorphism.*

A *bipartite graph* is a graph whose vertices can be partitioned into two disjoint sets V_1 and V_2 such that every edge connects a vertex from V_1 and a vertex from V_2. The sets V_1 and V_2 are called the *parts* of G. Bipartite graphs are also denoted by the triple (V_1, V_2, E). A *complete bipartite graph* is a bipartite graph $G = (V_1, V_2, E)$ that contains an edge (v, w) for every $v \in V_1$ and $w \in V_2$. $K_{a,b}$ denotes a complete bipartite graph with a vertices in one part, and b vertices in the other part.

A *matching* in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ such that each vertex in V is incident on at most one edge in M. If a vertex v is incident on an edge in M, we say that v is *matched* by M. A matching in G is called a *maximum matching* if it has maximum size among all matchings in G.

A *cut* (respectively, *m-cut*) in a graph $G = (V, E)$ is a set of edges F such that $G - F$ is not connected (respectively, $G - F$ contains at least m connected components). For a set of vertices S, $(S, V - S)$ denotes the cut that contains all the edges with one endpoint in S and the other in $V - S$. A *minimum cut* in a graph is a cut of minimum size.

A *source-sink labelled graph* is a triple (G, s, t) where G is a graph, and s and t are vertices of G. The *series composition* of the source-sink labelled graphs (G_1, s_1, t_1) and (G_2, s_2, t_2) is the source-sink labelled graph (G, s_1, t_2) obtained by taking the graphs G_1, G_2 and identifying t_1 with s_2. The *parallel composition* of (G_1, s_1, t_1) and (G_2, s_2, t_2) is the source-sink labelled graph (G, s_1, t_1) obtained by taking the graphs G_1, G_2 and identifying s_1 with s_2, and t_1 with t_2. A source-sink labelled graph (G, s, t) is a *series-parallel graph* if either G consists of the vertices s and t and an edge between them, or (G, s, t) is the series or parallel composition of two series-parallel graphs. A graph $G = (V, E)$ is a *series-parallel graph* if (G, s, t) is a series-parallel graph for some $s, t \in V$. Note that G may contain parallel edges.

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(T = (I, F), \{X_i : i \in I\})$, where $X_i \subseteq V$ for every $i \in I$, and T is tree such that

- $\cup_{i \in I} X_i = V$.
- For every edge $(u, v) \in E$, there is an $i \in I$ for which $u, v \in X_i$.
- For all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree-decomposition $(T, \{X_i : i \in I\})$ is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all possible tree-decompositions

of G. The family of graphs with treewidth 1 is exactly the family of forests. The family of graphs with treewidth at most 2 contains all the series-parallel graphs.

A *path-decomposition* of a graph $G = (V, E)$ is a tree-decomposition $(T, \{X_i : i \in I\})$ of G in which T is a path. The *pathwidth* of a graph G is is the minimum width over all possible path-decompositions of G.

A *network* is a source-sink labelled directed graph (G, s, t), with capacity $c_e \geq 0$ for every edge e in G. A *flow* in a network $(G = (V, E), s, t)$ is a mapping $f : E \to \mathbb{R}^+$ such that

- $f(e) \leq c_e$ for every $e \in E$.
- For each $v \in V - \{s, t\}$, $\sum_{w:(w,v)\in E} f((w, v)) = \sum_{w:(v,w)\in E} f((v, w))$.

The *value* of a flow f is $\sum_{w:(s,w)\in E} f((s, w))$.

For a set A, 2^A denotes the set of all subsets of A.

Chapter 2

The Subtree Isomorphism Problem

2.1 Introduction

A fundamental problem in graph theory is the *subgraph isomorphism problem*: Given two graphs G and H, find a subgraph of G which is isomorphic to H (if there is one). This problem is NP-hard as it includes, for example, the clique problem.

When restricting the graph G, the subgraph isomorphism problem may become easier. Polynomial-time algorithms for the subgraph isomorphism problem were given for trees [94], two-connected outerplanar graphs [81], and two-connected series-parallel graphs [82]. All these graphs have a treewidth of at most 2. The subgraph isomorphism problem is NP-hard when the graph G is an arbitrary graph with treewidth 2 [93] (In fact, is remains NP-hard even when G has treewidth of 2, H is a tree, and each graph has at most one vertex with degree greater than 3), or even when G is a graph with pathwidth 2 [61]. However, the problem is polynomial for the family of graphs with treewidth at most p for every $p \geq 2$, if we also add the restriction that G is p-connected, or that H has bounded degree [93]. A faster algorithm for the former case was given in [42].

The subgraph isomorphism problem is also polynomial on planar graphs, or more generally, on the family of graphs with no $K_{3,a}$ minor for every fixed a [50].

In this chapter we study the *subtree isomorphism problem*, i.e., the subgraph isomorphism problem when G and H are trees. Figure 2.1 gives an instance of this problem. The subgraph isomorphism and the subtree isomorphism problems have applications in pattern recognition, computational biology, and chemistry [5, 103, 126]. Throughout this chapter, n and k denote the number of vertices in G and H, respectively. When $k = n$ we get the *tree isomorphism problem*, which has a linear time algorithm due to Hopcroft and Tarjan [69]. Polynomial algorithms for subtree isomorphism were first given by Matula [94] and by Edmonds (cf. [95]).

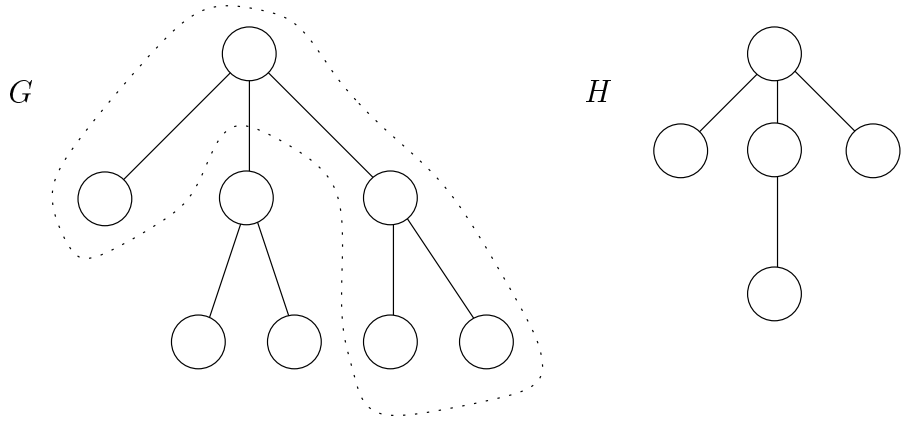

Figure 2.1: An instance of the subtree isomorphism problem. The tree G has a subtree which is isomorphic to H.

Faster algorithms were given by Matula [95] and Chung [35]. The time complexity of Chung's algorithm is $O(k^{1.5}n)$. Furthermore, the subtree isomorphism problem is in RNC, namely, it can be solved by a randomized parallel algorithm in $\log^{O(1)} n$ time [59, 74], and the problem on a restricted family of trees is in LOGSPACE and hence in NC [60]. In contrast, the subgraph isomorphism problem is NP-hard when G is a tree and H is a forest (*subforest isomorphism* [57]).

The subtree isomorphism problem on rooted trees is as follows: Given two rooted trees G and H, find a subgraph G' of G (whose root is the root of G) such that there is an isomorphism between H and G' that maps the root of H to the root of G'. This problem can be solved in $O(\min(k^{1.5}, \sqrt{kn}\log\log n/\log n)\cdot n)$ time [80]. A similar problem is the tree pattern matching problem [32,37,38,47, 67,78,88]: The input is two rooted trees G, H with an order on the children of every vertex in the trees. The goal is to find a 1-1 mapping f from the vertices of H into the vertices of G such that for every vertex v in H, the i-th child of v is mapped to the i-th child of $f(v)$. The tree pattern matching problem can be solved in $n\log^{O(1)} k$ time [37].

Several generalizations of the subtree isomorphism problem have been studied. In the *largest common subtree problem*, the input is trees $G_1, \ldots, G_l$, and the goal is to find a tree H with the maximum number of vertices, such that each G_i contains a subgraph that is isomorphic to H. The problem for two trees is polynomial (and in RNC), while it is NP-hard for three or more trees [2]. Approximation algorithms for this problem were given in [3, 75]. Another generalization is the *maximum subforest problem*: Finding a subforest of a given tree with the maximum number of edges which does not contain a subgraph isomorphic to any tree from a given set of trees. We will study this problem in Chapter 3.

2.1.1 New results

Our main result is an $O((k^{1.5}/\log k)n)$-time algorithm. Our algorithm, like most previous studies of the problem, is based on the close relationship between subtree isomorphism and maximum matching in bipartite graphs. The subtree isomorphism problem is recursively translated into a collection of smaller subtree isomorphism problems, which are solved using maximum matching algorithms. The improved complexity is achieved by a combinatorial lemma that bounds the possible number of distinct subtrees involved, and by using the notion of clique partition and its application by Feder and Motwani to finding maximum matching in bipartite graphs [53]. We show that for the matching problem resulting from the subtree isomorphism problem, we can find a clique partition in a simple way. The ideas we use here also give an improved algorithm for the maximum subforest problem (see Chapter 3).

We also give a randomized (Las Vegas) algorithm for the decision problem whose expected running time is $O(k^{\omega-1}n)$, where ω is the exponent of matrix multiplication. Using the best known bound for ω [40], the above bound is $O(k^{1.376}n)$. This algorithm follows directly from a randomized algorithm for finding the cardinality of a maximum matching given by Cheriyan [33]. The results in this chapter were published in [116] and [118].

The rest of the chapter is organized as follows: Section 2.2 describes an $O(k^{1.5}n)$ algorithm for subtree isomorphism. In Section 2.3 we prove some simple lemmas on matching that are needed in later sections. In Section 2.4 we prove the combinatorial lemma and use it together with the clique partition in order to improve the running time of the algorithm from Section 2.2. Finally, Section 2.5 describes the randomized algorithm.

We finish this section with some definitions. A *rooted tree* is a triplet $G = (V, E, r)$, where (V, E) is an unrooted tree, and r is some vertex in V which is called the *root*. We will sometimes write G^r to denote that r is the root of the rooted tree G. Also, for an unrooted tree G, we denote by G^r the rooted tree formed by choosing the vertex r to be its root. We denote by G^r_v the rooted subtree of G^r whose vertices are all descendants of v, and its root is v.

We say that two rooted trees G^r and H^s are *isomorphic* if there is an isomorphism between G and H which maps r to s. Likewise, we write $H^s \subseteq_T G^r$ if there is a rooted subtree J^r of G^r which is isomorphic to H^s (note that the subtree J^r must have the same root as G^r).

2.2 An $O(k^{1.5}n)$ algorithm

In this section we describe an $O(k^{1.5}n)$ algorithm for the subtree isomorphism problem that will be the basis for the improved algorithms in sections 2.4 and 2.5. It is based on Chung's algorithm [35] and has the same asymptotic time com-

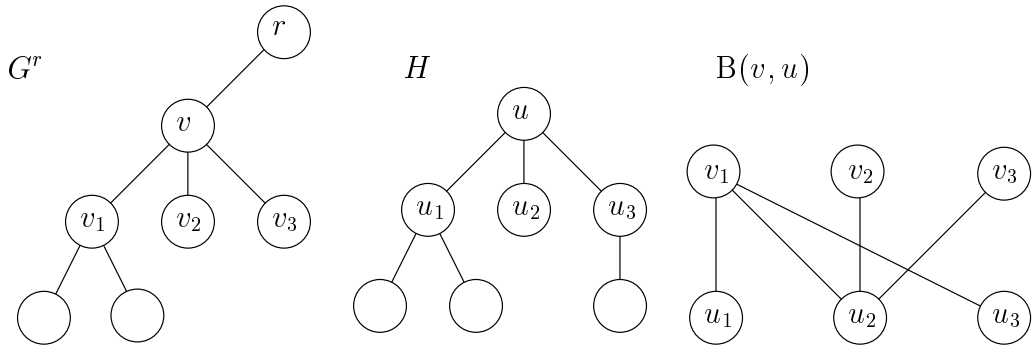

Figure 2.2: An instance of subtree isomorphism. Here, we have $H^u, H^{u_2}_u \not\subseteq_T G^r_v$ and $H^{u_1}_u, H^{u_3}_u \subseteq_T G^r_v$, so $S(v,u) = \{u_1, u_3\}$. The graph $\mathrm{B}(v,u)$ is the bipartite graph which we construct in order to compute $S(v,u)$. There is an edge $u_i v_j$ in this graph iff $u \in S(v_j, u_i)$. $H^u \not\subseteq_T G^r_v$ as $\mathrm{B}(v,u)$ does not contain a matching of size 3. $H^{u_1}_u \subseteq_T G^r_v$ as $\mathrm{B}_{u_1}(v,u) = \mathrm{B}(v,u) - u_1$ contains a matching of size 2.

plexity, but is simpler as the tree G is traversed only once, compared to twice in Chung's algorithm. We briefly describe the algorithm for completeness. We note that the improvements of sections 2.4 and 2.5 can also be applied to Chung's original algorithm. For simplicity, we describe an algorithm for the decision problem. It can be extended easily to an algorithm for the search problem.

Let $G = (V, E)$ and $H = (V_H, E_H)$ be the input trees (w.l.o.g. $|V_H| > 1$). Select a vertex r of G to be the root. We wish to know for each $v \in V$ and $u \in V_H$ whether $H^u \subseteq_T G^r_v$. In order to compute this efficiently we also need to know for each neighbor w of u if $H^w_u \subseteq_T G^r_v$ (notice that H^w_u is the graph formed by removing H^u_w from H^u). This information is relevant because $H^u \subseteq_T G^r_v$ iff for every child u' of u there is a distinct child v' of v such that $H^u_{u'} \subseteq_T G^r_{v'}$. More generally, we have the following lemma:

Lemma 2.2.1. *For every vertex v in G^r, vertex u in H, and a vertex $w \in N[u]$, we have that $H^w_u \subseteq_T G^r_v$ iff for every child u' of u in H^w_u there is a distinct child v' of v such that $H^u_{u'} \subseteq_T G^r_{v'}$.*

We store this information in sets $S(v,u)$ defined as follows: For every $v \in V$, and for every $u \in V_H$,

$$S(v,u) = \{w \in N[u] : H^w_u \subseteq_T G^r_v\}.$$

See Figure 2.2 for an example for this definition. Notice that (1) $u \in S(v,u)$ if and only if $H^u = H^u_u \subseteq_T G^r_v$, (2) $u \in S(v,u)$ implies $S(v,u) = N[u]$, and (3) $d(v) < d(u) - 1$ implies $S(v,u) = \phi$.

The algorithm computes the sets $S(v,u)$ for all v and u. We show how to compute $S(v,u)$ inductively by going over the vertices v of G^r from the leaves to the root and computing $S(v,u)$ for all $u \in V_H$ (the algorithm is also described using pseudo-code in Figure 2.3). The base of the induction is when v is a leaf of

```
1  Select a vertex r of G to be the root of G.
2  for all u ∈ H, v ∈ G do S(v, u) ← φ.
3  for all leaves v of G^r do
4      for all leaves u of H do S(v, u) ← N(u).
5  for all internal vertices v of G^r in a postorder do
6      Let v_1, ..., v_t be the children of v.
7      for all vertices u = u_0 of H with degree at most t + 1 do
8          Let u_1, ..., u_s be the neighbors of u.
9          Construct a bipartite graph B(v, u) = (X, Y, E_vu),
           where X = {u_1, ..., u_s}, Y = {v_1, ..., v_t},
           and E_vu = {u_i v_j : u ∈ S(v_j, u_i)}.
           Denote X_0 = X and X_i = X − {u_i}.
10         for all 0 ≤ i ≤ s do compute the size m_i of a maximum
           matching between X_i and Y.
11         S(v, u) ← {u_i : m_i = |X_i|, 0 ≤ i ≤ s}.
12         if u ∈ S(v, u) then answer YES and stop
13     end for
14 end for
15 Answer NO.
```

Figure 2.3: Algorithm Subtree-Isomorphism(G, H).

G^r. Then $S(v, u)$ can be computed for all u as follows: If u is a leaf of H, then $S(v, u)$ consists of one vertex which is the single neighbor of u in H. Otherwise, if u is an internal vertex, $S(v, u)$ is empty.

For the inductive step, consider an internal vertex v (from (3) we can assume that $d(v) \geq d(u)-1$). We first need to compute $S(v', u)$ for all the children v' of v, for all $u \in V_H$. Then, in order to decide for $w \in N[u]$ if $w \in S(v, u)$, we construct a bipartite graph $\mathrm{B}_w(v, u)$ with the two parts $X_w^{v,u}$ and $Y^{v,u}$, where $X_w^{v,u}$ is the set of children of u in H_u^w, $Y^{v,u}$ is the set of children of v, and $u'v'$ is an edge of $\mathrm{B}_w(v, u)$ iff $H_{u'}^u \subseteq_T G_{v'}^r$ (i.e., iff $u \in S(v', u')$). By Lemma 2.2.1, w is in $S(v, u)$ iff $\mathrm{B}_w(v, u)$ has a matching of size $|X_w^{v,u}|$. Therefore, in order to compute $S(v, u)$ we need to find maximum matchings in $d(u)+1$ bipartite graphs. However, all these graphs are similar one to another: Each graph $\mathrm{B}_w(v, u)$ (for $w \neq u$) is obtained by deleting the vertex w from the graph $\mathrm{B}_u(v, u)$. In Section 2.3 (Corollary 2.3.2) we shall show that it suffices to find a maximum matching only in $\mathrm{B}_u(v, u)$, and then we can efficiently compute the size of the maximum matching in $\mathrm{B}_w(v, u)$ for all $w \neq u$. In the following, we will use $\mathrm{B}(v, u)$ instead of $\mathrm{B}_u(v, u)$. See Figure 2.2 for an example of the relation between $S(v, u)$ and the graph $\mathrm{B}(v, u)$.

Algorithm Subtree-Isomorphism is described by the pseudo-code shown in Figure 2.3.

Theorem 2.2.2. *Algorithm Subtree-Isomorphism solves the subtree isomorphism problem in $O(k^{1.5}n)$ time and $O(kn)$ space.*

Proof. The correctness of the algorithm follows from Lemma 2.2.1. Let us consider the space complexity. Let $V = \{x_1, \ldots, x_n\}$, and $V_H = \{y_1, \ldots, y_k\}$. As each set $S(v,u)$ is a subset of $N[u]$, we can maintain $S(v,u)$ using a binary vector $A(v,u)$ of size $|N[u]|$. Also, we maintain a $k \times k$ matrix I, where $I(u,w)$ is the index of $w \in N[u]$ in the vector $A(v,u)$. In other words, $w \in S(v,u)$ iff bit number $I(u,w)$ in $A(v,u)$ is set. Thus the space complexity is $O(k^2 + \sum_{j=1}^{k} d(y_j)n) = O(kn)$ and each access to $S(v,u)$ takes constant time.

As for the algorithm's time complexity, the crux is step 10. It computes the size of several maximum matchings in some graphs. Since these graphs are very related to each other, we are able to show in Section 2.3 that computing the sizes of their maximum matchings can be done in the same time bound taken by computing a single maximum matching. We therefore perform step 10 of the algorithm using Corollary 2.3.2, and therefore the dominant part of the algorithm's time complexity is finding maximum matchings in the graphs $\mathrm{B}(x_i, y_j)$ for all i, j in step 10. Finding a maximum matching in $\mathrm{B}(x_i, y_j)$ takes $O(d(y_j)^{1.5}d(x_i))$ time using the algorithm of Hopcroft and Karp [68] (or the equivalent algorithm of Dinic [43]) and therefore the time needed to handle a vertex x_i is at most $O(\sum_{j=1}^{k} d(y_j)^{1.5}d(x_i)) = O((2k)^{1.5}d(x_i))$. Thus, the total time complexity is $O(k^{1.5}n)$. ∎

We note that the above algorithm can be changed to solve the *subtree homeomorphism* problem without changing the asymptotic complexity. The same modification applies to the algorithms in the sections below.

2.3 Matching

In this section, we give several lemmas about matchings which are needed for obtaining the more efficient algorithms for the subtree isomorphism problem. For the following lemmas, let $B = (X, Y, E)$ be a bipartite graph, where $X = \{x_1, \ldots, x_s\}, Y = \{y_1, \ldots, y_t\}$ and $s \leq t + 1$. Denote $X_0 = X$ and $X_i = X - \{x_i\}$ for $1 \leq i \leq s$. For $0 \leq i \leq s$, let m_i denote the size of a maximum matching between X_i and Y. Clearly for every $i \geq 1$, either $m_i = m_0$ or $m_i = m_0 - 1$.

An important notion in matching theory is critical vertices (see, e.g., [87]): A vertex x in a graph G is *critical* if the size of a maximum matching in $G - x$ is strictly less than the size of a maximum matching in G (i.e., x_i is critical iff $m_i = m_0 - 1$). Let M be a maximum matching of B. We define a directed graph B_M by $B_M = (X \cup Y, E_M)$ where

$$E_M = \{(x,y) : xy \in E - M, x \in X, y \in Y\} \cup \{(y,x) : xy \in M, x \in X, y \in Y\}.$$

We denote by X_M all the vertices from X which are unmatched in M.

Lemma 2.3.1. *For every maximum matching M of B and every vertex $x_i \in X$, x_i is critical (i.e. $m_i = m_0 - 1$) if and only if x_i is matched in M, and there is no directed path in B_M from a vertex in X_M to x_i.*

Proof. ($\rightarrow$) The proof is by contradiction. If x_i is unmatched in M, then M is a matching between X_i and Y and therefore $m_i = m_0$, a contradiction. Also, if there is a path P in B_M from a vertex in X_M to x_i, then $M \triangle P$ $(= M \cup P - M \cap P)$ is a maximum matching in which x_i is unmatched, and again we have a contradiction as $m_i = m_0$.

($\leftarrow$) Conversely, assume by contradiction that $m_i = m_0$. Let y be the vertex matched to x_i in M. As $|M - \{x_i y\}| = m_0 - 1 < m_i$, $M - \{x_i y\}$ is not a maximum matching between X_i and Y, and by Berge's theorem (see [87]) there is an augmenting path P whose one end is a vertex $x_j \in X_M$, and the other end is y (because if the other end is a vertex in Y that is unmatched in M then P is an augmenting path for M). But this implies a directed path in B_M from x_j to x_i, a contradiction. ■

Corollary 2.3.2. *Given a maximum matching M of B, we can compute the value of m_i for $0 \leq i \leq s$ in $O(st)$ time.*

Proof. Building the graph B_M and finding all the vertices reachable from X_M can be done in $O(st)$ time using a depth-first search [128]. We then apply Lemma 2.3.1. For each vertex x_i, if x_i is matched in M and is not reachable from X_M we set $m_i = |M| - 1$ and otherwise $m_i = |M|$. ■

Lemma 2.3.3. *The problem of finding a maximum matching in B can be reduced in $O(st)$ time to the problem of finding a maximum matching in a subgraph of B with at most s^2 vertices and edges and with maximum degree at most s.*

Proof. Let X' be the set of all vertices of X with degree less than s. Let B' be the subgraph induced from B by the vertices of X' and their neighbors. Building X' and B' takes $O(|X| + |Y| + |E|) = O(st)$ time. We claim that given a maximum matching M' of B' we can build a maximum matching M of B: First, add all the edges of M' to M. For each vertex $x \in X - X'$ find an unmatched neighbor $y \in Y$ and add xy to M (such a vertex y must exist since each such x has at least s neighbors and at most $s-1$ of them are matched). Finding an unmatched neighbor of x takes $O(s)$ time, and therefore building M takes $O(s^2)$ time. ■

2.4 An $O(\frac{k^{1.5}}{\log k}n)$ algorithm

We now improve the algorithm from Section 2.2 by a $\log k$ factor. We use the previous algorithm but we solve the maximum matching problems more efficiently using the idea of clique partition of a bipartite graph and its usage in finding maximum matching [53]. The algorithm of Feder and Motwani, originally stated

for bipartite graphs with equal size parts, can be extended to general bipartite graphs. This allows one to give an algorithm for subtree isomorphism whose time complexity is $O((k^{1.5}/\log k)n)$. Instead of describing the modifications to the algorithm of Feder and Motwani, we will give here a simpler way for obtaining an $O((k^{1.5}/\log k)n)$-time algorithm for subtree isomorphism. In contrast with [53] where the denseness of the graph is exploited, we achieve the reduction in time complexity by utilizing the special structure of the matching problems that must be solved in the subtree isomorphism algorithm.

The modified algorithm, called Improved-Subtree-Isomorphism, is the same as the algorithm Subtree-Isomorphism with the exception that, in step 10, we solve the maximum matching problems differently. Let v be some vertex in G^r whose children are $v_1, \ldots, v_t$, and let u be a vertex in H whose neighbors are $u_1, \ldots, u_s$. We now consider finding a maximum matching in $B = \mathrm{B}(v, u)$. Recall that $B = (X, Y, E)$ with $X = \{u_1, \ldots, u_s\}$ and $Y = \{v_1, \ldots, v_t\}$. We assume that $s \leq t + 1$ (because otherwise $S(v, u) = \phi$).

We first apply Lemma 2.3.3 and build a subgraph $B' = (X', Y', E')$ of B having maximum degree at most s. Then, like in [53], we partition the edges of B' into complete bipartite graphs $C_1, \ldots, C_p$. We do the partition in the following way: First, we sort the vertices of X' in lexicographic order where the key of a vertex u is $N(u)$. Afterward, we split X' into sets of equal keys $X^1, \ldots, X^p$ (i.e., all the vertices in a set X^i have the same neighbors in Y'). Now, for $1 \leq i \leq p$ we set C_i to be the subgraph induced by the vertices of X^i and all their neighbors in Y'. We now follow the method of [53] and build a network B^* whose vertices are $V^* = X' \cup Y' \cup \{c_1, \ldots, c_p, a, b\}$. The edges are $E^* = E_1 \cup E_2$, where

$$E_1 = \{au_i : u_i \in X'\} \cup \{v_i b : v_i \in Y'\}$$

$$E_2 = \{u_i c_j : j \leq p, u_i \in C_j\} \cup \{c_j v_i : j \leq p, v_i \in C_j\}$$

All edges have capacity 1. The source is a and the sink is b. We find a maximum (integral) flow f in B^* using Dinic's algorithm [43] (see also [51]), and construct from this flow a maximum matching in B'. (Since the capacity of all edges is 1, the flow f can be decomposed into edge-disjoint paths from a to b where the flow along each path is 1. Since each such path is of the form $[a, u_i, c_k, v_j, b]$, we can define a matching in B' by taking the edge $u_i v_j$ for each such path. The maximality of this matching follows from the maximality of the flow.)

We will now analyze the time complexity of the algorithm described above. We denote by $D(u)$ the number of distinct trees in the forest $H^u_{u_1}, \ldots, H^u_{u_s}$.

Lemma 2.4.1. *Algorithm Improved-Subtree-Isomorphism finds a maximum matching in* $\mathrm{B}(v, u)$ *in* $O(st + ts^{0.5}D(u))$ *time.*

Proof. Note that if for some i, j the rooted trees $H^u_{u_i}$ and $H^u_{u_j}$ are isomorphic, then in $B = \mathrm{B}(v, u)$ the vertices u_i, u_j have exactly the same neighbors, and this

remains true in B' (assuming that u_i and u_j were not deleted in B'). Therefore $p \leq D(u)$.

The time for constructing B, B', and B^* is $O(st)$ (we sort the vertices of X' using radix-sort). We now bound the time for finding a maximum flow in B^*: The size of E_1 is $|X'| + |Y'|$, where $|X'| \leq s$ and $|Y'| \leq sp$ (since each vertex in Y' has at least one edge arriving from some vertex c_j, and no more than s edges depart from each vertex c_j to the vertices in Y'). The size of E_2 is at most $2sp$ as the number of edges in E_2 incident on some vertex c_j is at most $|X'| + d_{B'}(u_i) \leq 2s$ where $u_i \in C_j$. Hence, the number of edges in B^* is $O(sp)$. The number of vertices in B^* is at most $s + t + p + 2 = O(t)$ (as $s \leq t + 1$). Now, Dinic's algorithm performs $O(\sqrt{|V^*|})$ stages (see [53]), and each stage takes $O(|E^*|)$ time. Hence, the total time is $O(t^{0.5}sp) = O(ts^{0.5}p) = O(ts^{0.5}D(u))$. ■

We denote again $V = \{x_1, , \ldots, x_n\}$ and $V_H = \{y_1, , \ldots, y_k\}$. By Lemma 2.4.1, the time complexity of algorithm Improved-Subtree-Isomorphism is

$$O(\sum_{i=1}^{n}\sum_{j=1}^{k}(d(y_j)d(x_i) + d(x_i)d(y_j)^{0.5}D(y_j))) = O(kn + n\sum_{j=1}^{k}d(y_j)^{0.5}D(y_j)).$$

We will bound the summation $\sum_{j=1}^{k} d(y_j)^{0.5}D(y_j)$ by $O(k^{1.5}/\log k)$.

We first need a simple combinatorial lemma. Let $g(n)$ denote the number of distinct (i.e., non-isomorphic) rooted trees with n vertices. We shall use the following result:

Lemma 2.4.2. (see, e.g., [100, p. 1197]) $g(n) = 2^{\Theta(n)}$.

Let $f(n)$ denote the maximum number of distinct rooted trees in a forest with n vertices.

Lemma 2.4.3. $f(n) = \Theta(n/\log n)$.

Proof. To show the lower bound, we use the fact that $g(n) \geq 2^n$ for large n. Hence, the number of distinct rooted trees with $l = \lfloor \log \frac{n}{\log n} \rfloor$ vertices is $g(l) \geq 2^{\log \frac{n}{\log n} - 1} = \frac{n}{2 \log n}$. Therefore we can build a forest by taking $\lfloor \frac{n}{2 \log n} \rfloor$ distinct trees with l vertices each, and the total number of vertices in this forest is $\lfloor \frac{n}{2 \log n} \rfloor l < n$. Thus $f(n) = \Omega(\frac{n}{\log n})$. We will now show the upper bound.

If we have a forest of rooted trees and r_i is the number of trees with i vertices, then the number of distinct trees in this forest is at most $\sum_{i=1}^{n} \min(r_i, g(i))$. Hence,

$$f(n) \leq \max\left\{\sum_{i=1}^{n}\min(r_i, g(i)) : r_1, \ldots, r_n \in \mathbb{N}, \sum_{i=1}^{n} i r_i \leq n\right\}.$$

By Lemma 2.4.2,

$$f(n) \leq \max\left\{\sum_{i=1}^{n} \min(r_i, c^i) : r_1, \ldots, r_n \in \mathbb{N}, \sum_{i=1}^{n} i r_i \leq n\right\}$$

for some integer constant c. Let x be the minimum integer for which $\sum_{i=1}^{x} ic^i \geq n$. Let $r_1, \ldots, r_n$ be the integers that maximize $\sum_{i=1}^{n} \min(r_i, c^i)$ under the constraint $\sum_{i=1}^{n} ir_i \leq n$. We can assume that $r_i \leq c^i$ for all i, because if $r_j > c^j$ for some j, we can set $r_j = c^j$ and the value of $\sum_{i=1}^{n} \min(r_i, c^i)$ does not change. Now, suppose that $r_j > 0$ for some $j > x$. This implies that there is a $k \leq x$ for which $r_k < c^k$ (because otherwise $\sum_{i=1}^{n} ir_i \geq \sum_{i=1}^{x} ir_i + jr_j = \sum_{i=1}^{x} ic^i + jr_j > n$, a contradiction). If we decrease r_j by one, and increase r_k by one, then the value of $\sum_{i=1}^{n} \min(r_i, c^i)$ does not change, and the constraint $\sum_{i=1}^{n} ir_i \leq n$ still holds (as $k < j$). We can repeat this process until $r_j = 0$ for all $j > x$ and therefore

$$f(n) \leq \sum_{i=1}^{n} \min(r_i, c^i) \leq \sum_{i=1}^{x} c^i = O(c^x).$$

The lemma follows from the fact that $x = \log_c n - \log_c \log_c n - O(1)$. ∎

We now continue with the analysis of algorithm Improved-Subtree-Isomorphism. Let ϵ be some constant $0 < \epsilon < 1/3$. We call a vertex of H *heavy* if its degree is at least $2k^{1-\epsilon}$, and otherwise it is called *light*. Clearly, the number of heavy vertices is at most k^ϵ. If u is a heavy vertex and v is a neighbor of u such that H_v^u does not contain a heavy vertex, then we call every vertex in H_v^u a *private vertex* of u. We denote by l_j the number of the private vertices of a heavy vertex y_j.

Lemma 2.4.4. *For every heavy vertex y_j, $d(y_j) \leq k^\epsilon + l_j$ and $D(y_j) \leq k^\epsilon + f(l_j)$.*

Proof. Let $u_1, \ldots, u_p$ denote the neighbors of y_j which are private vertices of y_j and let $v_1, \ldots, v_q$ denote the rest of the neighbors of y_j. Clearly, p is at most the total number of private vertices of y_j which is l_j. Furthermore, for each vertex v_i we can chose a heavy vertex w_i in $H_{v_i}^{y_j}$. As the vertices we chose are distinct, we have that q is less than the number of heavy vertices. Hence, $d(y_j) = q + p \leq k^\epsilon + l_j$.

The trees $H_{u_1}^{y_j}, \ldots, H_{u_p}^{y_j}$ constitute all the private vertices of y_j, and therefore they have a total of l_j vertices and there are at most $f(l_j)$ distinct trees among them. Hence, $D(y_j) \leq q + f(l_j) \leq k^\epsilon + f(l_j)$. ∎

Lemma 2.4.5. $\sum_{j=1}^{k} d(y_j)^{0.5} D(y_j) = O(k^{1.5}/\log k)$.

Proof. We split $\sum_{j=1}^{k} d(y_j)^{0.5} D(y_j)$ into two sums. Summing over the light vertices of H we have

$$\sum_{j:y_j \text{ is light}} d(y_j)^{0.5} D(y_j) \leq \sum_{j:y_j \text{ is light}} d(y_j)^{1.5} \leq (2k^{1-\epsilon})^{0.5} \sum_{j:y_j \text{ is light}} d(y_j)$$
$$\leq (2k^{1-\epsilon})^{0.5} 2k = 2^{1.5} k^{1.5-\epsilon/2},$$

where the last inequality follows from the fact that $\sum_{j=1}^{k} d(y_j) = 2k - 2$.

Summing over the heavy vertices and using Lemma 2.4.4 we have

$$\sum_{j:y_j \text{ is heavy}} d(y_j)^{0.5} D(y_j) \leq \sum_{j:y_j \text{ is heavy}} (k^{\epsilon/2} + l_j^{0.5})(k^{\epsilon} + f(l_j)).$$

Since $f(l_j) \leq l_j \leq k$ and since the number of heavy vertices is at most k^{ϵ}, we have

$$\leq \sum_{j:y_j \text{ is heavy}} (k^{3\epsilon/2} + k^{0.5+\epsilon} + k^{1+\epsilon/2} + l_j^{0.5} f(l_j))$$
$$\leq 3k^{1+3\epsilon/2} + \sum_{j:y_j \text{ is heavy}} l_j^{0.5} f(l_j)$$

and by Lemma 2.4.3, the fact that $\sum_{j:y_j \text{ is heavy}} l_j \leq k$ (as each vertex can be a private vertex of at most one heavy vertex), and the fact that the function $h(x) = x^{1.5}/\log x$ is convex,

$$\leq 3k^{1+3\epsilon/2} + \sum_{j:y_j \text{ is heavy}} c\frac{l_j^{1.5}}{\log l_j} \leq 3k^{1+3\epsilon/2} + c\frac{k^{1.5}}{\log k}.$$ ■

We therefore proved the following theorem:

Theorem 2.4.6. *Algorithm Improved-Subtree-Isomorphism solves the subtree isomorphism problem in $O((k^{1.5}/\log k)n)$ time.*

2.5 An $O(k^{1.376}n)$ algorithm

In this section we give a randomized algorithm for the decision problem of subtree isomorphism. The algorithm is more efficient asymptotically than the deterministic algorithm of the previous section.

Again, the algorithm Randomized-Subtree-Isomorphism is based on the algorithm Subtree-Isomorphism but solving the maximum matching problems is done differently. Consider some vertex v in G^r whose children are $v_1, , \ldots, v_t$, and some vertex u in H whose neighbors are $u_1, \ldots, u_s$. We use the algorithm of Cheriyan [33] to find the size of a maximum matching in $\mathrm{B}(v, u)$ and to find all the critical vertices in $\mathrm{B}(v, u)$.

Cheriyan's algorithm for finding critical vertices (in a general graph) is as follows: Given an input graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, choose a large prime number q and build an $n \times n$ matrix C in the following way: For each edge $ij \in E$, choose a random number w_{ij} uniformly from $\{1, 2, \ldots, q-1\}$, and set $c_{ij} = w_{ij}$ and $c_{ji} = -w_{ij}$. Set all the other elements of C to zero. Finally, find a basis $a_1, \ldots, a_r$ for the null space of C over the field Z_q. Then, with high probability, the size of a maximum matching in G is equal to $(n-r)/2$ and for all $i \in V$, vertex i is critical iff all the i-th coordinates of $a_1, \ldots, a_r$ are zeros.

As the graph $\mathrm{B}(v, u)$ is bipartite, when applying Cheriyan's algorithm on $\mathrm{B}(v, u)$, the matrix C is of the form

$$C = \begin{pmatrix} 0 & C' \\ C'' & 0 \end{pmatrix},$$

where C' is a $t \times s$ block and C'' is an $s \times t$ block. Therefore, we can find a basis for the null space of C by finding bases for the null spaces of C' and C''. This can be done in $O(s^{\omega-1}t)$ time [70], where ω denotes the exponent of matrix multiplication. Thus, the running time of algorithm Randomized-Subtree-Isomorphism is $O(\sum_{i=1}^{n} \sum_{j=1}^{k} d(y_j)^{\omega-1} d(x_i)) = O(k^{\omega-1}n)$.

Theorem 2.5.1. *Algorithm Randomized-Subtree-Isomorphism solves the decision problem in $O(k^{\omega-1}n)$ expected time.*

Since $\omega < 2.376$ [40], we have

Corollary 2.5.2. *Algorithm Randomized-Subtree-Isomorphism solves the decision problem in $O(k^{1.376}n)$ expected time.*

Chapter 3

The Maximum Subforest Problem

3.1 Introduction

A very common problem in algorithmic graph theory and combinatorial optimization is to find in a given graph an optimal subgraph. Examples include the maximum matching, the maximum clique and the traveling salesman problems. For a graph property P, the *maximum subgraph problem with respect to* P is to find, in a given graph $G = (V, E)$, a largest possible subset of edges $E' \subseteq E$ such that (V, E') satisfies P. Such problems are also called *edge deletion problems*, as they can be formulated as finding a minimum subset O of the edges such that $G - O$ satisfies P.

A graph property P is *hereditary* if for every graph satisfying P, all its vertex-induced subgraphs also satisfy P. Any hereditary graph property P can be characterized by the *obstruction set* $\mathcal{H}_P$ of all minimal graphs that do not satisfy P: A graph satisfies P if and only if it does not contain any graph from $\mathcal{H}_P$ as an induced subgraph.

Many maximum subgraph problems are NP-hard. However, when restricting the input graph, some problems become polynomial. In particular, it has been shown that many problem are polynomial on trees (e.g., [36, 79]) or on series-parallel graphs (e.g., [76]). Takamizawa et al. [127] gave a general result that states that for a hereditary property P with a finite obstruction set, the maximum subgraph problem can be solved in linear time on series-parallel graphs. While there are many graph properties that satisfy this condition, there are many other properties which are not hereditary (e.g., having a Hamiltonian cycle), or are hereditary but have an infinite obstruction set (e.g., the property of being a bipartite graph, whose obstruction set consists of all odd cycles).

Another limitation of the result of Takamizawa et al. is that the family of series-parallel graphs is rather small. Therefore, the question arises whether there

are graph families that generalize both trees and series-parallel graphs, and still optimization problem are easy on them. Such a generalization is the family of graph with bounded treewidth. For examples of algorithms for specific problems on graphs with bounded treewidth see [6, 7, 10]. The result given in [22, 91, 113–115] can be viewed as a generalization of the result of Takamizawa et al. to the family of bounded treewidth graphs (in some of these papers, the input graph is also required to have bounded degree). Moreover, the latter results apply to larger classes of properties.

Here is one concrete example for this type of results. Scheffler [113] studied properties P for which there is an additive function f, constants m and c, and a relation P' such that

$$P(G) = \exists G_1, \ldots, G_m \forall v \, P'(G|_{N_c(v)}, G_1|_{N_c(v)}, \ldots, G_m|_{N_c(v)}, v) \wedge f(G_1) \geq B,$$

where $N_c(v)$ is the set of all vertices that are at distance at most c from v (including v). Scheffler showed that for such properties, deciding if a graph G has property P can be done in linear time on graphs with bounded treewidth and bounded degree. In particular, by defining $f(G)$ to be the number of edges in G, this result shows that for many graph properties, the corresponding maximum subgraph problem is polynomial. Among these properties are all hereditary properties with a fixed obstruction set, and bipartiteness.

Courcelle [41] studied the *property identification problem*, namely deciding whether a graph G has property P. Courcelle showed that for a property P that is defined by a monadic second order logic formula, the property identification problem can be solved in linear time when the input graph has bounded treewidth. A *monadic second order formula* is a logic formula that is built from logic connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, vertex variables, edge variables, vertex set variables, edge set variables, the existential $(\exists)$ and universal $(\forall)$ quantifiers, the membership symbol $(\in)$, and the equality symbol. For example, bipartiteness can be expressed by the formula

$$\exists W \forall v \forall w \, (v, w) \in E \rightarrow ((v \in W \wedge \neg(w \in W)) \vee (w \in W \wedge \neg(v \in W))).$$

This result has been extended to show that for a property that is defined by a monadic second order logic formula, the maximum subgraph problem can be solved in linear time on graphs with bounded treewidth [9, 29].

The above algorithms are based on finding a tree decomposition of the input graph, and then applying a dynamic programming algorithm on that tree. The bottleneck in the time complexity was the tree decomposition, since at the time when the above papers were written, the best algorithm for finding a tree decomposition were super-linear. Later, Bodlaender gave a linear time decomposition algorithm [25]. A different kind of algorithm was given by Arnborg et al. [8] for the property identification problem: The algorithm is based on graph reductions, and thus, it does not require a tree decomposition.

For discussing parallel complexity, recall that an optimal-speedup algorithm is a parallel algorithm that has the same overall number of operations as the fastest sequential algorithm. The fastest optimal-speedup parallel algorithm for finding a tree decomposition (i.e., one that uses $O(n)$ operations on an n-vertex graph) takes $O(\log^2 n)$ time on an EREW machine [27]. The algorithm of [9] can be used to solve the maximum subgraph problem with the same parallel time complexity. Bodlaender and Hagerup [27] gave an optimal-speedup algorithm for the property identification problem on graphs with bounded treewidth, requiring $O(\log n)$ time on a CRCW machine. The algorithm is based on the reduction algorithm of Arnborg et al. [8]. This algorithm was extended to solve the maximum subgraph problem on graphs with bounded treewidth with the same time complexity [24, 26]. Bodlaender gave a dynamic algorithm for several optimization problems on graphs with treewidth 2 such that each update of the graph takes $O(\log n)$ time, and each query takes $O(1)$ or $O(\log n)$ time [23].

A major problem with the above algorithms is that the constants hidden in the time complexity are extremely large. In order to explicitly consider these constants, we will look at the time complexity as a function of the size of the input graph and the property P, or, in other words, we consider P to be part of the input. We therefore need to define how P is represented in the input. One possible way is to assume that P is represented by a monadic second order logic formula. An alternative approach is to assume that P is a hereditary property, and is represented by its obstruction set (we assume that the obstruction set is finite). Thus, the input has two parts: a query graph and an obstruction set which is a collection of forbidden graphs. In this case, the time complexity of the algorithm is measured as a function of the number of vertices in the query graph plus the total number of vertices in the forbidden graphs. We will concentrate on the case when all the input graphs are trees: Given a tree G and an obstruction set of trees $\mathcal{H}$, the goal is to find a subforest of G that does not contain a subtree isomorphic to any tree from $\mathcal{H}$ and has maximum number of edges. We call this problem the *maximum subforest problem* (MSP). We call the corresponding edge deletion problem the *tree deletion problem* (TDP). The maximum subforest problem is a generalization of the subtree isomorphism problem which was studied in Chapter 2.

Using the approach of Takamizawa et al. [127], the maximum subforest problem can be solved in $2^{2^{O(k)}}n$ time, where n is the number of vertices in G, and k is the total number of vertices in the trees in $\mathcal{H}$.

3.1.1 New results

We show that MSP is NP-hard. We also give a $2^{O(k^2/\log k)}n$-time algorithm for MSP, improving the result implied by the work of Takamizawa et al.

Next, we study the approximation problem. While MSP and TDP are equivalent if an exact solution is required, approximating the former is easier: We show

that TDP is hard to approximate within $c \log k$ factor for some constant c, and in contrast, we give a polynomial time approximation scheme for MSP. The results above (except the hardness results) were published in [117].

We note that some restrictions of MSP are tractable. In [129], we show, for example, that MSP is polynomial if each tree in $\mathcal{H}$ has diameter of at most 5. This result is best possible as it is shown in this thesis that MSP is NP-hard when $\mathcal{H}$ contains trees with diameter 6. In fact, we give stronger results that imply hardness of approximation for TDP when $\mathcal{H}$ contains trees with diameter 6.

The rest of this chapter is organized as follows: Basic definitions are given in Section 3.2. In Section 3.3 we give a general framework for exact algorithms for solving MSP. Section 3.4 describes a combinatorial lemma. In Section 3.5 we analyze the complexity of the MSP algorithm using the combinatorial lemma. In Section 3.6 we use the lemma and the exact algorithm to give a polynomial time approximation scheme for MSP. Finally, we show the hardness of several restrictions of MSP and the hardness of approximation of TDP in Section 3.7.

3.2 Preliminaries

We use the definitions from Section 2.1.1 under the context of rooted forests. For example, we write $H^s \subseteq_T G^r$ if there is a rooted subforest J^r of G^r which is isomorphic to H^s (recall that the isomorphism must map r to s). Furthermore, for a forest (rooted or unrooted) G and an unrooted tree H, we write $H \subseteq G$ if H is isomorphic to a subgraph of G. For a forest G and a set of trees $\mathcal{H}$ we write $\mathcal{H} \subseteq G$ if $H \subseteq G$ for some $H \in \mathcal{H}$. If $\mathcal{H} \not\subseteq G$ we say that G is *$\mathcal{H}$-free*.

For a graph G, $E(G)$ denotes the set of edges of G, and $e(G) = |E(G)|$. We denote the family of all rooted forests by $\mathcal{R}$.

An *l-ary rooted forest operator* is a function f which acts on l rooted forests and yields a rooted forest. Given $G_1, \ldots, G_l$, the forest $f(G_1, \ldots, G_l)$ is built by taking the forests $G_1, \ldots, G_l$, and then performing some of the following operations:

1. Merging the roots of some of the input forests.
2. Adding new vertices.
3. Adding new edges, where each endpoint of a new edge is either a root of some input forests or a new vertex.

Finally, the root of $f(G_1, \ldots, G_l)$ is either a root of some input forests or a new vertex. For an operator f, let $a(f)$ denote the number of forests on which f operates. For a set of operators Φ, let $a(\Phi) = \max_{f \in \Phi} a(f)$. An operator f is a *suboperator* of an operator f' if $a(f) = a(f')$ and for any $G_1, \ldots, G_{a(f)}$, $f(G_1, \ldots, G_{a(f)})$ is a subgraph of $f'(G_1, \ldots, G_{a(f)})$. For an operator f, let sub(f)

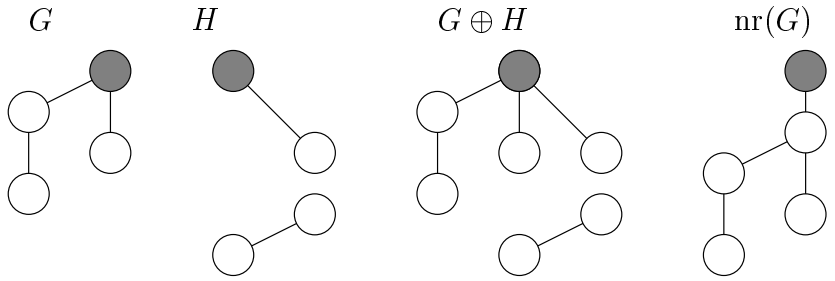

Figure 3.1: The $\oplus$ and nr operators.

denote the set of all suboperators of f. A set of operators Φ is called *closed* if $\text{sub}(f) \subseteq \Phi$ for any $f \in \Phi$. A set of operators Φ is called *complete* if every rooted forest can be built from copies of the single-node rooted tree by a series of applications of the operators in Φ.

We define some rooted forests operators: Given two rooted forests G and H, $G \oplus H$ is the rooted forest formed by taking G and H and identifying their roots. The nr (new root) operator takes a rooted forest G, adds a vertex s and an edge between s and the root of G, and makes s the new root. The nir (new isolated root) operator takes a rooted forest, adds to it a disconnected vertex and makes it the new root. Note that $\text{sub}(\text{nr}) = \{\text{nr}, \text{nir}\}$. See Figure 3.1 for examples. The set $\{\oplus, \text{nr}, \text{nir}\}$ is closed and complete.

The tree $K_{1,l}$ is composed by taking l vertices, and a distinguished vertex called the *center* and connecting the center to all other vertices. We denote by $\hat{P}_l$ the rooted tree formed by taking a path with l vertices and choosing one of the two path endpoints as the root.

Let G be a tree and P be a graph property. We define the predicate $P(G)$ to have value 1 if G has property P, and 0 otherwise.

3.3 A framework for solving MSP

In this section we describe a general method for solving maximum subforest problems based on decomposition. We will use this method in Section 3.5 to give an exact algorithm for MSP, and in Section 3.6 to give an approximation scheme for MSP. The general idea behind our framework is similar to the one used by Bern el al. [19] and others (e.g., [9, 29]), although some aspects are different, as will be explained later.

Before describing the framework, we give some definitions. Let P be some graph property. The *maximum subforest problem with respect to* P is to find, in a given tree $G = (V, E)$, a subforest of G with property P and with the maximum number of edges. In the following we will describe an algorithm for solving this problem under certain assumptions on P.

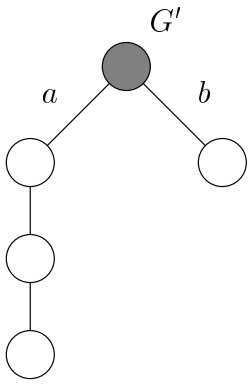

Figure 3.2: Example for the definition of complete collection. Let P be the property of not containing a path of length 4. The collection $\mathcal{C} = \{G' - \{a\}, G' - \{a, b\}\}$ is a complete collection of G' w.r.t. $\leq^3$.

Let Φ be a closed and complete set of rooted forest operators. A *composition tree (w.r.t. Φ)* of a rooted forest G is a rooted tree H, where each internal node v of H is labeled by an operator $f \in \Phi$ such that $a(f)$ is equal to the number of children of v. Each node in the tree is associated with a rooted forest: A leaf is associated with the forest $\hat{P}_1$ and an internal node is associated with the rooted forest formed by applying the node's operator on the forests associated with the children of the node. The forest associated with the root of the composition tree is isomorphic to G.

We now describe the basic idea of our algorithm. Let G be the input to the maximum subforest problem, and let G' be a forest that corresponds to some node in the decomposition tree of G. We want to create a collection of candidate subforests of G', such that the optimal solution for G will contain one of these candidates as an induced subgraph. In other words, we want to find all "pieces" within G' that may take place in an optimal solution. To do this, we need a way to choose the collection. As an example, consider the tree G' in Figure 3.2, and the property P of not containing a path of length 4. Out of the two subforests $G' - \{a\}$ and $G' - \{b\}$, it is better to take $G' - \{a\}$ into the collection: If there is an optimal solution G^* to MSP such that the subforest of G^* that is induced by the vertices of G' is $G' - \{b\}$, then $G^* \cup \{b\} - \{a\}$ is also an optimal solution. Hence, in this case, only $G' - \{a\}$ and $G' - \{a, b\}$ will be the candidates. The key to the efficiency of the approach is eliminating as many candidate subforests ("pieces") as possible.

We now give a formal description of this idea: We will define partial orders which are used to compare candidates for the collection. Let $\leq$ be some partial order on rooted forests. We say that $\leq$ is *preserved by* Φ if for any $f \in \Phi$, and any $G_1, \ldots, G_{a(f)}, G'_1, \ldots G'_{a(f)}$ such that $G_i \leq G'_i$ for $i = 1, \ldots, a(f)$, there is an operator $f' \in \mathrm{sub}(f)$ such that $f(G_1, \ldots, G_{a(f)}) \leq f'(G'_1, \ldots, G'_{a(f)})$. We say that

$\leq$ *preserves* P if $G \leq G'$ implies $P(G) \leq P(G')$. We say that $\leq$ *preserves* P *with size* if $\leq$ preserves P, and additionally, $G \leq G'$ and $P(G) = 1$ implies that $e(G) \leq e(G')$. A (P, Φ)-*order* is a partial order that preserves P with size, and is preserved by Φ.

For example, let $\Phi = \{\oplus, \mathrm{nr}, \mathrm{nir}\}$ and let P be the property of not containing a path of length 4. For a rooted forest G, we denote by $h(G)$ the height of the forest G, namely the length of a longest path that starts in the root of G. We define a partial order $\leq^1$ by $G \leq^1 G'$ iff $h(G) \geq h(G')$. We have that $\leq^1$ is preserved by Φ: If $G_1 \leq^1 G_1'$ and $G_2 \leq^1 G_2'$ then

$$h(G_1 \oplus G_2) = \max(h(G_1), h(G_2)) \geq \max(h(G_1'), h(G_2')) = h(G_1' \oplus G_2'),$$

and therefore $G_1 \oplus G_2 \leq^1 G_1' \oplus G_2'$. Similarly, $G \leq^1 G'$ implies $\mathrm{nr}(G) \leq^1 \mathrm{nr}(G')$ (as $h(\mathrm{nr}(G)) = h(G) + 1 \geq h(G') + 1 = h(\mathrm{nr}(G'))$) and $\mathrm{nir}(G) \leq^1 \mathrm{nir}(G')$. The partial order $\leq^1$ does not preserve P, since $\hat{P}_2 \leq^1 \mathrm{nir}(\hat{P}_5)$, but $P(\hat{P}_2) = 1 > 0 = P(\mathrm{nir}(\hat{P}_5))$. The partial order $\leq^2$ defined by $G \leq^2 G'$ iff $h(G) \geq h(G')$ and $P(G) \leq P(G')$ preserves P, and is preserved by Φ. The partial order $\leq^3$ defined by $G \leq^3 G'$ iff $P(G) = 0$, or ($G \leq^2 G'$ and $e(G) \leq e(G')$) is a (P, Φ)-order.

For a collection $\mathcal{C}$ of rooted forests and a partial order $\leq$, a *complete collection of* $\mathcal{C}$ *(w.r.t.* $\leq$*)* is a collection $\mathcal{C}' \subseteq \mathcal{C}$ such that (1) there are no $G, G' \in \mathcal{C}'$ such that $G \leq G'$, and (2) for any $G \in \mathcal{C} - \mathcal{C}'$ there is some $G' \in \mathcal{C}'$ such that $G \leq G'$. A collection $\mathcal{C}$ of rooted subforests of a rooted forest G is called a complete collection of G (w.r.t. $\leq$) if $\mathcal{C}$ is a complete collection (w.r.t. $\leq$) of the set of all subforests of G. See Figure 3.2 for an example.

If $\leq$ is a (P, Φ)-order and we have a procedure to compute $\leq$, then the maximum subforest problem with respect to P can be solved by the algorithm MaxSubforest that is given in Figure 3.3.

Lemma 3.3.1. *Algorithm MaxSubforest solves the maximum subforest problem with respect to property* P*.*

Proof. To prove the correctness of this algorithm, we will show that for any tree G' corresponding to a node v in the composition tree, $\mathcal{C}(G')$ is a complete collection of G'. The proof is by induction on the structure of G'. The base of the induction is trivial. Suppose that $G' = f(G_1, \ldots, G_l)$, where $G_1, \ldots, G_l$ are the forests that correspond to the children of v. By construction there are no $H, H' \in \mathcal{C}(G')$ such that $H \leq H'$. It remains to show that for any subforest H of G', there is a subforest $H' \in \mathcal{C}(G)$ such that $H \leq H'$. Let H be some subforest of G'. We have that $H = f'(H_1, \ldots, H_l)$, where $f' \in \mathrm{sub}(f)$ and H_i is a subforest of G_i for $i = 1, \ldots, l$. By the induction hypothesis, there are $H_1' \in \mathcal{C}(G_1), \ldots, H_l' \in \mathcal{C}(G_l)$ such that $H_i \leq H_i'$ for $i = 1, \ldots, l$. Since $\leq$ is preserved by Φ, it follows that $H \leq f'(H_1', \ldots, H_l')$. Furthermore, it is easy to verify that $\mathcal{C}(G')$ is a complete collection of $\mathcal{C}_0(G')$. Thus, as $f'(H_1', \ldots, H_l') \in \mathcal{C}_0(G')$, there is a forest $H' \in \mathcal{C}(G)$ such that $f'(H_1', \ldots, H_l') \leq H'$. Therefore, $\mathcal{C}(G')$ is a complete collection of G'.

1 Arbitrarily choose a vertex r in G.
2 Compute a composition tree for G^r.
3 Scan the nodes of the composition tree in postorder.
for every node v **do**
4 Let G' be the forest that corresponds to v.
5 **if** v is a leaf **then** let $\mathcal{C}(G') = \{G'\}$.
else
6 Let f be the operator associated with v.
Let $G_1, \ldots, G_l$ be the trees that correspond to the children of v.
(i.e., $G' = f(G_1, \ldots, G_l)$)
7 Let $\mathcal{C}_0(G') = \{f'(H_1, \ldots, H_l) : f' \in \mathrm{sub}(f), H_1 \in \mathcal{C}(G_1), \ldots, H_l \in \mathcal{C}(G_l)\}$.
Let $\mathcal{C}(G') = \phi$.
8 **for** every $H \in \mathcal{C}_0(G')$ **do**
9 **if** there is least one forest $H' \in \mathcal{C}(G')$ such that $H' \leq H$ but $H \not\leq H'$
then remove all such forests from $\mathcal{C}(G')$ and add H to $\mathcal{C}(G')$.
else if there is no $H' \in \mathcal{C}(G')$ such that $H \leq H'$ **then** add H to $\mathcal{C}(G')$.
10 Check for each forest in $\mathcal{C}(G^r)$ if it has property P, and output a forest from $\mathcal{C}(G^r)$ that has property P and has maximum number of edges.

Figure 3.3: Algorithm MaxSubforest(G).

Let H^* be an optimal solution for the maximum subforest problem. From the fact that $\mathcal{C}(G^r)$ is a complete collection, it follows that there is a subforest $H \in \mathcal{C}(G^r)$ such that $H^* \leq H$, and from the fact that $\leq$ preserves P with size, it follows that H is an optimal solution. The algorithm returns a forest H' from $\mathcal{C}(G^r)$ that has property P and has maximum number of edges, and in particular, $e(H) \leq e(H')$. Therefore, H' is an optimal solution. ■

We now compute the time complexity of the algorithm. Let $|\leq|$ denote the size of the largest complete collection of $\mathcal{R}$ w.r.t. $\leq$ (we assume that $|\leq|$ is finite). Suppose that checking whether a forest G with at most n vertices has property P takes at most $T_1(n)$ time, and checking whether $G \leq G'$ for two forests with at most n vertices each takes at most $T_2(n)$ time. Suppose that v is some node in the decomposition tree, G' is the forest that corresponds to v, and $G_1, \ldots, G_l$ are the forests that correspond to the children of v. The time complexity of building the collection $\mathcal{C}(G')$ is $O(|\mathcal{C}_0(G')| \cdot |\leq| \cdot T_2(n))$. We have that

$$|\mathcal{C}_0(G')| \leq \max_{f \in \Phi} |\mathrm{sub}(f)| \cdot \prod_{i=1}^{l} |\mathcal{C}(G_i)| \leq \max_{f \in \Phi} |\mathrm{sub}(f)| \cdot |\leq|^{a(\Phi)}.$$

Therefore, we have:

Theorem 3.3.2. *The time complexity of algorithm MaxSubforest is*

$$O(\max_{f \in \Phi} |\mathrm{sub}(f)| \cdot |\leq|^{a(\Phi)+1} \cdot T_2(n) \cdot n + |\leq| \cdot T_1(n)).$$

Naturally, given P and Φ, our goal will be to find a (P, Φ)-order $\leq$ such that $|\leq|$ is as small as possible.

We now give an example for the framework. For the rest of the section, let $\Phi_1 = \{\oplus, \mathrm{nr}, \mathrm{nir}\}$, and assume that P is some hereditary property. We define two partial orders on rooted forests: For two rooted forests G and G',

$$G \leq_P G' \iff \forall \text{rooted forest } J,\ P(G \oplus J) \leq P(G' \oplus J),$$

and

$$G \leq'_P G' \iff P(G) = 0 \text{ or } (G \leq_P G' \text{ and } e(G) \leq e(G')).$$

It is clear that $\leq_P$ is partial order. To show that $\leq'_P$ is a partial order, we need to show that $\leq'_P$ is transitive. Suppose that $G \leq'_P G' \leq'_P G''$. If $P(G) = 0$ then we are done. Otherwise, we have $1 = P(G) = P(G \oplus \hat{P}_1) \leq P(G' \oplus \hat{P}_1) = P(G')$, so we have that $G \leq_P G' \leq_P G''$ and $e(G) \leq e(G') \leq e(G'')$. Therefore, $G \leq_P G''$ and $e(G) \leq e(G'')$, and it follows that $G \leq'_P G'$.

Lemma 3.3.3. *For any rooted forest operator f (not necessarily in Φ_1), and for any $G_1, \ldots, G_{a(f)}, G'_1, \ldots, G'_{a(f)}$, if $G_i \leq_P G'_i$ for $i = 1, \ldots, a(f)$ then*

$$f(G_1, \ldots, G_{a(f)}) \leq_P f(G'_1, \ldots, G'_{a(f)}).$$

In particular, $\leq_P$ is preserved by Φ_1.

Proof. Let J be some rooted forest. Let J' be the subforest of $f(G'_1, G_2, \ldots, G_{a(f)}) \oplus J$ induced by the vertices of $G_2, \ldots, G_{a(f)}$, and J. The forest $f(G'_1, G_2, \ldots, G_{a(f)}) \oplus J$ is isomorphic to $G'_1 \oplus J'$. Therefore,

$$P(f(G_1, \ldots, G_{a(f)}) \oplus J) = P(G_1 \oplus J') \leq P(G'_1 \oplus J') = P(f(G'_1, G_2, \ldots, G_{a(f)}) \oplus J).$$

Repeating this argument gives that

$$\begin{aligned} P(f(G_1, \ldots, G_{a(f)}) \oplus J) &\leq P(f(G'_1, G_2, \ldots, G_{a(f)}) \oplus J) \\ &\leq P(f(G'_1, G'_2, G_3, \ldots, G_{a(f)}) \oplus J) \\ &\vdots \\ &\leq P(f(G'_1, \ldots, G'_{a(f)}) \oplus J). \end{aligned}$$

■

Lemma 3.3.4. *$\leq'_P$ is a (P, Φ_1)-order.*

Proof. To show that $\leq'_P$ is preserved by Φ_1, let G_1, G_2, G'_1, and G'_2 be rooted forests such that $G_1 \leq'_P G'_1$ and $G_2 \leq'_P G'_2$. If either $P(G_1) = 0$ or $P(G_2) = 0$ then we have $P(G_1 \oplus G_2) = 0$ as P is hereditary, and therefore $G_1 \oplus G_2 \leq'_P G'_1 \oplus G'_2$. Otherwise, $G_1 \leq_P G'_1$ and $G_2 \leq_P G'_2$, so Lemma 3.3.3 implies that $G_1 \oplus G_2 \leq_P G'_1 \oplus G'_2$. Furthermore, $e(G_1) \leq e(G'_1)$ and $e(G_2) \leq e(G'_2)$, so $e(G_1 \oplus G_2) =$

$e(G_1) + e(G_2) \leq e(G'_1) + e(G'_2) = e(G'_1 \oplus G'_2)$. Therefore, $G_1 \oplus G_2 \leq'_P G'_1 \oplus G'_2$. Similarly, we have that if $G \leq'_P G'$ then $\mathrm{nr}(G) \leq'_P \mathrm{nr}(G')$ and $\mathrm{nir}(G) \leq'_P \mathrm{nir}(G')$.

To show that $\leq'_P$ preserves P with size, suppose that $G \leq'_P G'$. If $P(G) = 0$ then clearly $P(G) \leq P(G')$. Otherwise, as $G \leq_P G'$, we have that $P(G) = P(G \oplus \hat{P}_1) \leq P(G' \oplus \hat{P}_1) = P(G')$. Furthermore, if $P(G) = 1$ then by the definition of $\leq'_P$, $e(G) \leq e(G')$. ■

We will use the partial order $\leq'_P$ in Section 3.5 and Section 3.6. We note that it is not hard to show that for any (P, Φ_1)-order $\leq$, $G \leq G'$ implies $G \leq'_P G'$ (Furthermore, for any partial order $\leq$ that preserves P and is preserved by Φ_1, $G \leq G'$ implies $G \leq_P G'$). Therefore $|\leq'_P| \leq |\leq|$, or in other words, $\leq'_P$ is an "optimal" (P, Φ_1)-order.

Let $=_P$ be the following equivalence relation: $G =_P G'$ iff $G \leq_P G'$ and $G' \leq_P G$. Let $|P|$ denotes the number of equivalence classes of $=_P$. Clearly, if $G =_P G'$ then either $G \leq'_P G'$ or $G' \leq'_P G$. Therefore, a complete collection cannot contain forests from the same equivalence class of $=_P$. It follows that $|\leq'_P| \leq |P|$. Since $a(\Phi_1) = 2$ and $\max_{f \in \Phi_1} |\mathrm{sub}(f)| = 2$, we conclude:

Theorem 3.3.5. *The time complexity of the algorithm in the above case is* $O(|P|^3 \cdot T_2(n) \cdot n + |P| \cdot T_1(n))$.

The difference between the approach we described in this section and the approach of Bern el al. [19], is that in the latter, an equivalence relation is used instead of a partial order, and the time complexity depends on the number of equivalence classes of the relation. For some properties, the number of equivalence classes in the appropriate equivalence relation is large, while the value of $|\leq|$ for the appropriate partial order is small. This generalized setting is used in [129] in order to give polynomial algorithms to restrictions of MSP.

3.4 Counting families of matching subsets

In this section we describe and analyze a combinatorial problem. Its solution will be needed for the approximation scheme that we shall develop in the next section and for improving the time complexity of the exact algorithm. The combinatorial problem is interesting in its own right and may have other applications.

Let T be some finite set, and let $\mathcal{S}$ be a multiset whose elements are from T. We define the following mapping $R_{\mathcal{S}}$: Given a multiset of sets $\mathcal{A} = \{A_1, \ldots, A_p\}$ such that each A_i is a subset of T, $R_{\mathcal{S}}(\mathcal{A})$ returns the set of all multisets $\mathcal{B}$ such that $\mathcal{B} \subseteq \mathcal{S}$ and there is an injective mapping $f_{\mathcal{B}} \colon \mathcal{B} \to \mathcal{A}$ such that $b \in f_{\mathcal{B}}(b)$ for every $b \in \mathcal{B}$. In words, $R_{\mathcal{S}}(\mathcal{A})$ is the set of all sub-multisets of $\mathcal{S}$ that can be constructed by taking at most one element from each set A_i. Hence, each sub-multiset is obtained by matching to each of its elements a different set A_i

containing that element. For this reason we call $R_{\mathcal{S}}(\mathcal{A})$ a *matching subsets family*. For example,

$$R_{\{1,2,3\}}(\{\{1,2\},\{1,3\}\}) = \{\phi,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\}\}$$

and

$$R_{\{1,1,2,2,3\}}(\{\{1,2\},\{1,3\}\}) = \{\phi,\{1\},\{1,1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\}\}.$$

For integers n and k, $k \leq n$ we define $r(n,k)$ to be the maximum of $|\text{Image}(R_{\mathcal{S}})|$ taken over all $\mathcal{S}$ such that $|\mathcal{S}| = n$ and $|T| = k$. For example, $r(2,2) = 5$: Take $\mathcal{S} = T = \{1,2\}$ and the five matching subsets families in $\text{Image}(R_{\mathcal{S}})$ are $\{\phi\},\{\phi,\{1\}\},\{\phi,\{2\}\},\{\phi,\{1\},\{2\}\}$ and $\{\phi,\{1\},\{2\},\{1,2\}\}$ (these families are the images of $\phi,\{\{1\}\},\{\{2\}\},\{\{1,2\}\}$ and $\{\{1\},\{2\}\}$, respectively. The other types of sets $\mathcal{S}$ contain identical elements and so have smaller images). Our goal is to obtain upper and lower bounds for the maximum number of distinct matching subsets families as a function of n and k. In particular, we will show that $r(n,k) \leq \binom{n+2^k-1}{n}$.

Lemma 3.4.1. *For every multisets $\mathcal{S}$ and $\mathcal{A}$, there is a multiset $\mathcal{A}' \subseteq \mathcal{A}$, such that $|\mathcal{A}'| \leq |\mathcal{S}|$ and $R_{\mathcal{S}}(\mathcal{A}) = R_{\mathcal{S}}(\mathcal{A}')$.*

Proof. Build a bipartite graph $G = (U,V,E)$, where U is a set of $|\mathcal{S}|$ vertices labeled by the elements of $\mathcal{S}$, V is a set of $|\mathcal{A}|$ vertices labeled by the sets of $\mathcal{A}$, and for a vertex $u \in U$ whose label is i and a vertex $v \in V$ whose label is A, there is an edge $uv \in E$ iff $i \in A$. By a theorem of Dulmage and Mendelsohn (see, e.g., [87, p. 99]), there are a set $X \subseteq U$ with $|N(X)| \leq |X|$ (if $X \neq \phi$ then $|N(X)| < |X|$) and a matching M that matches all the vertices of $U - X$ to vertices in $V - N(X)$. We define $\mathcal{A}'$ to contain all the sets corresponding to vertices in $N(X)$, and all the sets corresponding to vertices of $V - N(X)$ which are matched in M. Clearly, $|\mathcal{A}'| \leq |\mathcal{S}|$ and since $\mathcal{A}' \subseteq \mathcal{A}$, it follows that $R_{\mathcal{S}}(\mathcal{A}') \subseteq R_{\mathcal{S}}(\mathcal{A})$.

To prove that $R_{\mathcal{S}}(\mathcal{A}') \supseteq R_{\mathcal{S}}(\mathcal{A})$, let $\mathcal{B}$ be any multiset in $R_{\mathcal{S}}(\mathcal{A})$. Let $f_{\mathcal{B}}\colon \mathcal{B} \to \mathcal{A}$ be the injection from the definition of R. We define $f'_{\mathcal{B}}\colon \mathcal{B} \to \mathcal{A}'$ in the following way: For every $b \in \mathcal{B}$, if the vertex u corresponding to b is in X, then let $f'_{\mathcal{B}}(b) = f_{\mathcal{B}}(b)$. Otherwise, let $f'_{\mathcal{B}}(b) = A$, where A is the set corresponding to the vertex that is matched to u in M. Since $f'_{\mathcal{B}}$ is an injection, we have that $\mathcal{B} \in R_{\mathcal{S}}(\mathcal{A}')$, and therefore, $R_{\mathcal{S}}(\mathcal{A}') \supseteq R_{\mathcal{S}}(\mathcal{A})$, as required. ∎

Corollary 3.4.2. $r(n,k) \leq \binom{n+2^k-1}{n}$.

Proof. The number of multisets of size at most a whose elements are from a set of size b is equal to the number of ways to partition a balls into $b+1$ cells, which is $\binom{a+b}{a}$. Therefore, as $|T| = k$, the number of multisets of size at most n whose elements are nonempty subsets of T is $\binom{n+2^k-1}{n}$, and the corollary follows from Lemma 3.4.1. ∎

We note that $\binom{n+2^k-1}{n} \leq \min(2^{O(kn)}, \left(\frac{en}{2^k-1}+e\right)^{2^k-1})$. We also give lower bounds on $r(n,k)$:

Lemma 3.4.3. *$r(n,k) \geq \left(\lfloor \frac{n}{k2^{k-1}} \rfloor + 1\right)^{2^k-1}$. Furthermore,*

$$r(n,k) \geq \begin{cases} 2^{\Omega(\frac{n}{\log_k n})} & \text{if } k = \log^{1+\Omega(1)} n \\ 2^{\Omega(k \log n)} & \text{if } k = \Omega(\frac{n}{\log n}) \end{cases}$$

Proof. Let $T = \{1, \ldots, k\}$ and let $\mathcal{S}$ be a some multiset containing each element of T at least $\lfloor n/k \rfloor$ times. We will give lower bound on $r(n,k)$ by giving a lower bound on the size of Image$(R_{\mathcal{S}})$. We call a mapping $g\colon (2^T - \{\phi\}) \to \mathbb{N}$ *bounded* if for all i, $\sum_{X:i\in X} g(X) \leq \lfloor n/k \rfloor$. For a bounded mapping g, we define $\mathcal{A}_g$ to be the multiset of sets obtained by taking each nonempty set $X \subseteq T$ exactly $g(X)$ times.

Claim 3.4.4. *If $g \neq h$ are two bounded mappings, then $R_{\mathcal{S}}(\mathcal{A}_g) \neq R_{\mathcal{S}}(\mathcal{A}_h)$.*

Proof. Let X be a set of minimal size for which $g(X) \neq h(X)$ and w.l.o.g. $g(X) > h(X)$. Build a multiset $\mathcal{B}$ in the following way: Begin with $\mathcal{B} = \phi$. For each $Y \subseteq X$, choose arbitrarily $i \in Y$, and add $g(Y)$ copies of i to $\mathcal{B}$. For each $Y \not\subseteq X$, choose arbitrarily $i \in Y - X$, and add $g(Y)$ copies of i to $\mathcal{B}$. Since g is bounded, we have that the multiplicity of each number i in $\mathcal{B}$ is at most $\lfloor n/k \rfloor$, and therefore $\mathcal{B} \subseteq \mathcal{S}$. Also, it is easy to see that $\mathcal{B} \in R_{\mathcal{S}}(\mathcal{A}_g)$. Let $\mathcal{B}'$ be the multiset obtained from $\mathcal{B}$ by removing all its elements which are in X. As $\mathcal{B}' \subseteq \mathcal{B}$ we have $\mathcal{B}' \in R_{\mathcal{S}}(\mathcal{A}_g)$.

If $\mathcal{B}' \notin R_{\mathcal{S}}(\mathcal{A}_h)$ then we are done. Otherwise, from the definition of $R_{\mathcal{S}}$ there is an injection $f\colon \mathcal{B}' \to \mathcal{A}_h$ such that $i \in f(i)$ for any $i \in \mathcal{B}'$. Clearly, $\sum_{Y:Y\not\subseteq X} g(Y) = |\mathcal{B}'|$ and $\sum_{Y:Y\not\subseteq X} h(Y) \geq |\mathcal{B}'|$ ($\sum_{Y:Y\not\subseteq X} h(Y) - |\mathcal{B}'|$ is the number of sets $Y \in \mathcal{A}_h$ which are not in the image of f and $Y \not\subseteq X$). If $\sum_{Y:Y\not\subseteq X} h(Y) > |\mathcal{B}'|$, select a set $Z \in \mathcal{A}_h$ which is not in the image of f and $Z \not\subseteq X$, and build a multiset $\mathcal{B}''$ by taking $\mathcal{B}'$ and adding a number $i \in Z - X$ chosen arbitrarily. Clearly, $\mathcal{B}'' \in R_{\mathcal{S}}(\mathcal{A}_h)$. However, the number of sets in $\mathcal{A}_g$ which contain at least one element from $\mathcal{B}''$ (i.e., an element from $Y - X$) is exactly $\sum_{Y:Y\not\subseteq X} g(Y) = |\mathcal{B}'|$, which is one less than the size of $\mathcal{B}''$. Hence $\mathcal{B}'' \notin R_{\mathcal{S}}(\mathcal{A}_g)$ so we are done again.

Now, suppose that $\sum_{Y:Y\not\subseteq X} h(Y) = |\mathcal{B}'|$, so $\sum_{Y:Y\not\subseteq X} g(Y) = \sum_{Y:Y\not\subseteq X} h(Y)$. From the minimality of X we have $g(Y) = h(Y)$ for all $Y \subsetneq X$. As $g(X) > h(X)$, we have $|\mathcal{B}| = \sum_Y g(Y) > \sum_Y h(Y)$ which implies that $\mathcal{B} \notin R_{\mathcal{S}}(\mathcal{A}_h)$. ∎

By the above claim, the number of distinct bounded mappings is a lower bound on the size of Image$(R_{\mathcal{S}})$. We will therefore give a lower bound on the number of bounded mappings. If $g(X) \leq \lfloor \frac{n}{k2^{k-1}} \rfloor$ for all X, then g is bounded (as for all i, $\sum_{X:i\in X} g(X) \leq |X : i \in X| \lfloor \frac{n}{k2^{k-1}} \rfloor = 2^{k-1} \lfloor \frac{n}{k2^{k-1}} \rfloor \leq \lfloor \frac{n}{k} \rfloor$). Hence, the number of bounded mappings is at least $\left(\lfloor \frac{n}{k2^{k-1}} \rfloor + 1\right)^{2^k-1}$.

We now give better lower bounds for large values of k. Let $\mathcal{F}$ be a set of nonempty subsets of T such that

$$\text{for all } i, \text{ the number of sets in } \mathcal{F} \text{ that contain } i \text{ is at most } \lfloor n/k \rfloor \tag{3.1}$$

Then any g that satisfies $g(X) \leq 1$ for $X \in \mathcal{F}$ and $g(X) = 0$ otherwise, is a bounded mapping, so the number of bounded mappings is at least $2^{|\mathcal{F}|}$. We now show how to build a set $\mathcal{F}$ that satisfies (3.1) with large size: Let l be the maximum integer such that $\binom{k}{l} \geq \lfloor \frac{n}{k} \rfloor$. If $k = \log^{1+\Omega(1)} n$ then $l = \Theta(\log_k \frac{n}{k})$. First, we assume that $\binom{k}{l} = \lfloor \frac{n}{k} \rfloor$. We take $\mathcal{F}$ to be all the subsets of T of size $l+1$ ($\mathcal{F}$ satisfies (3.1) as for each i, the number of sets in $\mathcal{F}$ that contain i is exactly $\binom{k}{l} = \lfloor \frac{n}{k} \rfloor$). Then

$$|\mathcal{F}| = \binom{k}{l+1} = \frac{k-l}{l+1}\binom{k}{l} = \Omega(\frac{n}{l}) = \Omega(\frac{n}{\log_k n})$$

which gives us the desired lower bound. If $\binom{k}{l} > \lfloor \frac{n}{k} \rfloor$ then let $p = 0.5\lfloor \frac{n}{k} \rfloor / \binom{k}{l}$ and build the set $\mathcal{F}$ by randomly adding each subset of T of size $l+1$ to $\mathcal{F}$ with probability p. Let X_i be the number of sets in $\mathcal{F}$ that contain i. Suppose that $k \leq n/100 \log n$. Using Chernoff bounds [34], we have that with high probability, $X_i \leq 2\mathrm{E}[X_i] = 2p\binom{k}{l} = \lfloor \frac{n}{k} \rfloor$ for all i, so $\mathcal{F}$ satisfies (3.1). Furthermore, with high probability,

$$|F| \geq \frac{1}{2}\mathrm{E}[|F|] = \frac{1}{2}p\binom{k}{l+1} = \frac{1}{2}p\frac{k-l}{l+1}\binom{k}{l} = \Omega(\frac{k}{l}) = \Omega(\frac{n}{\log_k n}).$$

Finally, we prove the third lower bound. Let $T_1, \ldots, T_l$ be a partition of T into disjoint subsets, and define $g(X) = 1$ if $X = T_i$ for some i, and $g(X) = 0$ otherwise. Since such a mapping g is bounded, we conclude that the number of bounded mappings is at least as the number of ways to partition T into disjoint subsets. This number is $2^{\Theta(k \log k)}$ (see, e.g., [100]). Therefore, for $k = n^{\Omega(1)}$ (and in particular, for $k = \Omega(n/\log n)$), the number of bounded mappings is $2^{\Omega(k \log n)}$. ■

Note that for fixed k our bounds are optimal up to constants: In this case, $r(n,k) = \Theta(n^{2^k-1})$. Another interesting case is when $k = n$. In this case we have $2^{\Omega(n \log n)} \leq r(n,n) \leq 2^{O(n^2)}$.

For Section 3.6 we will need a variation of the above problem. For a set T and an integer n, we define a mappings $R_{T,n}$ as follows: Given a multiset $\mathcal{A}$ of subsets of T, $R_{T,n}(\mathcal{A})$ is the set of all multisets of size at most n that can be constructed by taking at most one element from each set in $\mathcal{A}$. In other words, $R_{T,n}(\mathcal{A}) = R_{\mathcal{S}}(\mathcal{A}) \cap U$, where $\mathcal{S}$ is a multiset containing each element of T exactly n times, and U is the set of all multisets of size at most n with elements from T. Therefore, we have $|\mathrm{Image}(R_{T,n})| \leq r(nk,k)$, where $k = |T|$. As for a lower bound on $|\mathrm{Image}(R_{T,n})|$, clearly, $\mathrm{Image}(R_{T,n}) \supseteq \mathrm{Image}(R_{\mathcal{S}})$ for any multiset $\mathcal{S}$ of size n with elements from T. Therefore, $|\mathrm{Image}(R_{T,n})| \geq r(n,k)$.

3.5 An exact algorithm

We now turn to the problem of finding an exact solution to MSP.

Given an instance $(G, \mathcal{H})$ for MSP, we define a graph property P by $P(G) = 1$ iff $\mathcal{H} \not\subseteq G$. We then use algorithm MaxSubforest.

Let H be a tree and v be a vertex in H. We say that a tree J is *v-subtree* of H if J is a rooted subtree of H that is induced by v and one or more connected components of $H - \{v\}$, and whose root is v. J is called a *simple v-subtree* of H if it contains exactly one connected component of $H - \{v\}$. A tree J is called a *cropped v-subtree* if $\mathrm{nr}(J)$ is a simple v-subtree. Let $u_1, \ldots, u_k$ denote all the vertices in all the trees in $\mathcal{H}$. We define for every $i \leq k$ the set F_i to be the set of all distinct (i.e., non-isomorphic) u_i-subtrees of H, where H is the tree from $\mathcal{H}$ that contains u_i. Also, let F_i' be the set of all distinct cropped u_i-subtrees of H, and let F_i'' be the multi-set of all cropped u_i-subtrees of H. Let $d_i = |F_i''|$ (i.e., d_i is the degree of u_i), $D_i = |F_i'|$, and $F = \cup_{i=1}^{k} F_i$. Note that given a set $\mathcal{H}$, the set F can be built in $2^{O(k)}$ time (This is done by building all the u_i-subtrees for every u_i, and then for every two resulting trees checking if they are isomorphic. The time complexity is $O(\sum_{i=1}^{k}(2^{d_i})^2 \cdot k) = 2^{O(k)}$.)

Now, we define a mapping h from rooted forests to 2^F as follows:

$$h(G) = \{H \in F : \mathcal{H} \subseteq G \oplus H\}.$$

Claim 3.5.1. *$G \leq_P G'$ iff $h(G) \supseteq h(G')$.*

Proof. Suppose that $h(G) \supseteq h(G')$ for two rooted forests G, G'. We need to show that for any rooted forest J, $P(G \oplus J) \leq P(G' \oplus J)$. It suffices to show that $P(G' \oplus J) = 0$ implies that $P(G \oplus J) = 0$, or in other words, $\mathcal{H} \subseteq G' \oplus J$ implies $\mathcal{H} \subseteq G \oplus J$. Let J be some rooted forest for which $\mathcal{H} \subseteq G' \oplus J$. There is a subgraph H of $G' \oplus J$ which is isomorphic to a tree from $\mathcal{H}$. Let J_H be the rooted subtree of J which is induced by the vertices of H. We have that $H \subseteq G' \oplus J_H$ and therefore $\mathcal{H} \subseteq G' \oplus J_H$. Furthermore, $J_H \in F$, thus $J_H \in h(G')$. It follows that $J_H \in h(G)$, so we have that $\mathcal{H} \subseteq G \oplus J_H$. Therefore $\mathcal{H} \subseteq G \oplus J$

The second direction is straightforward: If $G \leq_P G'$, then for every rooted forest J, $P(G' \oplus J) = 0$ implies that $P(G \oplus J) = 0$, namely $\mathcal{H} \subseteq G' \oplus J$ implies $\mathcal{H} \subseteq G \oplus J$. In particular, this is satisfied for every $J \in F$, and therefore $h(G) \supseteq h(G')$. ∎

Claim 3.5.1 gives an efficient procedure for checking whether $G \leq_P G'$ and therefore we can use algorithm MaxSubforest. In order to make the algorithm more efficient, we need to be able to compute $h(G')$ efficiently for every forest G' that is built during the execution of the algorithm. Each such forest is of the form $G_1 \oplus G_2$, $\mathrm{nr}(G_1)$ or $\mathrm{nir}(G_1)$. Suppose that $G' = G_1 \oplus G_2$. The sets $h(G_1)$ and $h(G_2)$ were computed previously by the algorithm. Clearly, $h(G') = \{H_1 \oplus H_2 \in F : H_1 \in h(G_1), H_2 \in h(G_2)\}$. Therefore, we can compute

$h(G')$ by building all the forests of the form $H_1 \oplus H_2$ and then for each forest checking if there is a forest in F that is isomorphic to it. The time complexity of this procedure is $O(|\mathcal{C}(G_1)| \cdot |\mathcal{C}(G_2)| \cdot |F|k) = O(|F|^3 k) = 2^{O(k)}$. Similarly, the value of $h(\mathrm{nr}(G_1))$ or $h(\mathrm{nir}(G_1))$ can be computed from $h(G_1)$ in $2^{O(k)}$ time. Checking whether a forest with at most n vertices has property P takes $O((k^{1.5}/\log k)n)$ time by Theorem 2.4.6. Therefore, the time complexity of the MSP algorithm is $O(|P|^2 \cdot 2^{O(k)} \cdot n)$. Clearly $|P| \leq |2^F| = 2^{2^{O(k)}}$.

In the rest of this section, we give a better upper bound on $|P|$, using the results of Section 3.4. For $i = 1, \ldots, k$, define a mapping $h_i \colon \mathcal{R} \to 2^{F_i}$ as follows:

$$h_i(G) = \{H \in F_i : H \subseteq_T G\}.$$

Furthermore, define $\hat{h}(G) = (h_1(G), \ldots, h_k(G))$.

Lemma 3.5.2. *$\hat{h}(G) = \hat{h}(G')$ implies $G =_P G'$.*

Proof. Suppose that $\hat{h}(G) = \hat{h}(G')$ for two rooted forests G, G'. By symmetry, it suffices to show that $\mathcal{H} \subseteq G \oplus J$ for some rooted forest J implies $\mathcal{H} \subseteq G' \oplus J$. Let J be some rooted tree for which $\mathcal{H} \subseteq G \oplus J$, and let H be a subgraph of $G \oplus J$ which is isomorphic to a tree from $\mathcal{H}$. Let G_H be the rooted subtree of G which is induced by the vertices of H. We have that $G_H \subseteq_T G$ and $G_H \in F_i$ for some i, so $G_H \in h_i(G)$. As $\hat{h}(G) = \hat{h}(G')$, we have that $G_H \in h_i(G')$. Therefore $G_H \subseteq_T G'$ and it follows that $H \subseteq G' \oplus J$. ∎

By Lemma 3.5.2, we have that $|P| \leq |\mathrm{Image}(\hat{h})| \leq \prod_{i=1}^{k} |\mathrm{Image}(h_i)|$. We now give an upper bound on $|\mathrm{Image}(h_i)|$ for some fixed i. Define a mapping R' in the following way: Given a multiset of sets $\mathcal{A} = \{A_1, \ldots, A_k\}$, where each A_j is a subset of F_i', let

$$R'(\mathcal{A}) = \{\oplus_{H \in \mathcal{B}} \mathrm{nr}(H) : \mathcal{B} \in R_{F_i''}(\mathcal{A}), \mathcal{B} \neq \phi\} \cup \{\hat{P}_1\}$$

i.e., for every nonempty multiset $\mathcal{B}$ from $R_{F_i''}(\mathcal{A})$, $R'(\mathcal{A})$ contains the rooted tree formed by taking the trees in $\mathcal{B}$ and connecting their roots to a new vertex that becomes the new root. By definition, $|\mathrm{Image}(R')| = |\mathrm{Image}(R_{F_i''})| \leq r(d_i, D_i)$.

Let G be some rooted forest and let $J \in F_i$. We decompose G by $G = \mathrm{nr}(G_1) \oplus \cdots \oplus \mathrm{nr}(G_c)$, where c is the number of the children of the root of G. Also, we decompose J by $J = \mathrm{nr}(J_1) \oplus \cdots \oplus \mathrm{nr}(J_d)$, where each J_j is in F_i'. For each $j \leq c$, let $A_j = \{H \in F_i' : H \subseteq_T G_j\}$. Then, $J \subseteq_T G$ iff for each $j \leq d$, there is a distinct j' such that $J_j \in A_{j'}$. This is a case of the matching subsets problem:

$$\begin{aligned} J \subseteq_T G &\iff \{J_1, \ldots, J_d\} \in R_{F_i''}(\{A_1, \ldots, A_c\}) \\ &\iff J \in R'(\{A_1, \ldots, A_c\}). \end{aligned}$$

(See Figure 3.4 for an example). Since every tree in $R'(\{A_1, \ldots, A_c\})$ is in F_i, we

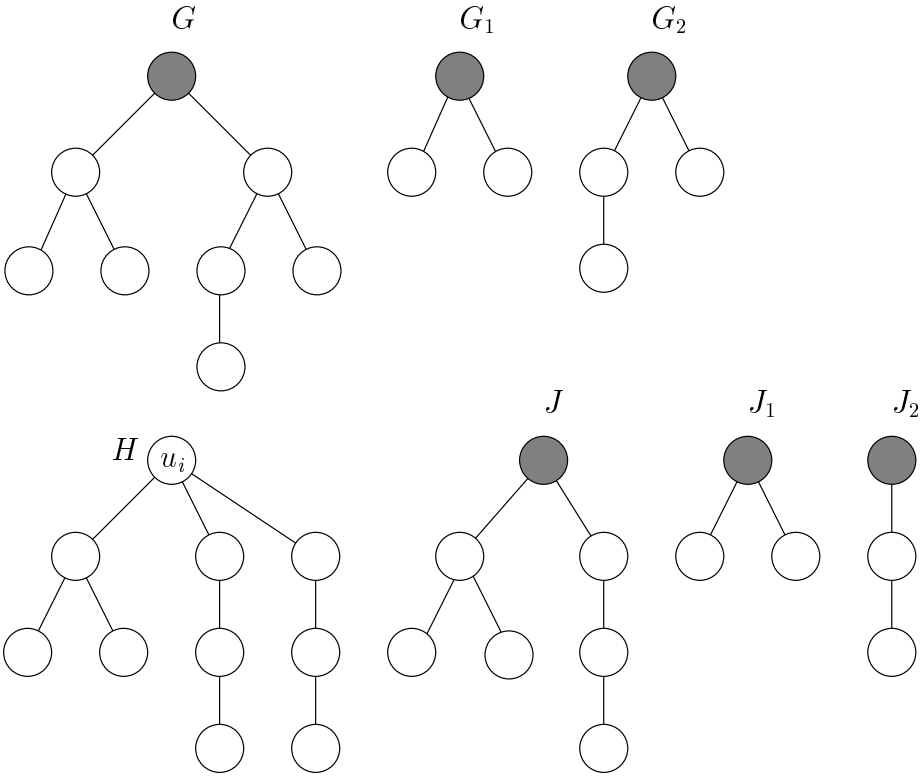

Figure 3.4: The connection between MSP and the matching subsets problem. Suppose that $\mathcal{H}$ consists of the tree H. We have $F_i' = \{J_1, J_2\}$ and $F_i'' = \{J_1, J_2, J_2\}$. Consider the the rooted forest G and the rooted tree J from F_i. We decompose G by $G = \mathrm{nr}(G_1) \oplus \mathrm{nr}(G_2)$, and decompose J by $J = \mathrm{nr}(J_1) \oplus \mathrm{nr}(J_2)$. We have $A_1 = \{J_1\}$ and $A_2 = \{J_1, J_2\}$, so $\{J_1, J_2\} \in R_{F_i''}(\{A_1, A_2\}) = \{\phi, \{J_1\}, \{J_2\}, \{J_1, J_2\}\}$. Therefore, $J \subseteq_T G$. More generally, we have that $h_i(G) = R'(\{A_1, A_2\}) = \{\hat{P}_1, \mathrm{nr}(J_1), \mathrm{nr}(J_2), J\}$.

have that $h_i(G) = R'(\{A_1, \ldots, A_c\})$. Hence,

$$|\text{Image}(h_i)| \leq |\text{Image}(R')| \leq r(d_i, D_i).$$

Using Corollary 3.4.2 we have

$$|P| \leq \prod_{i=1}^{k} r(d_i, D_i) \leq \prod_{i=1}^{k} 2^{O(d_i D_i)} \leq 2^{O(\sum_{i=1}^{k} d_i D_i)} = 2^{O(k^2/\log k)},$$

where the last equality follows from an analysis similar to the one in Lemma 2.4.5. We therefore have the following theorem:

Theorem 3.5.3. *The maximum subforest problem can be solved in* $2^{O(k^2/\log k)}n$ *time.*

We also note that the same approach can be used to solve the maximum subgraph problem for the case when G is a sparse graph: If G is a connected graph with n vertices and $n + p$ edges then we can find a maximum $\mathcal{H}$-free subgraph of G in $2^{O(k^5)}n + 2^{O(k^5 p)}$ time.

3.6 A polynomial time approximation scheme

In this section we give a polynomial time approximation scheme for MSP. The basic idea behind the algorithm is as follows: We partition the trees in $\mathcal{H}$ into two kinds: "small" trees and "big" trees. We first find an *optimal* subforest G' of G that does not contain a subforest that is isomorphic to a "small" tree. Due to the "smallness" of the forbidden trees, this stage will be done in polynomial time. In the second stage, we need to remove edges from G' such that at least one edge is removed from each subforest that is isomorphic to a "big" tree. Since each of these trees is "big", we can perform this task by deleting at most a constant fraction of the edges in G'.

A vertex x in a tree H is called an *l-separator* if every connected component in $H - \{x\}$ has at most l vertices. A tree H is called *l-separable* if it has an l-separator. We now describe a family $\{A_l\}_{l \geq 1}$ of approximation algorithms for MSP. Algorithm A_l is as follows:

1. Let $\mathcal{H}_l = \{H \in \mathcal{H} : H \text{ is } l\text{-separable}\}$. Find a maximum $\mathcal{H}_l$-free subforest of G, and call it G'.

2. Go over all connected components of G'. For each component T, select a root r and scan its vertices in postorder. When reaching a vertex $v \neq r$ such that the number of unmarked descendent vertices of v (including v) is at least $l + 1$, tag the edge between v and its parent and also mark v and its descendents.

3. Output the untagged edges of G'.

Lemma 3.6.1. *Algorithm A_l returns a $\mathcal{H}$-free subforest of G with a number of edges that is at least $l/(l+1)$ times the number of edges in a maximum $\mathcal{H}$-free subforest of G.*

Proof. We denote by G_A the solution returned by the algorithm. We first show that G_A is $\mathcal{H}$-free. Suppose that $H \subseteq G_A$ for some $H \in \mathcal{H}$. Since H is connected, we have that H is isomorphic to a subtree of some connected component S of G_A. Step 2 of the algorithm ensures that in G_A every connected component is l-separable. Since H is isomorphic to a subtree of S, we have that H is also l-separable, hence $H \in \mathcal{H}_l$. But as $H \subseteq G_A$, we conclude that $H \subseteq G'$, a contradiction to the fact that G' is $\mathcal{H}_l$-free. Therefore, $\mathcal{H} \not\subseteq G_A$.

We now prove the stated approximation factor. Note that for each edge of G' that was tagged in step 2, there are at least l edges which were not tagged (the edges in the subtree under the tagged edge). Hence, $e(G_A) \geq \frac{l}{l+1} e(G')$.

Let OPT denote the number of edges in a maximum $\mathcal{H}$-free subforest of G. Since $\mathcal{H}_l$ is a subset of $\mathcal{H}$, we have that the number of edges in a maximum $\mathcal{H}_l$-free subforest of G is at least OPT, namely, $e(G') \geq \text{OPT}$. Thus, $e(G_A) \geq \frac{l}{l+1}\text{OPT}$. ∎

To show that the algorithms family $\{A_l\}_{l \geq 1}$ is a polynomial time approximation scheme, we need to show that for every fixed l, finding the maximum $\mathcal{H}_l$-free subforest of a tree G is polynomial in n and k. We now proceed to give an algorithm for this problem, based on the framework described in Section 3.3.

Define a property P by by $P(G) = 1$ iff $\mathcal{H}_l \not\subseteq G$. From the previous section, the time for finding the maximum $\mathcal{H}_l$-free subforest of a tree G is $O(|P|^2 \cdot |F|^3 \cdot kn)$. The bounds on $|P|$ and $|F|$ which were given in the previous section do not suffice here. We will give better bounds using the fact that the trees in $\mathcal{H}_l$ are l-separable.

We denote by d the maximum degree of a vertex in a tree in $\mathcal{H}$. Let B_0 be the set of all l-separable rooted trees with degree at most d, and let B be the set of all distinct rooted trees in B_0 in which the root is an l-separator. Since $F \subseteq B_0$, we have that $|F| \leq |B_0|$. Any tree in B_0 can be obtained by taking a tree in B, and choosing a new root. A tree in B has at most $dl+1$ vertices, hence $|B_0| \leq (dl+1)|B|$.

Let C be the set of all rooted trees with at most l vertices and degree at most d, and let $q = |C|$. It is known that $q = 2^{\Theta(l)}$ (see Lemma 2.4.2). Any tree in B can be obtained by choosing at most d trees from C, with repetitions, and connecting their roots to a new vertex, and conversely, any tree that is built using this process is in B. Therefore $|B| = \binom{d+q}{q} = O(d^q)$ (the last equality follows from the fact that l is constant, so q is also constant), so $|F| \leq (dl+1)|B| = d^{2^{O(l)}}$.

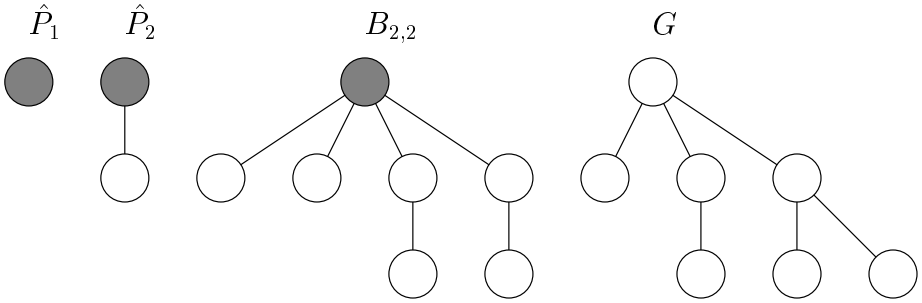

Figure 3.5: Example of the definition of h. Here $l = 2$, $d = 10$. In this case the set C consists of the two rooted trees $\hat{P}_1$ and $\hat{P}_2$. For any pair x, y of positive integers with $x + y \leq 10$ we match the rooted tree $B_{x,y}$ from B which is obtained by taking x copies of $\hat{P}_1$ and y copies of $\hat{P}_2$, adding a new vertex and connecting it by edges to all the roots of the trees, and making the new vertex the new root. If $\mathcal{H}_l$ consists of an unrooted tree isomorphic to $B_{2,2}$, then for the graph G above, $h_1(G) = \{B_{x,y} : 0 \leq y \leq 2, 0 \leq x \leq 3 - y\}$ and $h_2(G) = \{\hat{P}_2\}$.

Define mappings $h_1, h_2, \hat{h}$ as follows:

$$h_1(G) = \{H \in B : H \subseteq_T G\}$$
$$h_2(G) = \{H \in C : \mathcal{H}_l \subseteq G \oplus H\}$$
$$\hat{h}(G) = (h_1(G), h_2(G)).$$

For an example of these definitions, see Figure 3.5.

Lemma 3.6.2. *$\hat{h}(G) = \hat{h}(G')$ implies $G =_P G'$.*

Proof. Suppose that $\hat{h}(G) = \hat{h}(G')$ for two rooted forest G, G'. We need to show that $\mathcal{H}_l \subseteq G \oplus J$ for some rooted forest J implies $\mathcal{H}_l \subseteq G' \oplus J$.

Let J be some rooted tree for which $\mathcal{H} \subseteq G \oplus J$, and let H be a subgraph of $G \oplus J$ which is isomorphic to a tree from $\mathcal{H}$. The tree H is l-separable, and let x be an l-separator of J. Let G_H and J_H be the rooted subforests of G and J, respectively, which are induced by the vertices of H. We consider two cases.

If x is the root of G_H (and J_H), then the root of G_H is an l-separator of G_H. If x is a vertex from J_H (except the root), then G_H is a subgraph of some connected component of $H - \{x\}$ and therefore the number of vertices in G_H is at most l. In particular, the root of G_H is an l-separator of G_H. In both cases it follows that $G_H \in B$ and since $G_H \subseteq_T G$ we have that $G_H \in h_1(G) = h_1(G')$. Therefore, $G_H \subseteq_T G'$ so $H = G_H \oplus J_H \subseteq G' \oplus J$. Hence, $\mathcal{H}_l \subseteq G' \oplus J$.

Now suppose that x in a vertex from G_H which is not the root. J_H is a subgraph of some connected component of $H - \{x\}$ and therefore the number of vertices in J_H is at most l. Thus, we have $J_H \in C$. As $H \subseteq G \oplus J_H$, we have $\mathcal{H}_l \subseteq G \oplus J_H$ implying that $J_H \in h_2(G) = h_2(G')$. Thus, $\mathcal{H}_l \subseteq G' \oplus J_H$ and we conclude that $\mathcal{H}_l \subseteq G' \oplus J$. ■

By Lemma 3.6.2 we have that

$$|P| \leq |\text{Image}(\hat{h})| \leq |\text{Image}(h_1)| \cdot |\text{Image}(h_2)|.$$

Clearly, $|\text{Image}(h_2)| \leq |2^C| = 2^q$. Define a mapping R' in the following way: Given a multiset of sets $\mathcal{A} = \{A_1, \ldots, A_k\}$, where each A_i is a subset of C, let

$$R'(\mathcal{A}) = \{\oplus_{H \in \mathcal{B}} \text{nr}(H) : \mathcal{B} \in R_{C,d}(\mathcal{A})\}.$$

Similarly to Section 3.5, we have that for any rooted tree G, $h_1(G) = R'(\mathcal{A}) \cap B$ for some $\mathcal{A}$. Hence,

$$|\text{Image}(h_1)| \leq |\text{Image}(R')| = |\text{Image}(R_{C,d})| \leq r(qd, d) \leq (qd)^{2^q - 1},$$

and therefore $|P| = d^{2^{2^{O(l)}}}$. We therefore proved the following theorem:

Theorem 3.6.3. *The algorithms family $\{A_l\}_{l \geq 1}$ is a polynomial time approximation scheme for the maximum subforest problem. In particular, algorithm A_l gives an $l/(l+1)$-approximation in $d^{2^{2^{O(l)}}} n$ time.*

The above theorem is of theoretical interest, but practically, the time complexity of the above approximation scheme is too large even for modest values of l. The bottleneck in the complexity is finding a maximum $\mathcal{H}$-free subforest of a tree G, where all trees in $\mathcal{H}$ are l-separable. However, we can give faster implementations of A_1 and A_2. To improve the running time of A_2 we need to describe how to find a maximum $\mathcal{H}$-free subforest of a tree G, for a set of 2-separable trees $\mathcal{H}$. Each 2-separable tree has diameter of at most 4, and by a result from [129], finding such subforest can be done in $O(|\mathcal{H}|n)$ time. Hence, A_2 can be implemented in $O(pn)$ time, where p is the number of 2-separable trees in $\mathcal{H}$ (note that $p \leq \binom{d+2}{2} = O(d^2)$).

The same algorithm can also find a $\mathcal{H}$-free subforest of a tree when all trees in $\mathcal{H}$ are 1-separable in $O(n)$ time, because in this case we can assume that $\mathcal{H}$ contains a single tree H (if it is not the case, we can remove from $\mathcal{H}$ all its trees but the smallest one). In fact, for this case we can give a much simpler algorithm. The algorithm is given in Figure 3.6. Hence, A_1 can be implemented in $O(n)$ time.

3.7 Hardness of the Tree Deletion Problem

In this section we prove several hardness of approximation results for the Tree Deletion problem. Those results imply also NP-hardness of the decision version of MSP, under the same restrictions.

Input: A tree G and a tree $H = K_{1,d}$.
Output: A maximum H-free subforest G' of G.

Select a vertex r of G to be the root of G.
for all vertices v in a postorder of G^r **do**
  if $v \neq r$ and the degree of v is at least d
    then remove the edge between v and its parent.
  while the degree of v is at least d **do**
    remove an edge between v and some child of v.
end for
return G.

Figure 3.6: Algorithm for the maximum subforest problem when the obstruction set consists of a single 1-separable tree.

Theorem 3.7.1. *If* P $\neq$ NP, *then there is a constant c such that there is no polynomial-time approximation algorithm for TDP with approximation factor less than $c \log k$. This result holds even under each of the following restrictions of TDP:*

1. *G and each tree in $\mathcal{H}$ has diameter of 6.*

2. *$\mathcal{H}$ consists of one tree with diameter 8.*

Furthermore, if P $\neq$ NP *then there is a constant c' such that there is no polynomial-time approximation algorithm with approximation factor less than $1 + c'$ for the following restrictions of TDP:*

3. *Each tree in $\mathcal{H}$ has maximum degree of 3 and at most two vertices with degree 3.*

4. *$\mathcal{H}$ consists of one tree with maximum degree of 3.*

Proof. All the reductions in this proof are from Hitting Set or a restriction of this problem. The input to the hitting set problem is sets $R_1, \ldots, R_m$ which are subsets of $S = \{1, \ldots, n\}$. The goal is to find a minimal subset $U \subseteq S$ such that $U \cap R_i \neq \phi$ for $i = 1, \ldots, m$. We can assume w.l.o.g. that each R_i has at least two elements. It was shown in [112] that if P $\neq$ NP then there is a constant c_0 such that there is no approximation algorithm for Hitting Set with approximation factor less than $c_0 \log n$.

We first prove case 1. Given an input $R_1, \ldots, R_m$ to the hitting set problem, we build rooted trees $T_1, \ldots, T_n$ by taking $T_i = B_{2n-2i,i}$, where the definition of $B_{x,y}$ was given in Figure 3.5 (see also Figure 3.7). Note that $T_i \not\subseteq_T T_j$ for every $i \neq j$, and all the trees have height 2. For every set R_i, we build a tree H_i by taking a vertex named v_i, and a copy of the tree T_j for every $j \in R_i$, and adding

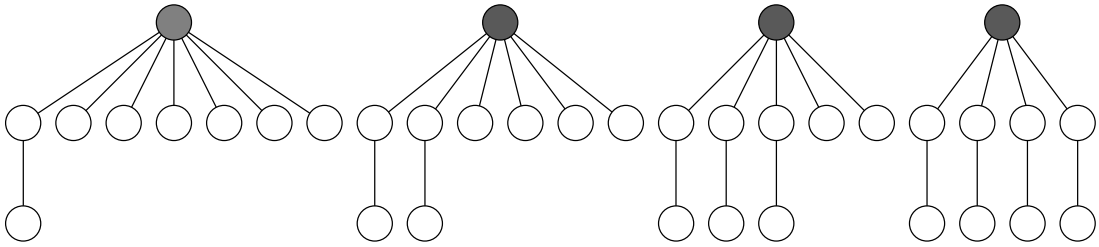

Figure 3.7: The constructions of the rooted trees $T_1, \ldots, T_n$ in case 1 for $n = 4$.

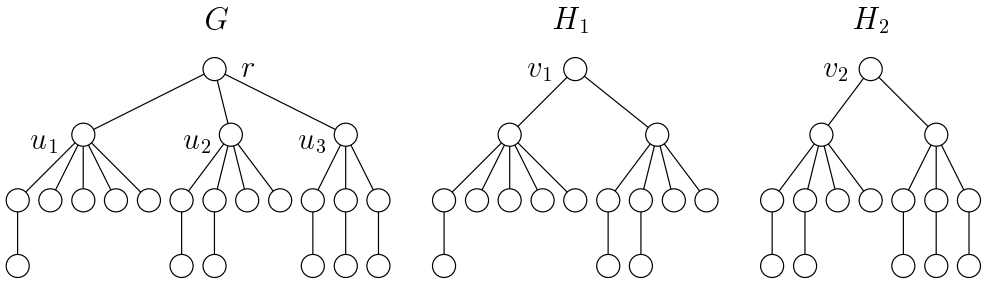

Figure 3.8: The reduction of case 1 for the input $R_1 = \{1, 2\}, R_2 = \{2, 3\}$.

edges between the roots of these trees and v_i. We define $\mathcal{H} = \{H_1, \ldots, H_m\}$. We build the tree G by taking the trees $T_1, \ldots, T_n$, adding a vertex r, and adding edges between r and the roots of $T_1, \ldots, T_n$. Denote by $u_1, \ldots, u_n$ the roots of the trees $T_1, \ldots, T_n$ in G, respectively. See Figure 3.8 for an example of this reduction. Clearly, G and the trees in $\mathcal{H}$ have diameter of 6.

For each $i \leq m$, we denote by G_i the subtree induced from G by the vertex r and the vertices of T_j for all $j \in R_i$. Clearly, G_i is isomorphic to H_i. Moreover, we claim that G_i is the only subgraph of G which is isomorphic to H_i. To show this claim, suppose that G' is a subtree of G which is isomorphic to H_i. We now argue which vertices in G can match v_i under the isomorphism. Since the trees $T_1, \ldots, T_n$ have height 2, it follows that in H_i the vertex v_i is the center of a path of length 6. Furthermore, r is the only vertex in G with this property. Therefore, G' must contain r, and the isomorphism between H_i and G' matches v_i to r. Each neighbor v of v_i is a root of a copy of some tree T_j, and is matched by the isomorphism to the root of some $T_{j'}$. Since $T_j \not\subseteq_T T_{j'}$ for $j \neq j'$, it follows that $j = j'$, and therefore $G' = G_i$. This completes the proof of the claim.

Thus, for a set A of edges of G that are incident on r, we have that $H_i \not\subseteq G - A$ iff A contains an edge (r, u_j) for some $j \in R_i$. Therefore, given a hitting set U of $T_1, \ldots, T_m$ of size k, the set $\{(r, u_i) : i \in U\}$ is a deletion set of $(G, \mathcal{H})$. Conversely, let A be a deletion set of $(G, \mathcal{H})$ of size k, and suppose that A contains an edge $e = (u, v)$ which is not incident on r. Let w be vertex after r on the path from r to u in G. Then, $A \cup \{(r, w)\} - \{e\}$ is also a deletion set of $(G, \mathcal{H})$. By repeating this argument, we obtain a deletion set A' of $(G, \mathcal{H})$ such that all the edges in

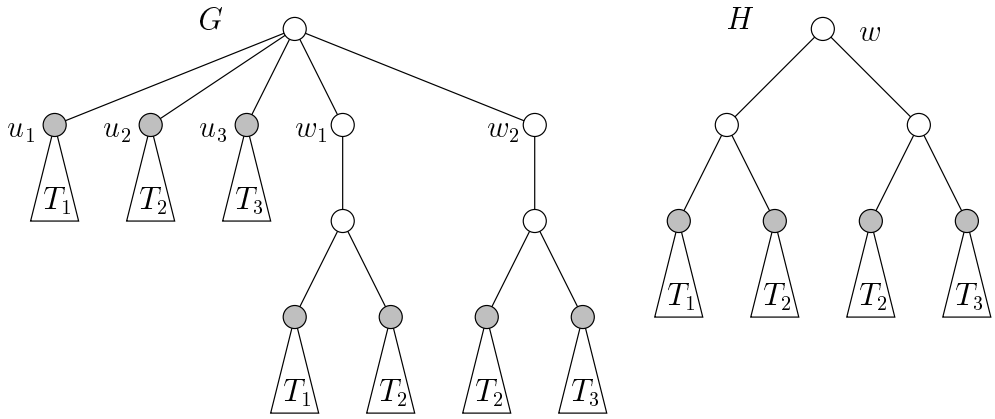

Figure 3.9: The reduction of case 2 for the input $R_1 = \{1, 2\}, R_2 = \{2, 3\}$. The heavy vertices are highlighted.

A' are incident on r and $|A'| \leq |A|$. Then, $\{i : (r, u_i) \in A'\}$ is a hitting set of $T_1, \ldots, T_m$ of size k. The correctness of the reduction follows.

We now deal with case 2. Given subsets $R_1, \ldots, R_m$, we build tree $T_1, \ldots, T_n$ where $T_i = B_{2n-2i+m+1,i}$. Then, we build trees $H_1, \ldots, H_m$ using $T_1, \ldots, T_n$ as in case 1. We build a tree H by taking a vertex w, and the trees $H_1, \ldots, H_m$, and adding an edge between w and v_i for every $i \leq m$. Let $\mathcal{H} = \{H\}$. For every $i \leq m$, define $H^i = H_w^{v_i}$, i.e., H^i is the rooted tree formed by taking H, removing the vertices of H_i, and choosing w as the root. The tree G is built by taking a vertex named r, the trees $T_1, \ldots, T_n$, and the trees $H^1, \ldots, H^m$ and adding edges between r and the roots of $T_1, \ldots, T_n, H^1, \ldots, H^m$. Denote by $u_1, \ldots u_n$ the roots of $T_1, \ldots, T_n$ in G and by $w_1, \ldots, w_m$ the roots of $H^1, \ldots, H^m$. See See Figure 3.9 for an example. Note that H has diameter of 8 and G has diameter of 10.

For each $i \leq m$, we denote by G_i the subtree induced from G by the vertex r, the vertices of H^i, and the vertices of T_j for all $j \in R_i$. Each subtree G_i is isomorphic to H. We claim that no other subtree of G is isomorphic to H. The correctness of the reduction follows from this claim in the same fashion as in the proof of case 1. We now prove the claim: Let G' be a subtree of G which is isomorphic to H. We say that a vertex is *heavy* if its degree is at least $m + n + 1$. Clearly, the isomorphism between H and G' must map a heavy vertex in H to a heavy vertex in G'. In H, the heavy vertices are those with distance 2 from w, or in other words, the roots of the all the copies of $T_1, \ldots, T_n$ in H. In G, the heavy vertices are $u_1, \ldots, u_n$ and descendents of w_i with distance 2 from w_i for $i = 1, \ldots, m$, or in other words, the roots of all the copies of $T_1, \ldots, T_n$ in G. We now argue which vertices in G can match w under the isomorphism. w is the center of a path of length 4 whose end vertices are heavy. The only vertices in G with this property are $u_1, \ldots, u_m$. Therefore, the isomorphism between H and G' maps w to some vertex w_i. Furthermore, the heavy vertices with distance 2 from

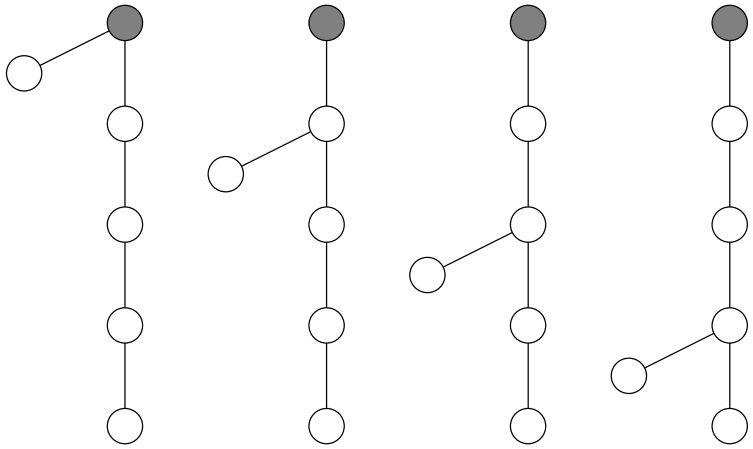

Figure 3.10: The constructions of the rooted trees $T_1, \ldots, T_n$ in case 3 for $n = 4$.

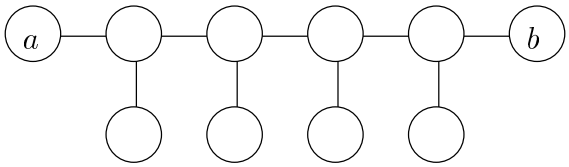

Figure 3.11: The tree J.

w_i are $u_1, \ldots, u_n$ and the descendents of w_i with distance 2 from w_i. It follows that $G' = G_i$.

For case 3, we give a reduction from a restriction of the hitting set problem in which the sets $R_1, \ldots, R_m$ have size exactly 2. This problem is equivalent to Vertex Cover, and therefore, if P $\neq$ NP it cannot be approximated within a factor of 1.166 [125]. Given sets $R_1, \ldots, R_m$ and a positive integer k, we build trees $T_1, \ldots, T_n$ as follows: The tree T_i is built by taking a copy of the tree $\hat{P}_{n+1}$ (the rooted path on $n+1$ vertices) and adding a new vertex which is connected to the vertex at distance $i-1$ from the root. See Figure 3.10 for an example. We then generate $\mathcal{H} = \{H_1, \ldots, H_m\}$ and G from $T_1, \ldots, T_n$ like in the reduction of case 1. Clearly, as each T_i has one vertex of degree 3 and the rest of its vertices have degree of at most 2, the restrictions are satisfied. The proof of the correctness of the reduction is similar to the proof in case 1, and thus is omitted.

We also provide a reduction from Vertex Cover in case 4. We begin by building trees $T_1, \ldots, T_n$ as in the construction of case 3. For each $i \leq n$ we build a tree T'_i by taking T_i and a copy for the tree J which is given in Figure 3.11, adding an edge between the root of T_i and a, and making b the new root. We build the trees $H_1, \ldots, H_m$ from $T'_1, \ldots, T'_n$ like in case 1. Then, we take a path of length $2m-1$ whose vertices are $w_1, \ldots, w_{2m-1}$. We add an edge $w_{2i-1}v_i$ for every $i \leq m$. The result is the tree H, and $\mathcal{H} = \{H\}$. For $i \leq m$, define $H^i = H^{v_i}_{w_{2i-1}}$. We build the tree G from $T'_1, \ldots, T'_n$ and $H^1, \ldots, H^m$ like in case 2. Note that all the subgraphs

of H and G that are isomorphic to J are due to copies of $T'_1, \ldots T'_n$ in H and G. Hence the copies of J play the same role of restricting the isomorphism as the heavy vertices in case 2. The correctness of the reduction follows from this fact, as in case 2. ■

In [129], we show that MSP is polynomial if each tree in $\mathcal{H}$ has diameter of at most 5. We also show that MSP is polynomial if each tree in $\mathcal{H}$ has at most one vertex with degree more than 2. These results are tight with respect to the restricting parameter, due to cases 1 and 3 of Theorem 3.7.1.

Chapter 4

Clustering

4.1 Introduction

Clustering is a fundamental problem that has applications in many areas. The input to the clustering problem is a set of objects and additional information which is either (1) similarity values for every pair of objects, or (2) an attributes vector for each object. The goal is to partition the objects into disjoint sets, called *clusters*, such that two criteria are met: The objects in each set have high similarity to each other (homogeneity), and objects in different sets have low similarity to each other (separation). There are several formulations for the clustering problem, as homogeneity and separation can be defined in many ways. In some applications, the number of clusters is known in advance, so the corresponding problem is to partition the objects into a given number of clusters.

An example of the application of clustering is analysis of gene expression data: Using DNA chips, it is possible to measure the expression levels of many genes in cells of a tissue at some condition. Repeating the experiment several times with different tissues or under different conditions, gives a matrix $L = \{l_{i,j}\}$, where $l_{i,j}$ is the expression level of gene i at experiment j. The i-th row of L is called the *expression pattern* of gene i. By partitioning the genes into clusters using the expression patterns as attributes vectors, one can learn about gene functionality, as genes from the same cluster are likely to have similar function. Alternatively, it is possible to partition the tissues using the columns of L as the attributes vectors. This can be used, for example, to distinguish between tumor and normal tissues.

When the input contains similarity scores, we can represent the input using a *weighted similarity graph*. Let $S_{v,w}$ denote the similarity score between v and w. In the similarity graph $G = (V, E)$, V is the set of all objects, and for every pair of objects v, w, there is an edge $(v, w) \in E$ with weight $S_{v,w}$. The problem can be simplified by transforming the similarity scores to binary similarity values using some threshold T, namely, $S'_{v,w} = 1$ if $S_{v,w} \geq T$ and $S'_{v,w} = 0$ otherwise.

Then, the input can be represented using a *similarity graph* $G = (V, E)$, where V is the set of all objects, and $(v, w) \in E$ iff $S'_{v,w} = 1$.

There are many algorithms for clustering, which use the attributes vectors, the similarity scores, or the similarity graph. For background and literature on clustering algorithms see, e.g., [52, 64, 65, 98]. We give here only a few examples.

The HCS algorithm [66] works in a recursive manner: Given a similarity graph G, it finds a minimum cut of G. If the size of the cut is bigger than a given bound, then the algorithm outputs G. Otherwise, the algorithm is called recursively on the two subgraphs of G induced by the two sides of the cut. The CLICK algorithm [123] is also based on minimum cuts, but it works on the weighted similarity graph and invokes probabilistic considerations.

The geometric k-clustering problem is as follows: Given n points in $\mathbb{R}^d$, partition the points into k disjoint sets $X_1, \ldots, X_k$ and find center points $c_1, \ldots, c_k$ such that $\sum_{i=1}^{k} \sum_{x \in X_i} D(x, c_i)$ is minimum, where D is some distance function. Finding an optimal solution is NP-hard, even when $k = 2$, when the distance function is squared Euclidean distance [45]. The problem can be solved in $O(n^{dk+O(1)})$ time [71], which is polynomial if d and k are fixed. Ostrovsky and Rabani [101] gave a polynomial time approximation scheme for the problem when k is fixed, and D is either the Euclidean distance, squared Euclidean distance, or L_1 distance. Arora et al. [11] gave a polynomial time approximation scheme for the problem when $d = 2$, and D is the Euclidean distance.

In practice, heuristic algorithms are often used to solve clustering problems. For the geometric k-clustering problem, the K-means algorithm [14, 90] uses hill climbing: The algorithm begins with an arbitrary partition of the objects into k sets, and then performs iterations that change the partition by moving one element from one cluster to a different cluster, until there is no such change that decreases the objective function. At each iteration, the change that gives the maximum decrease in the objective function is chosen. Another heuristic is due to Lloyd [84]: The algorithm begins from an arbitrary partition. At each iteration, the algorithm computes the centers $c_1, \ldots, c_k$ of the current clusters, and then creates a new partition into clusters by assigning element i to cluster j if c_j is the closest center to the vector of i. In both algorithms the number of clusters is prespecified.

We now describe a formulation of the clustering problem due to Ben-Dor et al. [17], which will be the basis to our study. A *clique graph* is a graph in which every connected component is a clique. There is a one-to-one correspondence between the clique graphs on a vertex set V, and the partitions of V into clusters: For a clique graph G, the corresponding partition has a cluster for each clique of G. The *distance* between two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ is defined as $D(G_1, G_2) = |E_1 \triangle E_2|$. The clustering problem can be formulated as follows: Given an input graph G, find a nearest clique graph to G, namely, a clique graph G' such that $D(G, G')$ is minimum. This problem is called the *cluster editing problem*. Sharan and Shamir showed that the cluster editing problem is NP-

hard [123]. A related problem is the *cluster deletion problem* which is to find a nearest clique graph to the input graph G amongst the subgraphs of G. This problem is also NP-hard [99].

Since the cluster editing problem is hard, additional assumptions are made in order to solve it efficiently. Ben-Dor et al. [17] assumed the following random model: Suppose that V is partitioned into disjoint sets $V_1, \ldots, V_m$ that are called the *underlying clusters*. Every two vertices from the same cluster are connected by an edge with probability p, and each two vertices from different clusters are connected by an edge with probability $r < p$. The random choices are independent. A graph that is built using this process is called a *random cluster graph*. For a random cluster graph G, we denote by G^* the clique graph that corresponds to the underlying clusters.

The random cluster graph models a common situation in experimental data, where noise obscures the true clusters: The probability $1-p$ is the *false negative* probability, i.e., the chance of incorrectly having no edge between two vertices of a cluster. Similarly, r is the *false positive* probability. If errors are independent and identically distributed, one gets the above model.

When addressing the clustering editing problem under the random graph model, several parameters are interrelated: Obviously, the smaller the gap $\Delta = p - r$, the harder the problem. Also, the size of the smallest cluster $k = \min_i |V_i|$ is limiting the performance, as very small clusters may be undetectable due to noise. The value k also bounds the number of clusters m. The challenge is to obtain provably good performance for a wide range of values for each parameter.

Denote the number of vertices in the graph by n. Ben-Dor et al. [16, 17] gave an algorithm that solves the cluster editing problem on a random cluster graph with high probability, under the assumptions that $\Delta = \Omega(1)$, $p = 1 - r$, and $k = \Omega(n)$. Moreover, Ben-Dor et al. [16] showed that under the assumptions above, with high probability, the graph G^* is the nearest clique graph to a random cluster graph G.

Similar results were given for the minimum bisection problem. The *minimum bisection problem* requires finding in a graph $G = (V, E)$, a set $S \subseteq V$ such that $|S| = n/2$ and the number of edges with one endpoint in S and the other endpoint in $V - S$ is minimum. One can show that this is equivalent to the problem of finding a clique graph $G' = (V, E')$ with two equal-sized cliques, such that $|E - E'|$ is minimum. The minimum bisection problem is NP-hard [58]. Consider the problem under a random cluster graph model: Assume that the input is a random cluster graph G with two equal sized underlying clusters. It is known (cf. [31]) that if $\Delta = \Omega(\sqrt{p} n^{-1/2} \sqrt{\log n})$, then with high probability, the optimal solution to the minimum bisection problem is the clique graph G^*. The first result for the minimum bisection problem on this model was given by Dyer and Frieze [48]. The main result of Dyer and Frieze is an algorithm that always solves the minimum bisection problem, provided that $\Delta = \Omega(n^{-1/4} \log^{1/4} n)$. The expected running time of the algorithm is $O(n^3)$. They also gave an algorithm

that solves the minimum bisection problem with high probability in $O(n^2)$ time, under the same assumption on Δ. The latter algorithm is very simple: First some vertex u is chosen, and then the $n/2$ vertices with highest values of $d_{N(u)}(v)$ are taken into S. Using Chernoff bounds [34], it is easy to show that with high probability, the set S consists of the vertices of the underlying cluster that contains u.

Boppana [28] gave a polynomial time algorithm that solves the the minimum bisection problem with high probability, assuming that $\Delta = \Omega(\sqrt{p}n^{-1/2}\sqrt{\log n})$. The algorithm of Boppana is based on eigenvalue methods. The algorithm uses the ellipsoid algorithm, so it has high running time in practice. Feige and Kilian [54] extended the algorithm of Boppana to a semi-random graph model: After the edges of the graphs are randomly chosen, an adversary can add edges between two vertices from the same cluster, and remove edges between two vertices from different clusters. The number of adversary steps is unbounded.

Simulated annealing, first used by Kirkpatrick et al. [77], is a generic method for solving optimization problems. Jerrum and Sorkin [72] gave an algorithm for the minimum bisection problem using constant-temperature simulated annealing (also called the Metropolis algorithm). The algorithm of Jerrum and Sorkin is as follows: It starts from an initial random bisection and then performs a series of iterations that change the bisection. At each iteration, two vertices are chosen randomly, one from each side of the current bisection, and the two vertices are swapped with probability that depends on the change of the cut size that is caused by the swap. Jerrum and Sorkin showed that if $\Delta \geq n^{-1/6+\epsilon}$ for some $\epsilon > 0$, then the algorithm solves the minimum bisection problem with high probability after $O(n^2)$ iterations. The proof is based on the analysis of the imbalance of the cut $(S, V - S)$: For two disjoint sets L and R, the *L, R-imbalance* of a cluster V_i is the number of vertices of V_i in L minus the number of vertices of V_i in R. The *imbalance of* L, R is the maximum value amongst the L, R-imbalance of the clusters. Jerrum and Sorkin showed that the maximum imbalance of $S, V - S$ behaves like a random walk with a bias for increase, and the bias grows over time. A special case of the algorithm above is when the probability for a swap is 1 if the swap decreases the cut size, and 0 if the swap increases the cut size. This version is, in fact, a hill climbing algorithm. Jules [73] showed that the hill climbing algorithm solves the minimum bisection problem with high probability if $\Delta = \Omega(n)$.

Condon and Karp [39] gave an algorithm that solves the minimum bisection problem with high probability when $\Delta \geq n^{-1/2+\epsilon}$ for some $\epsilon > 0$. The algorithm is based on successive augmentation of disjoint sets L and R: At each step, two vertices are chosen randomly from $V - (L \cup R)$ and then one vertex is added to L and the other to R, where the choice of which vertex is added to which set is made according to the number of edges between the two vertices and the sets L and R. The set L is subsequently used to partition the input graph into two parts. The algorithm is described in more details later in the introduction as it

is related to our approach. The proof of performance of the algorithm is based on analyzing the behavior of the L, R-imbalance. The algorithm of Condon and Karp also applies to the case when there is a constant number of equal sized underlying clusters.

Carson and Impagliazzo [31] gave an algorithm similar to that of Condon and Karp. The main difference is that in the algorithm of Carson and Impagliazzo, at each step only one vertex is chosen. This algorithm solves the minimum bisection problem with high probability when $\Delta = \omega(\sqrt{p}n^{-1/2}\log n)$. Carson and Impagliazzo also showed that the hill climbing algorithm solves the bisection problem with high probability when $\Delta = \Omega(n^{-1/4}\log^4 n)$ and $p = \Omega(1)$, and a modified version of the hill climbing algorithm solves the problem when $\Delta = \Omega(n^{-1/2}\log^4 n)$ and $p = \Omega(1)$. This improves both the result of Jerrum and Sorkin and the result of Jules.

A related result was given by Alon et al. [4] for the hidden clique problem. The input to that problem is a graph that is built by the following process: Each two vertices in the graph are connected by an edge with probability 1/2, and then k vertices are chosen and a clique is built over these vertices. The goal is to find the clique. Note that this problem is a special case of the clustering problem when $p = 1$, $r = 1/2$, $|V_1| = k$, and $|V_i| = 1$ for all $i > 1$. Alon et al. gave a polynomial-time algorithm that solves the problem with high probability when $k = \Omega(\sqrt{n})$.

4.1.1 New results

We deal with the clustering problem under the random cluster graph model. Unlike the previous results, we use a slightly different formulation to the problem: The *clustering problem* is to find, in a given random cluster graph, the underlying clusters. This problem is asymptotically equivalent (under some restrictions on the range of the parameters Δ and k) to the cluster editing problem or the minimum bisection problem.

All previous studies addressed the clustering problem when the number of clusters is constant and most studied the case of equal sized clusters. Our algorithms relax both of these assumptions simultaneously, and at the same time achieve better running time.

Let us be more precise. We give an $O(mn^2/\log n)$-time algorithm that solves the clustering problem with high probability, assuming that $k = \Omega(\Delta^{-1-\epsilon}\sqrt{n\log n})$ for some constant $\epsilon > 0$. The running time improves if k is asymptotically larger than its lower bound. Furthermore, if the sizes of the clusters are equal (or almost equal), we give an $O((m/\log n + 1)n^2)$-time algorithm that requires only $k = \Omega(\Delta^{-1}\sqrt{n}\log n)$ (here too, the running time improves if k is asymptotically larger).

A comparison between previous results and our results is given in Table 4.1. The algorithms presented here have a wider range of provable performance than

each of the previous algorithms, and are also faster when restricted to the same parameter range. For example, for unequal sized clusters, the algorithm of Ben-Dor et al. [17] requires $k = \Omega(n)$ and $\Delta = \Omega(1)$, while our algorithm can handle instances with $k = \Theta(\sqrt{n \log n})$, and instances with $\Delta = \Theta(n^{-1/2+\epsilon})$. Furthermore, under the requirements of Ben-Dor et al., the running time of our algorithm is $O(n \log n)$. For the case of two equal sized clusters, our algorithm handles almost the same range of Δ as the algorithm of Boppana [28], but our algorithm is faster, and is also more general since it handles as many as $m = \Theta(\sqrt{n}/\log n)$ clusters.

We note that Table 4.1 cites results as given in the papers, even though in several cases better results can be obtained by improving the analysis or by making small modifications to the algorithms. For example, the algorithm of Condon and Karp can be extended to the case when the number of clusters is non-constant. Also, the running time of the algorithm of Ben-Dor et al. can be improved to $O(n \log n)$ [131]. The results above were presented in [119] and published in [121].

After obtaining our results, McSherry [97] gave a new polynomial algorithm for the clustering problem that requires that $k = \Omega(\Delta^{-1}\sqrt{n \log n} + \Delta^{-2/3} n^{2/3})$. For the case of equal sized clusters, our algorithm has a wider range of k when $\Delta = w(n^{-1/2} \log^3 n)$, while McSherry's algorithm has a wider range of k when $\Delta = o(n^{-1/2} \log^3 n)$ (note that McSherry's algorithm has a lower bound of $\Omega(n^{-1/2}\sqrt{\log n})$ on Δ). For the case of unequal clusters, our algorithm has a wider range of k when $\Delta \geq n^{-1/2+\epsilon}$ for some $\epsilon > 0$, while McSherry's algorithm has a wider range of k when Δ is smaller.

We also study a variant of the cluster editing problem in which the number of clusters is known in advance: The *m-cluster editing problem* requires finding, for a given graph, the nearest clique graph with exactly m connected components. We show that for every $m \geq 2$, the m-cluster editing problem is NP-hard. This result was published in [124].

4.1.2 Outline of our approach

In the rest of this section we outline the basic techniques that we use and their differences from previous studies. Our approach is similar to that of the algorithm of Condon and Karp [39] (we will refer to it as the CK algorithm). Recall that for a vertex v in a graph G, and a set S of vertices of G, $d_S(v)$ denotes the number of neighbors of v in S. We will first give a detailed description of the CK algorithm (in a slightly simplified version, for clarity):

1. Begin with empty sets L and R. Choose $n/4$ pairs of vertices (without repetitions). Repeatedly, for each pair v, w, compute the expressions $d_L(v) - d_R(v)$ and $d_L(w) - d_R(w)$. Add the vertex whose expression is larger to L, and add the other vertex to R.

Paper	Requirements		Complexity
	k	Δ	
Ben-Dor et al. [17]	$\Omega(n)$	$\Omega(1)$	$O(n^2 \log^{O(1)} n)$
This paper	$\Omega(\Delta^{-1-\epsilon}\sqrt{n \log n})$	$\Omega(n^{-1/2+\epsilon})^*$	$O(mn^2/\log n)$

(a) General case.

Paper	Requirements		Complexity
	m	Δ	
Dyer & Frieze [48]	2	$\Omega(n^{-1/4} \log^{1/4} n)$	$O(n^2)$
Boppana [28]	2	$\Omega(\sqrt{p}n^{-1/2}\sqrt{\log n})$	$n^{O(1)}$
Jerrum & Sorkin [72]	2	$\Omega(n^{-1/6+\epsilon})$	$O(n^3)$
Jules [73]	2	$\Omega(1)$	$O(n^3)$
Condon & Karp [39]	$O(1)$	$\Omega(n^{-1/2+\epsilon})$	$O(n^2)$
Carson & Impagliazzo [31]	2	$\omega(\sqrt{p}n^{-1/2} \log n)$	$O(n^2)$
Feige & Kilian [54]	2	$\Omega(\sqrt{p}n^{-1/2}\sqrt{\log n})$	$n^{O(1)}$
This paper	$O(\sqrt{n}/\log n)^*$	$\Omega(mn^{-1/2} \log n)$	$O((\frac{m}{\log n} + 1)n^2)$

(b) Equal sized clusters.

Table 4.1: Results on the clustering problem (sorted in chronological order). For the comparison, the lower bound $k = \Omega(\Delta^{-1}\sqrt{n} \log n)$ of our algorithm for equal sized clusters was translated to a lower bound on Δ using the fact that $k \leq n/m$. Note that all previous papers assume $m = 2$ or $m = O(1)$ (the requirement $k = \Omega(n)$ in [17] implies that $m = O(1)$), and all except [17] assume equal sized clusters. For the values that are marked by $*$, no implicit requirement is made, and the requirement is implied by the bound on the other parameter.

2. Sort all the vertices in decreasing order according to their $d_L(\cdot)$ values, and find the largest gap in the sorted sequence. If this gap is small, output V and stop. Otherwise, put the vertices before the gap into a set L', and the vertices after the gap into a set R'.

3. Recursively solve the problem on the subgraph induced by L' and on the subgraph induced by R'.

Let $l_1, \ldots, l_m$ denote the L, R-imbalances of the cluster $V_1, \ldots, V_m$ at some point of the algorithm, and suppose w.l.o.g. that $l_1 > l_2 \geq \cdots \geq l_m$. Condon and Karp showed that during the first step of the algorithm, the imbalance of L, R behaves like a random walk with growing bias for increase. The general idea

of the proof is that the imbalance changes only when handling a pair v, w in which one vertex is from V_1 and the other vertex is not. Furthermore, for such a pair, it is more likely that the vertex from V_1 will be added to L, and therefore the imbalance will increase. More precisely, given some vertices $v, w \notin L \cup R$, the random variable $X = (d_L(v) - d_R(v)) - (d_L(w) - d_R(w))$ is a sum of $4|L|$ independent random variables ($2|L|$ variables are Bernoulli variables and the rest are the negated Bernoulli variables). The expectation of X depends on the L, R-imbalances of the clusters that contain v and w: If $v \in V_i$ and $w \in V_j$ then the expectation of X is $\Delta|L| \cdot (l_i - l_j)$. Using estimates on sums of independent random variables (Esseen's inequality), it is shown that the probability that v will be added to L (which is equal to the probability that $X > 0$ plus half of the probability that $X = 0$) is $1/2 + \Omega(\min(\frac{(l_i - l_j)\Delta}{\sqrt{|L|}}, 1))$ for $i > j$. It follows that with high probability (w.h.p.), at the end of Step 1, the imbalance of L, R is $\Theta(n)$.

We now consider the second step of the algorithm, which we will call the *splitting step*, and suppose that there are at least two clusters. Using Chernoff bounds, w.h.p., for every vertex v, the deviation of $d_L(v)$ from its expectation is at most $\alpha = n^{1/2+\epsilon/2}$. The expectation of $d_L(v)$ depends on the imbalance of the cluster that contains v: If $v \in V_i$ then the expectation of $d_L(v)$ is roughly $A_i = \Delta(\frac{n}{4m} + l_i/2) + r\frac{n}{4}$. There is an index i such that $l_i - l_{i+1} \geq (l_1 - l_m)/(m-1) = \Omega(n/m)$ (the last equality follows from the fact that $l_1 = \Theta(n)$ and $l_m \leq 0$). Therefore, $A_i - A_{i+1} = \Omega(\Delta n/m) = \Omega(n^{1/2+\epsilon})$, and the largest gap in the sorted sequence of $d_L(\cdot)$ values is at least $(A_i - A_{i+1}) - 2\alpha = \Omega(n^{1/2+\epsilon})$. It follows that w.h.p., every cluster V_i, is contained either in L' or in R'.

We now give a short description of our algorithms. The main difference between our algorithm and the CK algorithm is the distribution of the l_i-s. The success of the splitting step in the CK algorithm depends on having a large gap in the sequence of the l_i-s. As shown previously, this gap is $\Omega(n/m)$, and this bound is true also in case m is not a constant (this case was not analyzed by Condon and Karp). Furthermore, this lower bound is likely to be tight: Condon and Karp conjectured (and showed in simulations) that w.h.p., at the end of Step 1, the l_i-s are distributed uniformly between $l = (1 - 1/m)\frac{n}{4m}$ and $-l$, namely $l_1, \ldots, l_m \approx l, (1 - \frac{2}{m})l, (1 - \frac{4}{m})l, \ldots, -(1 - \frac{2}{m})l, -l$. This implies that the largest gap in the l_i-s sequence is $\Theta(n/m)$. In contrast, our algorithms were designed in order to keep the values of $l_2, \ldots, l_m$ much smaller than l_1, (namely, $l_2, \ldots, l_m \leq (1 - \delta)l_1$ for some constant $\delta > 0$), and use this fact to achieve a $\Theta(n)$ gap. This allows us to give better algorithms for the case where m is non-constant. Furthermore, since it is known that there is a large gap between l_1 and l_2, we can use a modified splitting step: Instead of finding the maximum gap in the sorted $d_L(\cdot)$ values, we find the first gap which is large enough. The result is that w.h.p. the set L' will contain only one cluster (V_1).

In order to build the sets L and R with the desired imbalances, we build a series of pairs of sets L_i, R_i, where the pair L_i, R_i is built from L_{i-1}, R_{i-1}

using a process similar to the CK algorithm: We choose disjoint pairs of vertices, and for each pair v, w we compute the expressions $d_{L_{i-1}}(v) - d_{R_{i-1}}(v)$ and $d_{L_{i-1}}(w) - d_{R_{i-1}}(w)$. The vertex whose expression is larger is added to L_i, and the other vertex to R_i. (The pair L_0, R_0 is built using a similar process). Note that unlike the CK algorithm, the sets L_{i-1}, R_{i-1}, L_i, and R_i are disjoint, so we have independence between the pairs of vertices, which simplifies the analysis of the algorithm. We show that the imbalances of the pairs L_i, R_i increases at each iteration.

We note that the algorithms presented here might not be directly applicable in real-life applications since our analysis holds only for large n, and since real life data might not behave like the random cluster graph model. However, the ideas we use might be useful for building good heuristic algorithms. For example, Ben-Dor et al. gave a heuristic algorithm (called CAST) based on their theoretical algorithm for the random cluster graph model, and demonstrated that the heuristic performs well in real clustering applications [17].

The rest of this chapter is organized as follows: Section 4.2 contains notation and estimates on the sum of independent random variables. Section 4.3 and Section 4.4 contain the basic ideas for the algorithms. The algorithm for the case of almost equal sized clusters is given in Section 4.5, and the algorithm for the general case is given in Section 4.6. In Section 4.7 we prove the hardness of the m-cluster editing problem. We provide at the end of the chapter a list of the main symbols used in the chapter and their meaning.

4.2 Preliminaries

We use the O-notations O, Ω and o with their usual meaning. However, we write $f \leq O(g)$ instead of the more common $f = O(g)$ to emphasize that an upper bound on f is given. We also use $f \leq \hat{o}(g)$ to denote that $f(n) \leq c \cdot g(n)$ for some constant c that can be made arbitrarily small, depending on other constants (This implies in particular that $f \leq O(g)$). For example, if we have the requirement $k \geq \Omega(\sqrt{n})$, then we can write that $m \leq n/k \leq \hat{o}(\sqrt{n})$ since the hidden constant in the latter inequality can be made arbitrarily small by increasing the hidden constant in the former inequality.

For proving the correctness of our algorithms we use the following theorems which give estimates on the sum of independent random variables. The following theorem is derived from Chernoff [34].

Theorem 4.2.1. *Let $X_1, \ldots, X_n$ be independent Bernoulli random variables, and let $X = \sum_{i=1}^{n} X_i$. Then, $\mathrm{P}\left[|X - \mathrm{E}[X]| \geq a\sqrt{3\mathrm{E}[X]}\right] \leq 2e^{-a^2}$ for every $a \geq 0$.*

Theorem 4.2.2 (Esseen's Inequality). *Let $X_1, \ldots, X_n$ be independent random variables such that $\mathrm{E}[X_i] = 0$ and $\mathrm{E}[|X_i|^3] < \infty$, for $i = 1, \ldots, n$. Let*

$B_n = \sum_{i=1}^n \mathrm{E}[X_i^2]$ and $L_n = B_n^{-3/2} \sum_{i=1}^n \mathrm{E}[|X_i|^3]$. Then $|\mathrm{P}\left[B_n^{-1/2} \sum_{i=1}^n X_i < x\right] - \Phi(x)| \leq AL_n$ where A is an absolute constant and $\Phi(x)$ denotes the normal $(0,1)$ cumulative distribution function.

For a proof of Esseen's inequality see [104, p. 111]. Esseen's inequality holds for $A = 0.8$ [130].

Corollary 4.2.3. *Let $X_1, \ldots, X_n$ be independent random variables such that $\mathrm{E}[|X_i|^3] < \infty$, for $i = 1, \ldots, n$. Let $B_n = \sum_{i=1}^n \mathrm{Var}(X_i)$, $L_n = B_n^{-3/2} \sum_{i=1}^n \mathrm{E}[|X_i - \mathrm{E}[X_i]|^3]$, and $X = \sum_{i=1}^n X_i$. Then $|\mathrm{P}[X > 0] - \Phi(\mathrm{E}[X]/\sqrt{B_n})| \leq 0.8L_n$ and $|\mathrm{P}[X < 0] - (1 - \Phi(\mathrm{E}[X]/\sqrt{B_n}))| \leq 0.8L_n$.*

Proof. For the first part of the corollary, define $Y_i = \mathrm{E}[X_i] - X_i$ and apply Theorem 4.2.2 on $Y_1, \ldots, Y_n$ with $x = \mathrm{E}[X]/\sqrt{B_n}$. The second part is proved by taking $Y_i = X_i - \mathrm{E}[X_i]$ and $x = -\mathrm{E}[X]/\sqrt{B_n}$. ■

We also use the following lemma.

Lemma 4.2.4. *Let A be a set with n elements, and B be a subset of A with k elements. Let S' be a random subset of A, and let S be a random subset of $A - S'$ of size s. Then $\mathrm{P}\left[||B \cap S| - \frac{k}{n}s| \geq a\sqrt{3(k/n)s}\right] \leq 6\sqrt{k}e^{-a^2}$ for every $a \geq 0$.*

Proof. Let $X_1, \ldots, X_n$ be independent random variables which have Bernoulli distribution with parameter k/n. Let $X = \sum_{i=|S'|+1}^{|S'|+s} X_i$ and $Y = \sum_{i=1}^n X_i$. Fix some $a \geq 0$, and let Z denote the event that $|X - \frac{k}{n}s| \geq a\sqrt{3(k/n)s}$. We have that

$$\mathrm{P}\left[||B \cap S| - \frac{k}{n}s| \geq a\sqrt{3\frac{k}{n}s}\right] = \mathrm{P}[Z|Y = k] = \frac{\mathrm{P}[Z \cap (Y = k)]}{\mathrm{P}[Y = k]} \leq \frac{\mathrm{P}[Z]}{\mathrm{P}[Y = k]}.$$

By Theorem 4.2.1, $\mathrm{P}[Z] \leq 2e^{-a^2}$. Furthermore,

$$\mathrm{P}[Y = k] = \binom{n}{k}\left(\frac{k}{n}\right)^k\left(1 - \frac{k}{n}\right)^{n-k} = \frac{n!}{k!(n-k)!} \cdot \frac{k^k(n-k)^{n-k}}{n^n}.$$

Stirling formula states that $\sqrt{2\pi l}\left(\frac{l}{e}\right)^l \leq l! \leq 1.09\sqrt{2\pi l}\left(\frac{l}{e}\right)^l$. Hence,

$$\begin{aligned}\mathrm{P}[Y = k] &\geq \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{1.09\sqrt{2\pi k}\left(\frac{k}{e}\right)^k \cdot 1.09\sqrt{2\pi(n-k)}\left(\frac{n-k}{e}\right)^{n-k}} \cdot \frac{k^k(n-k)^{n-k}}{n^n} \\ &= \frac{1}{1.09^2\sqrt{2\pi}}\sqrt{\frac{n}{k(n-k)}} \geq \frac{1}{3\sqrt{k}}.\end{aligned}$$

The lemma follows from the above bounds. ■

In the following, we say that an event happens with high probability (w.h.p.) if its probability is $1 - n^{-\Omega(1)}$. Using this terminology we obtain the following corollaries of Theorem 4.2.1 and Lemma 4.2.4, respectively.

Corollary 4.2.5. *Under the conditions of Theorem 4.2.1, w.h.p.,* $|X - \mathrm{E}[X]| \leq O(\sqrt{\mathrm{E}[X] \log n})$.

Corollary 4.2.6. *Under the conditions of Lemma 4.2.4, w.h.p.,* $||B \cap S| - \frac{k}{n}s| \leq O(\sqrt{(k/n)s \log n})$.

We also obtain the following corollary:

Corollary 4.2.7. *Under the conditions of Lemma 4.2.4, and if additionally* $S' = \phi$ *and* $s \leq n/2$*, then w.h.p.,* $|\frac{|B-S|}{n-s} - \frac{k}{n}| \leq O(\sqrt{(k/n)n^{-1} \log n})$.

Proof. Clearly,

$$\left|\frac{|B-S|}{n-s} - \frac{k}{n}\right| = \frac{\left||B-S| - (1-\frac{s}{n})k\right|}{n-s} = \frac{\left||B-S| - k + \frac{s}{n}k\right|}{n-s} = \frac{\left||B \cap S| - \frac{s}{n}k\right|}{n-s},$$

and by Corollary 4.2.6, w.h.p.,

$$\left|\frac{|B-S|}{n-k} - \frac{s}{n}\right| \leq O(\frac{1}{n-s}\sqrt{(k/n)s \log n}) \leq O(\sqrt{(k/n)n^{-1} \log n}). \qquad \blacksquare$$

4.3 The basic algorithm

In this section we give a top-level description of our algorithms.

Let $G = (V, E)$ be a random cluster graph. Denote $A_i = |V_i|$, $a_i = |V_i|/n$, and $\Gamma = \max_i |\frac{|V_i|}{n} - \frac{1}{m}|$. A set $S \subseteq V$ is called a *subcluster* if $S \subseteq V_i$ for some cluster V_i. An induced subgraph G_S is called a *cluster collection* if for all i, either $V_i \subseteq S$ or $V_i \subseteq V - S$. Suppose we have a procedure Find(G, V') that receives a random cluster graph $G = (V, E)$ and a set $V' \subseteq V$, and returns a subcluster of size $\Omega(\log n/\Delta^2)$, by only considering vertices and edges in $G_{V'}$. Then, we use the following algorithm for solving the clustering problem: Repeatedly, find a subcluster S in the graph, find the cluster that contains S (using a procedure similar to the splitting step of the CK algorithm), and remove the cluster from the graph. Formally, the algorithm is as follows:

Solve(G)

1. Randomly partition the vertices of G into equal sized sets W_1, W_2, and W_3.
2. $S_0 \leftarrow$ Find(G, W_1).
3. Let S be a random subset of S_0 of size $s = \Theta(\log n/\Delta^2)$. Let $v_1, \ldots, v_{n/3}$ be an ordering of W_2 such that $d_S(v_1) \geq d_S(v_2) \geq \cdots \geq d_S(v_{n/3})$. Let $D = \Theta(\sqrt{s \log n})$. If $\max_j\{d_S(v_j) - d_S(v_{j+1})\} < D$, output V and stop.

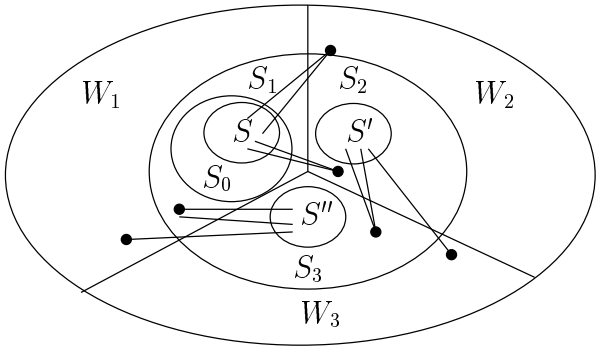

Figure 4.1: The sets created by algorithm Solve. The edges represented are counted by the algorithm. For example, the set S_2 is built by looking at the edges between W_2 and S.

4. Let j be an index such that $d_S(v_j) - d_S(v_{j+1})$ is maximum, and let $S_2 = \{v_1, \ldots, v_j\}$.
5. Let S' be a random subset of S_2 of size s. Let $w_1, \ldots, w_{n/3}$ be an ordering of W_3 such that $d_{S'}(w_1) \geq \cdots \geq d_{S'}(w_{n/3})$. Let j' be an index such that $d_{S'}(w_{j'}) - d_{S'}(w_{j'+1})$ is maximum, and let $S_3 = \{w_1, \ldots, w_{j'}\}$.
6. Let S'' be a random subset of S_3 of size s. Let $u_1, \ldots, u_{n/3}$ be an ordering of W_1 such that $d_{S''}(u_1) \geq \cdots \geq d_{S''}(u_{n/3})$. Let j'' be an index such that $d_{S''}(u_{j''}) - d_{S''}(u_{j''+1})$ is maximum, and let $S_1 = \{u_1, \ldots, u_{j''}\}$.
7. Output $S_1 \cup S_2 \cup S_3$ and delete the vertices in $S_1 \cup S_2 \cup S_3$ from G.
8. Goto 1.

See also Figure 4.1. Here and in the following, n is the number of vertices in the current subgraph of the input graph G. For simplicity, we assume that n is divisible by 3. The description of the algorithm above (and the rest of the algorithms in this chapter) is slightly incomplete since the hidden constants are not specified. We chose not to write the constants explicitly as it would have affected the readability of the proofs.

Note that the partition of the graph into three parts is necessary in order to avoid dependencies: This partition guarantees, for example, that on step 3 of the algorithm, for every $u \in S$ and $v \in W_2$, the existence of the an edge between u and v is independent of the choice of the set S. To see this, suppose that the edges of the graph are not chosen before the algorithm begins, but instead, the existence of an edge is determined randomly when the algorithm first checks the existence of that edge. Since procedure Find only considers edges with both endpoints in W_1, the existence of the edges between W_1 and W_2 has not been determined when the algorithm enters step 3, so for each $u \in W_1$ and $v \in W_2$, the event that there is an edge between u and v is independent of the choice of S.

In fact, this would remain true, even if the set S is chosen by an adversary that can only look at edges in G_{W_1}. Note that partitioning the graph into two sets and using the same arguments as above would not be enough, since some edges between the two parts would be considered twice by the corresponding algorithm. However, it is possible to show that the algorithm is correct in this case, using additional arguments.

Let $T(G)$ denote the running time of procedure Find on an input graph G.

Lemma 4.3.1. *If procedure* Find *returns w.h.p. a subcluster of size* $\Omega(\log n/\Delta^2)$*, then w.h.p., algorithm* Solve *solves the clustering problem. W.h.p., the running time of algorithm* Solve *is* $O(mT(G) + mn\log n/\Delta^2)$.

Proof. We shall prove that the failure rate in one iterations is n^{-c} for some constant c. The probability of failure of the algorithm is at most m times the probability of failure in one iteration, and since $m \leq n$ and c can be chosen sufficiently large, we obtain that the overall failure probability is $n^{-\Omega(1)}$. We shall show that if G consists of one cluster, then w.h.p. the algorithm stops at Step 3, and otherwise, if $S_0 \subseteq V_i$ then w.h.p., $S_1 \cup S_2 \cup S_3 = V_i$.

W.l.o.g. we assume that $S_0 \subseteq V_1$. Since the edges between W_1 and W_2 are independent of the choice of S, by Corollary 4.2.5 we have that for a vertex $v \in W_2$, the probability that $|d_S(v) - \mathrm{E}[d_S(v)]| \geq \frac{1}{2}D$ is $n^{-c'}$ for a constant c' that depends on D. The probability that this happens for at least one vertex in W_2 is therefore at most $\frac{n}{3} \cdot n^{-c'}$. Thus, by choosing the constant in D in way that makes c' sufficiently large, we obtain that w.h.p., $|d_S(v) - \mathrm{E}[d_S(v)]| < \frac{1}{2}D$ for all $v \in W_2$. For $v \in V_1$ we have that $\mathrm{E}[d_S(v)] = sp$, and for $v \notin V_1$ we have $\mathrm{E}[d_S(v)] = sr$. Hence, if G consists of one cluster, then w.h.p. $|d_S(v) - d_S(v')| < D$ for all $v, v' \in W_2$, and therefore, algorithm Solve stops at Step 3. Otherwise, for two vertices $v, v' \in W_2$, if either $v, v' \in V_1$ or $v, v' \notin V_1$ then w.h.p. $|d_S(v) - d_S(v')| < D$, and if $v \in V_1$ and $v' \notin V_1$ then w.h.p. $d_S(v) - d_S(v') > s\Delta - D \geq D$ where the last inequality is achieved by choosing appropriate constants for s, D so $s\Delta \geq 2D$. It follows that algorithm Solve does not stop at Step 3, and furthermore, $S_2 = V_1 \cap W_2$. Using similar arguments we show that w.h.p. $S_3 = V_1 \cap W_3$ and $S_1 = V_1 \cap W_1$.

From the correctness of the algorithm, we have that w.h.p. algorithm Solve performs exactly m iterations. The time complexity of a single iteration is $O(T(G) + n\log n/\Delta^2)$, where the second term is the time for computing the degrees at steps 3, 5, and 6. (Sorting the degrees can be done in $O(n)$ time by bin sorting.) ■

Note that algorithm Solve requires the knowledge of Δ. However, if instead of Δ we use some known lower bound Δ' on Δ, then the algorithm remains correct, and the running time becomes $O(mT(G) + mn\log n/\Delta'^2)$.

If the value of $mT(G) + mn\log n/\Delta^2$ is known (or if an upper bound on this value is known), then the algorithm can be modified to to stop after $O(mT(G) +$

$mn \log n/\Delta^2$) steps with a "Failure" output. This change does not affect the correctness of the algorithm, and the running time is always $O(mT(G)+mn \log n/\Delta^2)$.

4.3.1 Handling large clusters and close edge probabilities

In order to build the procedure Find, we will need the following additional requirements on the input graph:

(R1) $k \geq \Omega(\sqrt{n} \log^{\alpha} n/\Delta^{\beta})$ for some $\alpha, \beta > 0$.

(R2) $p, r \in [\frac{1}{4}, \frac{3}{4}]$.

(R3) $\max_i a_i < \frac{2}{3}$.

While requirement of the type (R1) is understandable, we would like to get rid of (R2) and (R3). We will show how to deal with these requirements in the rest of this section.

To eliminate (R2), we transform the input graph to a graph that always satisfies the requirement using the following algorithm:

Solve$_2(G)$:

1. Build a graph G' on the vertex set of G in the following way: For every pair of vertices u, v, add the edge (u, v) to G' with probability $\frac{3}{4}$ if (u, v) is an edge in G, and with probability $\frac{1}{4}$ otherwise.
2. Run Solve(G').

Lemma 4.3.2. *If procedure* Find *returns w.h.p. a subcluster of size* $\Omega(\log n/\Delta^2)$ *on graphs that satisfy (R1)–(R3), then w.h.p., algorithm* Solve$_2$ *solves the clustering problem on graphs that satisfy (R1) and (R3). The running time of algorithm* Solve$_2$ *is* $O(mT(G') + mn \log n/\Delta^2)$.

Proof. Clearly, G' is a random cluster graph on the clusters $V_1, \ldots, V_m$ with edge probabilities $p' = \frac{3}{4}p + \frac{1}{4}(1-p) = \frac{1}{2}p + \frac{1}{4}$ and $r' = \frac{1}{2}r + \frac{1}{4}$. G' satisfies (R1) and furthermore, every cluster collection G'' of G' satisfies (R1) as the size of the smallest cluster in G'' is greater or equal to the size of the smallest cluster in G. As $p', r' \in [\frac{1}{4}, \frac{3}{4}]$ and $p' - r' = \Theta(\Delta)$, by Lemma 4.3.1 Solve(G') returns w.h.p. the correct partition. ∎

To eliminate (R3) we use a procedure that recursively partitions the input graph into cluster collections until subgraphs that satisfy (R3) are obtained. This is done by the following algorithm:

Solve$_3(G)$:

1. Let $S = \phi$. Independently add each vertex $v \in V$ to S with probability $q = \Theta(\log n/\Delta^2 n)$. If $|S| \geq 2qn$, output 'Failure' and stop.

2. Let $v_1, \ldots, v_n$ be an ordering of V such that $d_S(v_1) \geq d_S(v_2) \geq \cdots \geq d_S(v_n)$. Let $D = \Theta(\sqrt{qn \log n})$. If $\max_j\{d_S(v_j) - d_S(v_{j+1})\} < D$ then call $\text{Solve}_2(G)$ and stop.

3. Let j be an index such that $d_S(v_j) - d_S(v_{j+1})$ is maximum, and let $R = \{v_1, \ldots, v_j\}$. Call $\text{Solve}_3(G_R)$ and $\text{Solve}_3(G_{V-R})$.

For the following lemma we assume that T satisfies $\sum_{i=1}^{l} T(G_i) \leq T(G)$ for any partition of G into vertex disjoint cluster collections $G_1, \ldots, G_l$, and furthermore, for the graph G' that is built by $\text{Solve}_2(G)$, $T(G') \leq O(T(G))$. These assumptions are satisfied by the time complexity function of the procedure we give in Section 4.6.

Lemma 4.3.3. *If procedure* Find *returns w.h.p. a subcluster of size* $\Omega(\log n/\Delta^2)$ *on graphs that satisfy (R1)–(R3) then w.h.p., algorithm* Solve_3 *solves the clustering problem on graphs that satisfy (R1). The running time of algorithm* Solve_3 *is* $O(mT(G) + mn \log n/\Delta^2)$.

Proof. First, by Corollary 4.2.5, w.h.p. $|S| \leq qn + O(\sqrt{qn \log n}) \leq 2qn$, so algorithm Solve_3 does not stop at Step 1.

Consider the first iteration of the algorithm. Suppose that $a_1 \geq 2/3$, so $a_i \leq 1/3$ for all $i > 1$. By Corollary 4.2.5, w.h.p. $|d_S(v) - \text{E}[d_S(v)]| < \frac{1}{2}D$ for all $v \in V$. For $v \in V_i$ we have $\text{E}[d_S(v)] = (A_i - 1)qp + (n - A_i)qr = A_i q\Delta - qp + nqr$, so for $v \in V_1$ and $v' \in V_i$ (for $i \neq 1$) we have $\text{E}[d_S(v)] - \text{E}[d_S(v')] = (A_1 - A_i)q\Delta \geq \frac{1}{3}nq\Delta$. Therefore, w.h.p. $\min_{v \in V_1} d_S(v) - \max_{v \in V - V_1} d_S(v) > \frac{1}{3}nq\Delta - D \geq D$ where the inequality follows by choosing the constants in q and D appropriately.

Hence, if at Step 2, $\max_j\{d_S(v_j) - d_S(v_{j+1})\} < D$ then w.h.p. $\max_i a_i < 2/3$. Otherwise, w.h.p. G_R and G_{V-R} are cluster collections (as for two vertices v, v' from the same cluster we have w.h.p. $|d_S(v) - d_S(v')| < D$). Let $G_1, \ldots, G_r$ denote the graphs on which algorithm Solve_2 is called during the run of $\text{Solve}_3(G)$. From the above, we have that w.h.p. $G_1, \ldots, G_r$ are cluster collections, and each G_i satisfies (R3). Furthermore, each G_i satisfies (R1). Therefore, the correctness of Solve_3 follows from Lemma 4.3.2.

We now compute the running time of algorithm Solve_3. We have $r \leq m$ as each G_i is nonempty. Therefore, the total time not including the calls to Solve_2 is $O(r \cdot qn^2) = O(mn \log n/\Delta^2)$. Let $n(G)$ denote the number of vertices in a graph G. The total time of the calls to Solve_2 is $O(\sum_{i=1}^{r}(mT(G_i) + mn(G_i) \log n/\Delta^2)) = O(mT(G) + mn \log n/\Delta^2)$. ∎

4.4 The partition procedure

In the previous section, we presented the basic structure of our algorithms. However, we still need to show how to build procedure Find, namely, how to find a subcluster of size $\Omega(\log n/\Delta^2)$. In this section we give the main ideas, and we will use these ideas in sections 4.5 and 4.6 to explicitly build procedure Find.

One of the key elements we use is the notion of imbalance which was also used in [72] and [39]. We use here a slightly different definition for imbalance than the one used in these papers (and in the introduction): For two disjoint sets L, R of vertices of equal size, we define the *L, R-imbalance of V_i*, denoted $\mathrm{I}(V_i, L, R)$, by

$$\mathrm{I}(V_i, L, R) = \frac{|V_i \cap L| - |V_i \cap R|}{|L|}.$$

The *imbalance of L, R* is the maximum value amongst $\mathrm{I}(V_1, L, R), \ldots, \mathrm{I}(V_m, L, R)$, and the *secondary imbalance of L, R* is the second largest value (the secondary imbalance is equal to the imbalance if the maximum value appears more than once). We will show that given two sets L, R with "large" imbalance, and "small" secondary imbalance, it is possible to generate a large subcluster. Therefore, our goal is to generate the sets L, R with such properties.

Let T be a subset of V. Let $f\colon T \to \mathbb{N}$ be some function that depends on the edges of G. Since G is a random graph, we have that each $f(v)$ is a random variable. The function f is called a *cluster function* if $\{f(v)|v \in T\}$ are independent random variables, and for every $u, v \in T$ that belong to the same cluster, $f(u)$, $f(v)$ have the same distribution. For example, $f(v) = d_{V-T}(v)$ is a cluster function.

If the values of f for vertices of one cluster are always greater than the values of f for the vertices of the other clusters, then we can easily generate a subcluster by picking $\Theta(\log n/\Delta^2)$ vertices with largest f-value. However, we are able to give such a cluster function f only when Δ is large. For smaller value of Δ, we are able to give a cluster function f with the following property: The expectation of $f(v)$ for vertices of one cluster, say V_1, is larger than the expectation of $f(v)$ for vertices in the other clusters, and the variance of $f(v)$ is "small" for all v. We will use this property of f in the following procedure (here T is a random set of vertices):

Partition(G, T, f):

1. Begin with two empty sets L, R. Randomly partition the vertices of T into pairs. For each pair v, w, place the vertex with larger f-value into L and the other into R, breaking ties randomly.
2. Return L, R.

We note that procedure Partition is very similar to the algorithm of Condon and Karp [39], and the cluster function of Lemma 4.4.3 is also used in [39].

Let $p^{>}_{ij}(f) = \mathrm{P}\,[f(v) > f(w) | v \in V_i, w \in V_j]$ (we will write $p^{>}_{ij}$ if f is clear from the context), and let $p^{=}_{ij}(f)$ and $p^{<}_{ij}(f)$ be the conditional probabilities that $f(v) = f(w)$ and $f(v) < f(w)$, respectively. Let $c_i(f) = 2a_i \sum_{j \neq i} a_j (p^{>}_{ij}(f) - p^{<}_{ij}(f))$. Suppose that L, R are the output of Partition(G, T, f) and denote $b_i = \mathrm{I}(V_i, L, R)$.

Theorem 4.4.1. *If T is a random set of size $2l \leq n/2$, $k \geq \Omega(\log n)$, and $\max_i a_i \leq 2/3$, then w.h.p. $|b_i - c_i(f)| \leq O(\sqrt{a_i l^{-1} \log n}(1 + \sqrt{a_i m}))$ for all i.*

Proof. Since T is a random set, the process of choosing T and then partitioning T into pairs, is equivalent to the process of selecting vertices $u^1, \ldots, u^{2l}$ from V (one after another) and pairing u^{2t-1} and u^{2t} for each t. Denote $a^r_j = \mathrm{P}\,[u^r \in V_j]$. Clearly, $a^r_j = |V_j - \{u^1, \ldots, u^{r-1}\}|/|V - \{u^1, \ldots, u^{r-1}\}|$. By Corollary 4.2.7, we have w.h.p. that $|a_j - a^r_j| \leq O(\sqrt{a_j n^{-1} \log n})$ for all j and r. In the following we assume that these inequalities hold.

We denote by I^1_i (resp., I^2_i) the number of indices t for which $u^{2t-1} \in V_i$, $u^{2t} \notin V_i$, and u^{2t-1} (resp., u^{2t}) is inserted into L. Similarly, we denote by I^3_i (I^4_i) the number of indices t for which $u^{2t-1} \notin V_i$, $u^{2t} \in V_i$, and u^{2t} (u^{2t-1}) is inserted into L. Clearly, $b_i = (I^1_i + I^3_i - I^2_i - I^4_i)/l$. We will now give an estimate on I^1_i for some fixed i. Let X_t be an indicator variable for the event that $u^{2t-1} \in V_i$, $u^{2t} \notin V_i$, and u^{2t-1} is inserted into L. Let $p_j = p^{>}_{ij}(f) + \frac{1}{2} p^{=}_{ij}(f)$. Then,

$$\begin{aligned}
\mathrm{P}\,[X_t = 1] &= a_i^{2t-1} \sum_{j \neq i} a_j^{2t} p_j \\
&= a_i \sum_{j \neq i} a_j p_j + a_i \sum_{j \neq i} (a_j^{2t} - a_j) p_j + (a_i^{2t-1} - a_i) \sum_{j \neq i} a_j^{2t} p_j \\
&\leq a_i \sum_{j \neq i} a_j p_j + a_i \sum_{j=1}^{m} |a_j^{2t} - a_j| + |a_i^{2t-1} - a_i|
\end{aligned}$$

and using the above bounds on $|a_j - a^r_j|$ we obtain

$$\leq a_i \sum_{j \neq i} a_j p_j + O(a_i \sum_{j=1}^{m} \sqrt{a_j n^{-1} \log n} + \sqrt{a_i n^{-1} \log n}).$$

Since $\sum_{j=1}^m a_j = 1$, we have that the maximum value of $\sum_{j=1}^m \sqrt{a_j}$ is obtained when $a_1 = a_2 = \cdots = a_m = 1/m$, so $\sum_{j=1}^m \sqrt{a_j} \leq \sqrt{m}$. Therefore,

$$\mathrm{P}\,[X_t = 1] \leq a_i \sum_{j \neq i} a_j p_j + O(\sqrt{a_i n^{-1} \log n}(1 + \sqrt{a_i m})).$$

Define the quantity in the last equation by θ. As $\mathrm{P}\,[X_t = 1] \leq \theta$ for all t, we have that the random variable $I^1_i = \sum_{t=1}^{l} X_t$ is dominated by a random variable Y, where Y has binomial distribution with l experiments and success probability $\min(1, \theta)$. Since $k \geq \Omega(\log n)$, we have that $\sqrt{a_i n^{-1} \log n} = a_i \sqrt{(a_i n)^{-1} \log n} \leq$

$a_i\sqrt{k^{-1}\log n} \le O(a_i)$ and $a_i\sqrt{mn^{-1}\log n} \le a_i\sqrt{(n/k)n^{-1}\log n} \le O(a_i)$. Therefore, $\theta \le O(a_i)$, so $\mathrm{E}[Y] \le l\theta \le O(a_i l)$. Thus, by Corollary 4.2.5, we have w.h.p. that

$$\begin{aligned} Y &\le \mathrm{E}[Y] + O(\sqrt{\mathrm{E}[Y]\log n}) \le l\theta + O(\sqrt{a_i l \log n}) \\ &\le la_i \sum_{j\neq i} a_j p_j + O(\sqrt{a_i l \log n}(1+\sqrt{a_i m})). \end{aligned}$$

It follows that w.h.p. $I_i^1 \le la_i \sum_{j\neq i} a_j p_j + O(\sqrt{a_i l \log n}(1+\sqrt{a_i m}))$.

Similar arguments give a lower bound on I_i^1, and lower and upper bounds on I_i^2,I_i^3, and I_i^4, namely w.h.p.,

$$|I_i^l - la_i \sum_{j\neq i} a_j (p_{ij}^{>} + \frac{1}{2}p_{ij}^{=})| \le O(\sqrt{a_i l \log n}(1+\sqrt{a_i m})) \qquad \text{for } l = 1,3$$

$$|I_i^l - la_i \sum_{j\neq i} a_j (p_{ij}^{<} + \frac{1}{2}p_{ij}^{=})| \le O(\sqrt{a_i l \log n}(1+\sqrt{a_i m})) \qquad \text{for } l = 2,4$$

Combining the above inequalities gives the desired bound on b_i. ■

In the next sections, we use procedure Partition in a slightly different scenario than in Theorem 4.4.1: We have a random set S of vertices, and then the set T is randomly chosen from $V - S$. It is easy to verify that the result of Theorem 4.4.1 is also valid in that scenario.

We now give two cluster functions, which will be used later in our algorithms. The first function, given in Lemma 4.4.2, will be used for building initial sets L_0, R_0 with large imbalance. The second function, in Lemma 4.4.3, will be used to iteratively build a series of sets L_i, R_i with increasing imbalance. Again, in both cases, assume T is a random subset of V.

Lemma 4.4.2. *Let u be some vertex from $V_1 - T$, and define $f(v) = d_{\{u\}}(v)$ for all $v \in T$. Then $c_1(f) = 2a_1(1-a_1)\Delta$ and $c_i(f) = -2a_1a_i\Delta$ for all $i > 1$.*

Proof. Clearly,

$$\begin{aligned} p_{1j}^{>} &= \mathrm{P}\left[f(v) = 1, f(w) = 0 | v \in V_1, w \in V_j\right] \\ &= \mathrm{P}\left[(u,v) \in E, (u,w) \notin E | v \in V_1, w \in V_j\right] = p(1-r) \end{aligned}$$

and $p_{1j}^{<} = (1-p)r$, so

$$c_1(f) = 2a_1 \sum_{j\neq i} a_j(p(1-r) - (1-p)r) = 2a_1(1-a_1)\Delta.$$

For $i, j > 1$, $p_{ij}^{>} = p_{ij}^{<} = r(1-r)$, $p_{i1}^{>} = r(1-p)$, and $p_{i1}^{<} = (1-r)p$. Therefore, $c_i(f) = -2a_1a_i\Delta$. ■

Lemma 4.4.3. *Let L' and R' be two disjoint subsets of $V-T$ of size l each, and define $f(v) = d_{L'}(v) - d_{R'}(v)$ for all $v \in T$. Let $b_i = \mathrm{I}(V_i, L', R')$. Suppose that $b_1 \geq b_2 \geq \cdots \geq b_m$, $p, r \in [\frac{1}{4}, \frac{3}{4}]$, $k \geq \Omega(\sqrt{n \log n})$, and $|b_i - b_j| \leq \alpha/\Delta\sqrt{l}$ for all i, j where $\alpha \leq 1$. Furthermore, suppose that the process of building L', R' was independent of the edges between $L' \cup R'$ and T. Let $\beta = \Gamma + \sqrt{(1/m + \Gamma) l^{-1} \log n}$ and $\gamma = \sqrt{\frac{2}{\pi} / (\frac{1}{m} p(1-p) + (1-\frac{1}{m}) r(1-r))}$. Then, w.h.p.,*

1. $c_1(f) \geq \gamma\Delta\sqrt{l} a_1 (1 - O(\beta\Delta) - \frac{2}{9}\alpha^2)(b_1 - \sum_{j=1}^{m} a_j b_j) - 3\frac{a_1}{\sqrt{l}}$.
2. $c_i(f) \leq \gamma\Delta\sqrt{l} a_i \left(b_i - \sum_{j=1}^{m} a_j b_j + (O(\beta\Delta) + \frac{2}{9}\alpha^2)(b_1 - b_m) \right) + 3\frac{a_i}{\sqrt{l}}$ *for all* $i > 1$.
3. $c_i(f) \geq \gamma\Delta\sqrt{l} a_i \left(b_i - \sum_{j=1}^{m} a_j b_j - (O(\beta\Delta) + \frac{2}{9}\alpha^2)(b_1 - b_m) \right) - 3\frac{a_i}{\sqrt{l}}$ *for all* $i > 1$.

Note that here the notations b_i and l have different meaning that in Theorem 4.4.1.

Proof. (1) For $j = 1, \ldots, m$, denote $A'_j = |V_j \cap (L' \cup R')|$. We denote $L' = \{u_1, \ldots, u_l\}$ and $R' = \{u_{l+1}, \ldots u_{2l}\}$. We now estimate $p^{>}_{1j} - p^{<}_{1j}$ for some fixed j. Fix some $v \in V_1 \cap T$ and $w \in V_j \cap T$. We have that $f(v) - f(w) = \sum_{i=1}^{4l} X_i$ where $X_1, \ldots, X_{4l}$ are independent random variables with $X_i = d_{\{u_i\}}(v)$ for $i \in [1, l]$, $X_i = -d_{\{u_i\}}(v)$ for $i \in [l+1, 2l]$, $X_i = -d_{\{u_{i-2l}\}}(w)$ for $i \in [2l+1, 3l]$, and $X_i = d_{\{u_{i-2l}\}}(w)$ for $i \in [3l+1, 4l]$. Let $\mathrm{B}(q)$ denote the Bernoulli distribution with parameter q (i.e., a random variable with $\mathrm{B}(q)$ distribution has a value of 1 with probability q, and value of 0 otherwise). Out of the variable $X_1, \ldots, X_l$, there are $\frac{A'_1 + lb_1}{2}$ variables with $\mathrm{B}(p)$ distribution, and the rest of the variables have $\mathrm{B}(r)$ distribution. Out of $X_{l+1}, \ldots, X_{2l}$, $\frac{A'_1 - lb_1}{2}$ variables have $-\mathrm{B}(p)$ distribution, and the rest of the variables have $-\mathrm{B}(r)$ distribution. Similar claims hold for $X_{2l+1}, \ldots, X_{4l}$, and therefore,

$$\begin{aligned}
\mathrm{E}[\sum_{i=1}^{4l} X_i] &= p\frac{A'_1 + lb_1}{2} + r(l - \frac{A'_1 + lb_1}{2}) - p\frac{A'_1 - lb_1}{2} - r(l - \frac{A'_1 - lb_1}{2}) \\
&\quad - p\frac{A'_j + lb_j}{2} - r(l - \frac{A'_j + lb_j}{2}) + p\frac{A'_j - lb_j}{2} + r(l - \frac{A'_j - lb_j}{2}) \\
&= plb_1 - rlb_1 - plb_j + rlb_j = \Delta l(b_1 - b_j).
\end{aligned}$$

Let $B_{1j} = \sum_{i=1}^{4l} \mathrm{Var}(X_i)$. Then $B_{1j} = (A'_1 + A'_j)p(1-p) + (4l - A'_1 - A'_j)r(1-r)$, and since $p, r \in [\frac{1}{4}, \frac{3}{4}]$, it follows that $3/16 \leq p(1-p), r(1-r) \leq 1/4$, so $\frac{3}{4}l \leq B_{1j} \leq l$. Denote $h(x) = (1-x)^3 x + x^3(1-x)$. For $L_{1j} = B_{1j}^{-3/2} \sum_{i=1}^{4l} \mathrm{E}[|X_i - \mathrm{E}[X_i]|^3]$ we have that

$$L_{4l} = \frac{1}{B_{1j}^{3/2}}((A'_1 + A'_j)h(p) + (4l - A'_1 - A'_j)h(r)) \leq \frac{1}{(\frac{3}{4}l)^{3/2}} \cdot 4l \cdot \frac{1}{8} < \frac{0.77}{\sqrt{l}}.$$

Now, $p_{ij}^{>} = \mathrm{P}\left[\sum_{i=1}^{4l} X_i > 0\right]$ and $p_{ij}^{<} = \mathrm{P}\left[\sum_{i=1}^{4l} X_i < 0\right]$. By Corollary 4.2.3, $|p_{1j}^{>} - \Phi\left(\frac{\Delta l(b_1-b_j)}{\sqrt{B_{1j}}}\right)| \le 0.8L_{4l} < \frac{3}{4\sqrt{l}}$, and $|p_{1j}^{<} - (1 - \Phi\left(\frac{\Delta l(b_1-b_j)}{\sqrt{B_{1j}}}\right))| < \frac{3}{4\sqrt{l}}$. We conclude that $p_{1j}^{>} - p_{1j}^{<} \ge 2\Phi\left(\frac{\Delta l(b_1-b_j)}{\sqrt{B_{1j}}}\right) - 1 - \frac{3}{2\sqrt{l}}$.

For $x \in [0, a]$, where $a \le 1$, we have $\Phi(x) \ge \frac{1}{2} + \frac{1}{\sqrt{2\pi}}x - \frac{1}{6\sqrt{2\pi}}x^3 \ge \frac{1}{2} + \frac{1}{\sqrt{2\pi}}x(1 - \frac{a^2}{6})$. As $0 \le \frac{\Delta l(b_1-b_j)}{\sqrt{B_{1j}}} \le \frac{\Delta l(b_1-b_j)}{\sqrt{3l/4}} \le \frac{2}{\sqrt{3}}\alpha$, then

$$p_{1j}^{>} - p_{1j}^{<} \ge \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_1 - b_j)}{\sqrt{B_{1j}}}(1 - \frac{2}{9}\alpha^2) - \frac{3}{2\sqrt{l}}.$$

Let $B = 4l\frac{1}{m}p(1-p) + 4l(1-\frac{1}{m})r(1-r)$. By Corollary 4.2.6, w.h.p. $|A_i' - 2la_i| \le O(\sqrt{a_i l \log n}) \le O(\sqrt{(1/m + \Gamma)l \log n}) \le O(\beta l)$ for all i, so $|A_i' - \frac{2l}{m}| \le |A_i' - 2la_i| + 2l|a_i - \frac{1}{m}| \le O(\beta l)$. Therefore,

$$|B_{1j} - B| = |(A_1' + A_j' - \frac{4l}{m})(p(1-p) - r(1-r))| \le O(\beta l \cdot \Delta|1-p-r|) \le O(\beta l \Delta).$$

Since $B, B_{1j} \ge \frac{3}{4}l$, we conclude that $|B_{1j} - B| \le O(\beta\Delta B_{1j})$ and $|B_{1j} - B| \le O(\beta\Delta B)$. Thus, for $B \ge B_{1j}$,

$$\left|\frac{1}{\sqrt{B_{1j}}} - \frac{1}{\sqrt{B}}\right| = \frac{1}{\sqrt{B}}(\sqrt{\frac{B}{B_{1j}}} - 1) \le \frac{1}{\sqrt{B}}(\sqrt{1 + O(\beta\Delta)} - 1) \le O(\frac{\beta\Delta}{\sqrt{B}})$$

and for $B < B_{1j}$,

$$\left|\frac{1}{\sqrt{B_{1j}}} - \frac{1}{\sqrt{B}}\right| = \frac{1}{\sqrt{B_{1j}}}(\sqrt{\frac{B_{1j}}{B}} - 1) \le O(\frac{\beta\Delta}{\sqrt{B_{1j}}}) = O(\frac{\beta\Delta}{\sqrt{B}}).$$

Therefore,

$$\begin{aligned} p_{1j}^{>} - p_{1j}^{<} &\ge \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_1 - b_j)}{\sqrt{B}}(1 - O(\beta\Delta))(1 - \frac{2}{9}\alpha^2) - \frac{3}{2\sqrt{l}} \\ &\ge \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_1 - b_j)}{\sqrt{B}}(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2) - \frac{3}{2\sqrt{l}} \\ &= \frac{\gamma}{2}\Delta\sqrt{l}(b_1 - b_j)(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2) - \frac{3}{2\sqrt{l}}, \end{aligned}$$

and

$$\begin{aligned} c_1(f) &\ge 2a_1\sum_{j=1}^{m} a_j\frac{\gamma}{2}\Delta\sqrt{l}(b_1 - b_j)(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2) - a_1\sum_{j\ne 1} a_j\frac{3}{\sqrt{l}} \\ &\ge \gamma\Delta\sqrt{l}a_1(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2)(b_1 - \sum_{j=1}^{m} a_j b_j) - 3\frac{a_1}{\sqrt{l}}. \end{aligned}$$

(2) Like in case (1), we have that for every $j \neq i$, $p_{ij}^{>} - p_{ij}^{<} \leq 2\Phi(\frac{\Delta l(b_i - b_j)}{\sqrt{B_{ij}}}) - 1 + \frac{3}{2\sqrt{l}}$ where $B_{ij} = (A'_i + A'_j)p(1-p) + (4l - A'_i - A'_j)r(1-r)$. For $x \in [0,1]$, $\Phi(x) \leq \frac{1}{2} + \frac{1}{\sqrt{2\pi}}x$, so for $j > i$,

$$\begin{aligned} p_{ij}^{>} - p_{ij}^{<} &\leq \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_i - b_j)}{\sqrt{B_{ij}}} + \frac{3}{2\sqrt{l}} \\ &\leq \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_i - b_j)}{\sqrt{B}}(1 + O(\beta\Delta)) + \frac{3}{2\sqrt{l}} \\ &= \frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 + O(\beta\Delta)) + \frac{3}{2\sqrt{l}}. \end{aligned}$$

Furthermore, for $x \in [-a, 0]$ (where $a \leq 1$), $\Phi(x) \leq \frac{1}{2} + \frac{1}{\sqrt{2\pi}}x(1 - \frac{a^2}{6})$, so for $j < i$,

$$\begin{aligned} p_{ij}^{>} - p_{ij}^{<} &\leq \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_i - b_j)}{\sqrt{B_{ij}}}(1 - \frac{2}{9}\alpha^2) + \frac{3}{2\sqrt{l}} \\ &\leq \sqrt{\frac{2}{\pi}}\frac{\Delta l(b_i - b_j)}{\sqrt{B}}(1 - O(\beta\Delta))(1 - \frac{2}{9}\alpha^2) + \frac{3}{2\sqrt{l}} \\ &\leq \frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2) + \frac{3}{2\sqrt{l}}. \end{aligned}$$

Therefore,

$$\begin{aligned} c_i(f) &\leq 2a_i\left(\sum_{j=1}^{i-1} a_j\frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2)\right. \\ &\qquad \left. + \sum_{j=i+1}^{m} a_j\frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 + O(\beta\Delta)) + \sum_{j\neq i} a_j\frac{3}{2\sqrt{l}}\right) \\ &\leq \gamma\Delta\sqrt{l}a_i\left(b_i - \sum_{j=1}^{m} a_j b_j + (O(\beta\Delta) + \frac{2}{9}\alpha^2)\sum_{j=1}^{m} a_j|b_i - b_j|\right) + 3\frac{a_i}{\sqrt{l}} \\ &\leq \gamma\Delta\sqrt{l}a_i\left(b_i - \sum_{j=1}^{m} a_j b_j + (O(\beta\Delta) + \frac{2}{9}\alpha^2)(b_1 - b_m)\right) + 3\frac{a_i}{\sqrt{l}}. \end{aligned}$$

(3) We have, $p_{ij}^{>} - p_{ij}^{<} \geq \frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2) - \frac{2}{\sqrt{l}}$ for $j > i$, and

$p_{ij}^{>} - p_{ij}^{<} \geq \frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 + O(\beta\Delta)) - \frac{2}{\sqrt{l}}$ for $j < i$. Therefore,

$$\begin{aligned} c_i(f) &\geq 2a_i \left(\sum_{j=1}^{i-1} a_j \frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 + O(\beta\Delta)) \right. \\ &\qquad \left. + \sum_{j=i+1}^{m} a_j \frac{\gamma}{2}\Delta\sqrt{l}(b_i - b_j)(1 - O(\beta\Delta) - \frac{2}{9}\alpha^2) - \sum_{j\neq i} a_j \frac{3}{2\sqrt{l}} \right) \\ &\geq \gamma\Delta\sqrt{l}a_i \left(b_i - \sum_{j=1}^{m} a_j b_j - (O(\beta\Delta) + \frac{2}{9}\alpha^2) \sum_{j=1}^{m} a_j |b_i - b_j| \right) - 3\frac{a_i}{\sqrt{l}} \\ &\geq \gamma\Delta\sqrt{l}a_i \left(b_i - \sum_{j=1}^{m} a_j b_j - (O(\beta\Delta) + \frac{2}{9}\alpha^2)(b_1 - b_m) \right) - 3\frac{a_i}{\sqrt{l}}. \end{aligned}$$

■

4.5 Almost equal sized clusters

In this section we present an algorithm for finding a subcluster in the case where the sizes of all clusters are almost equal: The algorithm requires that $\Gamma \leq O(1/m \log n)$. We use algorithm Solve_2 with a procedure Find which is described below.

Procedure Find begins by building two initial sets L_0 and R_0 using procedure Partition and the cluster function of Lemma 4.4.2. Then a series of sets L_t, R_t is generated, where L_t and R_t are generated from L_{t-1} and R_{t-1} using procedure Partition and the cluster function of Lemma 4.4.3. The main idea is that the L_t, R_t-imbalance increases at each iteration. In order to stop the process, procedure Find also builds sets $\hat{L}_t$ and $\hat{R}_t$ which are built by the same process in which L_t and R_t are built. At each iteration, we sort the vertices of some random set of vertices S according to their f-values where $f(v) = d_{\hat{L}_t}(v) - d_{\hat{R}_t}(v)$. If there is a large gap in the sorted f-values, then the procedure stops, and outputs the vertices whose f-values are above the gap.

Formally, procedure Find is as follows:

Find(G, V')

1. Let $\hat{l} = \Theta(m^2\Delta^{-2}\log^2 n + \Delta^2 m^{-1}\log^5 n)$, $l = \Theta(\hat{l}/\log n)$, $l' = \Theta(\hat{l}/\log^2 n)$, $s = \Theta(m\Delta^{-2}\log n)$, and $D = \Theta(\sqrt{\hat{l}\log n})$. Let $l_t = l$ for $0 \leq t \leq 2$ and $l_t = l'$ for $t \geq 3$.
2. Randomly select disjoint sets of vertices S, $\hat{W}$, and W_0 from V' of sizes s, $2\hat{l}$, and $2l_0$, respectively.
3. Randomly select some unchosen vertex u from V'.
4. $\hat{L}_0, \hat{R}_0 \leftarrow \text{Partition}(G, \hat{W}, d_{\{u\}})$ and $L_0, R_0 \leftarrow \text{Partition}(G, W_0, d_{\{u\}})$.
5. For $t = 0, 1, \ldots, \log n$ do:

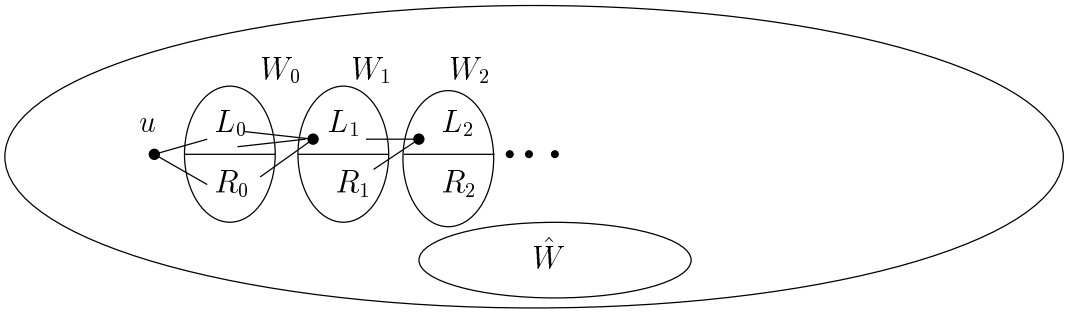

Figure 4.2: The sets created by procedure Find.

(a) Let $v_1, \ldots, v_s$ be an ordering of S such that $f(v_1) \geq f(v_2) \geq \cdots \geq f(v_s)$, where $f(v) = d_{\hat{L}_t}(v) - d_{\hat{R}_t}(v)$. If $\max_j\{f(v_j) - f(v_{j+1})\} \geq D$, then let j be the first index for which $f(v_j) - f(v_{j+1}) \geq \frac{1}{3}D$, and return $\{v_1, \ldots, v_j\}$. Otherwise, continue.

(b) Randomly select a set W_{t+1} from V' of $2l_{t+1}$ unchosen vertices.

(c) $\hat{L}_{t+1}, \hat{R}_{t+1} \leftarrow \text{Partition}(G, \hat{W}, d_{L_t} - d_{R_t})$.

(d) $L_{t+1}, R_{t+1} \leftarrow \text{Partition}(G, W_{t+1}, d_{L_t} - d_{R_t})$.

6. Output 'Failure'.

See also Figure 4.2. The total number of vertices selected by procedure Find is at most $s + 1 + 2\hat{l} + 2\sum_{t=0}^{\log n+1} l_t$, and this number can be more than $n/3$ for some choice of parameters, making the procedure inapplicable. However, we assume that the bound $k \geq \Omega(\Delta^{-1}\sqrt{n}\log n)$ is satisfied, and in that case, $\Delta^{-1} \leq O(k/\sqrt{n}\log n) \leq O(\sqrt{n}/m\log n)$. Therefore, $s \leq \hat{o}(n/m\log n)$ and $\hat{l} \leq \hat{o}(n)$. It follows that the total number of vertices selected by Find is at most $n/3$. The requirement $k \geq \Omega(\Delta^{-1}\sqrt{n}\log n)$ also implies that $n \geq c\sqrt{n}$ for some constant c, so $n \geq c^2$. Assuming that the constant c is large, we obtain that n is large.

Note that in the first three iterations of Step 5, the sets L_t, R_t are bigger than in the rest of the steps. The reason is that in these steps, the imbalance is small, so in order to get a small *relative* difference between the imbalances and the expected imbalances, we need to use bigger sets (see Theorem 4.4.1). We also note that the $\Theta(\log n)$ or $\Theta(\log^2 n)$ size ratio between $\hat{L}_t, \hat{R}_t$ and L_t, R_t has an important role: It allows us to ensure that the requirement $|b'_i - b'_j| \leq \alpha/\Delta\sqrt{l'}$ of Lemma 4.4.3 is satisfied for a small α.

Denote $b_i^t = \mathrm{I}(V_i, L_t, R_t)$, $\hat{b}_i^t = \mathrm{I}(V_i, \hat{L}_t, \hat{R}_t)$, $r_i^t = b_i^t/b_1^t$, and $\hat{r}_i^t = \hat{b}_i^t/\hat{b}_1^t$. In the following, we assume w.l.o.g. that the vertex u chosen in Step 3 is from V_1. Let t^* be the value of t when the procedure Find stops. For simplicity, we assume that $t^* \geq 4$.

Lemma 4.5.1. *Suppose that* $\Gamma \leq O(1/m\log n)$, $k \geq \Omega(\Delta^{-1}\sqrt{n}\log n)$, *and* $p, r \in [\frac{1}{4}, \frac{3}{4}]$. *Then, w.h.p.,*

1. *For all* i, $|b_i^t - \hat{b}_i^t| \le \hat{o}(\frac{\Delta}{m})$ *for* $0 \le t \le 2$, *and* $|b_i^t - \hat{b}_i^t| \le \hat{o}(\frac{\Delta}{m}\sqrt{\log n})$ *for* $3 \le t \le t^*$.

2. $b_1^0, \hat{b}_1^0 \ge (1 - \hat{o}(1))\frac{\Delta}{m}$.

3. *If* $m \ge 3$, *then for all* $i > 1$, $-\frac{1}{2} - \hat{o}(1) \le r_i^0, \hat{r}_i^0 \le \hat{o}(1)$.

4. *For* $1 \le t \le 3$, $b_1^t, \hat{b}_1^t \ge \log^{t/2} n \cdot \frac{\Delta}{m}$.

5. *For* $4 \le t \le t^*$, $b_1^t, \hat{b}_1^t \ge 2^{t-3}\log^{3/2} n \cdot \frac{\Delta}{m}$.

6. *If* $m \ge 3$, *then for all* $1 \le t \le 3$ *and all* $i > 1$,

$$(1+\delta)^t(\min(0, r_i^0) - \delta t) \le r_i^t \le (1+\delta)^t(\max(0, r_i^0) + \delta t)$$

and

$$(1+\delta)^t(\min(0, \hat{r}_i^0) - \delta t) \le \hat{r}_i^t \le (1+\delta)^t(\max(0, \hat{r}_i^0) + \delta t)$$

where $\delta \le \hat{o}(1)$.

7. *If* $m \ge 3$, *then for all* $4 \le t \le t^*$ *and all* $i > 1$,

$$(1+\delta')^{t-3}(\min(0, r_i^3) - \delta'(t-3)) \le r_i^t \le (1+\delta')^{t-3}(\max(0, r_i^3) + \delta'(t-3))$$

and

$$(1+\delta')^{t-3}(\min(0, \hat{r}_i^3) - \delta'(t-3)) \le \hat{r}_i^t \le (1+\delta')^{t-3}(\max(0, \hat{r}_i^3) + \delta'(t-3))$$

where $\delta' \le \hat{o}(1/\log n)$.

Note that as $t^* \le \log n$, we have from (6) and (7) that $-\frac{1}{2} - \hat{o}(1) \le r_i^t \le \hat{o}(1)$ for all t when $m \ge 3$. Furthermore, in the case $m = 2$, we have by definition that $b_2^t = -b_1^t$ and $\hat{b}_2^t = -\hat{b}_1^t$ for all t, so $r_2^t, \hat{r}_2^t = -1$ for all t.

Proof. In the proof of items 2–7, we only prove the bounds on b_i^t and r_i^t, as the bounds on $\hat{b}_i^t$ and $\hat{r}_i^t$ are similar. The lemma is proved using induction on t.

(1) Denote $c_i^0 = c_i(d_{\{u\}})$ and $c_i^t = c_i(d_{L_t} - d_{R_t})$ for $t \ge 1$. By Theorem 4.4.1 we have w.h.p. that $|b_i^t - c_i^t| \le d_i$ and $|\hat{b}_i^t - c_i^t| \le d_i$ for all i and $t \le 2$, where

$$d_i \le O(\sqrt{\frac{a_i}{l}\log n}(1 + \sqrt{a_i m})) = O(\sqrt{\frac{1}{ml}\log n}) \le \hat{o}(\frac{\Delta}{m}).$$

(note that the difference between $\hat{b}_i^t$ and c_i^t is even smaller). Furthermore, for $3 \le t \le t^*$ we have w.h.p. that $|b_i^t - c_i^t| \le d_i'$ and $|\hat{b}_i^t - c_i^t| \le d_i'$ for all i, where

$$d_i' \le O(\sqrt{\frac{a_i}{l'}\log n}(1 + \sqrt{a_i m})) = O(\sqrt{\frac{1}{ml'}\log n}) \le \hat{o}(\frac{\Delta}{m}\sqrt{\log n}).$$

Throughout this proof we assume that these inequalities hold. In particular, for every i, $|b_i^t - \hat{b}_i^t| \le 2d_1$ for $t \le 2$ and $|b_i^t - \hat{b}_i^t| \le 2d_1'$ for $3 \le t \le t^*$.

(2) From Lemma 4.4.2, we have that

$$c_1^0 = 2a_1(1-a_1)\Delta \ge 2(1-o(1))\frac{1}{m}(1-(1+o(1))\frac{1}{m})\Delta \ge (1-o(1))\frac{\Delta}{m}.$$

Therefore, $b_1^0 \ge c_1^0 - d_1 \ge (1-\hat{o}(1))\frac{\Delta}{m}$. Note that for $m \ge 3$ we have that $b_1^0 \ge (1-\hat{o}(1))\frac{4}{3}\frac{\Delta}{m}$.

(3) By Lemma 4.4.2, $c_i^0 = -2a_1a_i\Delta$ for every $i > 1$. Therefore, $b_i^0 \le c_i^0 + d_i \le d_i \le \hat{o}(\frac{\Delta}{m})$. Thus, $r_i^0 \le \hat{o}(1)$. Furthermore, $c_i^0 \ge -2((1+o(1))\frac{1}{m})^2\Delta \ge -(1+o(1))\frac{2}{3}\frac{\Delta}{m}$, so $b_i^0 \ge c_i^0 - d_i \ge -(1+\hat{o}(1))\frac{2}{3}\frac{\Delta}{m}$. Thus, $r_i^0 \ge -(1+\hat{o}(1))\frac{2}{3}\frac{\Delta}{m}/((1-\hat{o}(1))\frac{4}{3}\frac{\Delta}{m}) = -\frac{1}{2} - \hat{o}(1)$.

(4) As $b_1^0 \ge \frac{1}{2}\frac{\Delta}{m}$, it suffices to prove that $b_1^{t+1} \ge 2\sqrt{\log n}\cdot b_1^t$ for all $0 \le t \le 2$. Fix some $0 \le t \le 2$. For the rest of the proof, we assume w.l.o.g. that $b_1^t \ge b_2^t \ge \cdots \ge b_m^t$. By Corollary 4.2.6, w.h.p., $||V_i \cap S| - a_i s| \le O(\sqrt{a_i s \log n}) = O(\sqrt{(1/m)\cdot s\log n}) \le 1/2 \cdot s/m$ for all i, hence $|V_i \cap S| \ge a_i s - 1/2 \cdot s/m = \Omega(s/m) = \Omega(\log n/\Delta^2)$. In particular, $V_1 \cap S \ne \phi$ and $S - V_1 \ne \phi$.

By Corollary 4.2.5, w.h.p., $|f(v) - \mathrm{E}[f(v)]| < \frac{1}{2}D$ for all $v \in S$. From the proof of Lemma 4.4.3, we have that for a vertex $v \in V_i \cap S$, $\mathrm{E}[f(v)] = \Delta\hat{l}\hat{b}_i^t$. Since the algorithm did not stop at iteration t, we have that $\Delta\hat{l}(\hat{b}_1^t - \hat{b}_2^t) \le 2D$, because otherwise, the difference between the minimum value of $f(v)$ for $v \in V_1 \cap S$ and the maximum value of $f(v')$ for $v' \in S - V_1$ would be at least D. From (1) and (6) we have that for all i, j, $|b_i^t - b_j^t| \le |\hat{b}_i^t - \hat{b}_j^t| + \hat{o}(\frac{\Delta}{m}) \le 2(\hat{b}_1^t - \hat{b}_2^t) + \hat{o}(\frac{\Delta}{m})$. By induction, $\hat{b}_1^t - \hat{b}_2^t \ge (1-\hat{o}(1))\hat{b}_1^t \ge \Omega(\frac{\Delta}{m})$. Therefore, for all i, j, $|b_i^t - b_j^t| \le (2+\hat{o}(1))(\hat{b}_1^t - \hat{b}_2^t) \le O(D/\Delta\hat{l}) = O(\sqrt{\log n}/\Delta\sqrt{\hat{l}})$. As $\hat{l} = \Theta(l\log n)$, we conclude that for all i, j, $|b_i^t - b_j^t| \le \alpha/\Delta\sqrt{l}$ where $\alpha \le \hat{o}(1)$.

Denote $F_0 = \gamma\Delta\sqrt{l}$ (where γ is defined in Lemma 4.4.3), and $F = F_0/m$. By Lemma 4.4.3,

$$\begin{aligned} b_1^{t+1} &\ge F_0 a_1(1 - O(\Gamma\Delta + \sqrt{\frac{1}{ml}\log n}\Delta) - \frac{2}{9}\alpha^2)(b_1^t - \sum_{j=1}^m a_j b_j^t) - 3\frac{a_1}{\sqrt{l}} - d_1 \\ &\ge F_0(1-o(1))\frac{1}{m}(1-\hat{o}(1))(b_1^t - \sum_{j=1}^m a_j b_j^t) - 3\frac{a_1}{\sqrt{l}} - d_1 \\ &= (1-\hat{o}(1))F(b_1^t - \sum_{j=1}^m a_j b_j^t) - 3\frac{a_1}{\sqrt{l}} - d_1. \end{aligned}$$

From (6) we have that $|b_j^t| \le b_1^t$ for all j, and therefore, using the fact that $\sum_{j=1}^m b_j^t = 0$, we have that

$$\sum_{j=1}^m a_j b_j^t = \sum_{j=1}^m (a_j - \frac{1}{m})b_j^t \le \sum_{j=1}^m |a_j - \frac{1}{m}|b_1^t \le m\Gamma b_1^t \le \hat{o}(\frac{1}{\log n}b_1^t).$$

Furthermore, from the fact that $F = \Omega(\Delta\sqrt{l}/m) \geq \Omega(\sqrt{\log n})$ and $b_1^t \geq \Omega(\Delta/m)$ we obtain

$$3\frac{a_1}{\sqrt{l}} \leq O(\frac{1}{\sqrt{l}}) \leq O(\frac{1}{\sqrt{\log n}}\frac{\Delta}{m}) \leq o(Fb_1^t),$$

and

$$d_1 \leq \hat{o}(\frac{\Delta}{m}) \leq o(Fb_1^t).$$

Therefore, $b_1^{t+1} \geq (1 - \hat{o}(1))Fb_1^t$. In particular, $b_1^{t+1} \geq 2\sqrt{\log n} \cdot b_1^t$.

(5) It suffices to prove that $b_1^{t+1} \geq 2b_1^t$ for all $3 \leq t \leq t^* - 1$. Similarly to the proof of (4), we have that for all i, j, $|b_i^t - b_j^t| \leq O(\sqrt{\log n}/\Delta\sqrt{\hat{l}})$, and since $\hat{l} = \Theta(l' \log^2 n)$, we conclude that $|b_i^t - b_j^t| \leq \alpha_2/\Delta\sqrt{l'}$ where $\alpha_2 \leq \hat{o}(1/\sqrt{\log n})$.

Denote $F_0' = \gamma\Delta\sqrt{l'}$, and $F' = F_0'/m$. By Lemma 4.4.3,

$$\begin{aligned} b_1^{t+1} &\geq F_0' a_1 (1 - O(\Gamma\Delta + \sqrt{\frac{1}{ml'}\log n}\Delta) - \frac{2}{9}\alpha_2^2)(b_1^t - \sum_{j=1}^{m} a_j b_j^t) - 3\frac{a_1}{\sqrt{l'}} - d_1' \\ &\geq (1 - \hat{o}(\frac{1}{\log n}))F' b_1^t - 3\frac{a_1}{\sqrt{l'}} - d_1'. \end{aligned}$$

Furthermore,

$$3\frac{a_1}{\sqrt{l'}} \leq O(\frac{1}{\sqrt{l'}}) \leq O(\frac{\Delta}{m}) \leq o(\frac{1}{\log n}F' b_1^t),$$

where the last equality follows from the fact that $F' = \Omega(\Delta\sqrt{l'}/m) \geq \Omega(1)$ and $b_1^t \geq \log^{3/2} n \cdot \Delta/m$. We also have

$$d_1' \leq \hat{o}(\frac{\Delta}{m}\sqrt{\log n}) \leq \hat{o}(\frac{1}{\log n}F' b_1^t).$$

Therefore, $b_1^{t+1} \geq (1 - \hat{o}(1/\log n))F' b_1^t$. In particular, $b_1^{t+1} \geq 2b_1^t$.

(6) We begin by giving an upper bound on b_i^{t+1}. By Lemma 4.4.2 and the fact that $\sum_{j=1}^{m} a_j b_j^t \geq -m\Gamma b_1^t \geq -o(b_1^t)$,

$$\begin{aligned} b_i^{t+1} &\leq F_0 a_i (b_i^t - \sum_{j=1}^{m} a_j b_j^t + (O(\Gamma\Delta + \sqrt{\frac{1}{ml}\log n}\Delta) + \frac{2}{9}\alpha^2)(b_1^t - b_m^t)) + 3\frac{a_i}{\sqrt{l}} + d_i \\ &\leq F_0 a_i (b_i^t + o(b_1^t) + \hat{o}(1)(b_1^t - b_m^t)) + o(Fb_1^t) \\ &\leq Fb_i^t + o(1)F|b_i^t| + \hat{o}(1)Fb_1^t + \hat{o}(1)F(b_1^t - b_m^t). \end{aligned}$$

Since $|b_i^t| \leq b_1^t$ and $b_1^t - b_m^t \leq 2b_1^t$, it follows that

$$b_i^{t+1} \leq Fb_i^t + \hat{o}(1)Fb_1^t = (r_i^t + \hat{o}(1))Fb_1^t \leq (\max(0, r_i^t) + \hat{o}(1))Fb_1^t.$$

Thus, we have that $b_1^{t+1} \geq (1-\delta_0)Fb_1^t$ and $b_i^{t+1} \leq (\max(0,r_i^t)+\delta_0)Fb_i^t$ for $\delta_0 \leq \hat{o}(1)$. Hence, for $\delta = 2\delta_0$ we have

$$\begin{aligned} r_i^{t+1} &\leq \frac{\max(0,r_i^t)+\delta_0}{1-\delta_0} \leq (1+\delta)\max(0,r_i^t)+\delta \\ &\leq (1+\delta)(1+\delta)^t(\max(0,r_i^0)+\delta t)+\delta \\ &\leq (1+\delta)^{t+1}(\max(0,r_i^0)+(t+1)\delta). \end{aligned}$$

The lower bound on r_i^{t+1} is shown using similar arguments: We have that

$$\begin{aligned} b_i^{t+1} &\geq F_0 a_i(b_i^t - \sum_{j=1}^{m} a_j b_j^t - (O(\Gamma\Delta + \sqrt{\frac{1}{ml}\log n}\Delta) + \frac{2}{9}\alpha^2)(b_1^t - b_m^t)) - 3\frac{a_i}{\sqrt{l}} - d_i \\ &\geq Fb_i^t - \hat{o}(1)Fb_1^t \geq (\min(0,r_i^t) - \hat{o}(1))Fb_1^t. \end{aligned}$$

Therefore,

$$r_i^{t+1} \geq \frac{\min(0,r_i^t)-\delta_0}{1-\delta_0} \geq (1+\delta)\min(0,r_i^t) - \delta \geq (1+\delta)^{t+1}\min(0,r_i^0) - (t+1)\delta.$$

(7) By Lemma 4.4.2,

$$\begin{aligned} b_i^{t+1} &\leq F_0' a_i(b_i^t - \sum_{j=1}^{m} a_j b_j^t + (O(\Gamma\Delta + \sqrt{\frac{1}{ml'}\log n}\Delta) + \frac{2}{9}\alpha_2^2)(b_1^t - b_m^t)) + 3\frac{a_i}{\sqrt{l'}} + d_i' \\ &\leq F'b_i^t + \hat{o}(\frac{1}{\log n})F'b_1^t \leq (\max(0,r_i^t) + \hat{o}(\frac{1}{\log n}))F'b_1^t. \end{aligned}$$

Thus, we have that $b_1^{t+1} \geq (1-\delta_0')F'b_1^t$ and $b_i^{t+1} \leq (\max(0,r_i^t)+\delta_0')F'b_i^t$ for $\delta_0' \leq \hat{o}(1/\log n)$. Therefore, $r_i^{t+1} \leq (1+\delta')^{t-3}(\max(0,r_i^3)+\delta'(t-3))$ for $\delta' = 2\delta_0'$. The lower bound is proved in a similar manner. ■

Lemma 4.5.2. *Suppose that* $\Gamma \leq O(1/m\log n)$, $k \geq \Omega(\Delta^{-1}\sqrt{n}\log n)$, *and* $p, r \in [\frac{1}{4}, \frac{3}{4}]$. *Then, w.h.p., procedure* Find *returns a subcluster of size* $\Omega(m\log n/\Delta^2)$. *The running time of procedure* Find *is* $O(m^3\Delta^{-4}\log^3 n \cdot (m+\log n) + \log^9 n)$.

Proof. By Lemma 4.5.1, for all $4 \leq t \leq t^*$, $b_1^t \geq 2^{t-3}\log^{3/2} n \cdot \frac{\Delta}{m} > 2^t \cdot \frac{\Delta}{m}$. As $m \leq n/k \leq O(\Delta\sqrt{n}/\log n)$, we have that $b_1^t \geq \Omega(2^t \log n/\sqrt{n}) > 2^t/n$. Since $b_1^t \leq 1$ by definition, we conclude that $t^* < \log n$, namely procedure Find stops at Step 5a. We now look at the values of f at the last iteration. W.l.o.g. assume that $\hat{b}_1^{t^*} \geq \hat{b}_2^{t^*} \geq \cdots \geq \hat{b}_m^{t^*}$. From the proof of Lemma 4.5.1 (4), $V_i \cap S \neq \phi$ for all i. By Corollary 4.2.5, w.h.p., $|f(v) - \mathrm{E}[f(v)]| < \frac{1}{6}D$ for all $v \in S$, where $\mathrm{E}[f(v)] = \Delta\hat{l}\hat{b}_i^{t^*}$ for $v \in V_i$. As $\max_j\{f(v_j) - f(v_{j+1})\} \geq D$, we have that there is an index i such that $\Delta\hat{l}\hat{b}_i^{t^*} - \Delta\hat{l}\hat{b}_{i+1}^{t^*} \geq \frac{2}{3}D$. By Lemma 4.5.1, $\Delta\hat{l}(\hat{b}_1^{t^*} - \hat{b}_2^{t^*}) \geq \Delta\hat{l}(\hat{b}_i^{t^*} - \hat{b}_{i+1}^{t^*}) \geq \frac{2}{3}D$. Therefore, $\min_{v\in V_1\cap S} f(v) - \max_{v\in S-V_1} f(v) > \frac{1}{3}D$. Furthermore, for $v, v' \in V_1 \cap S$, $|f(v) - f(v')| < \frac{1}{3}D$. It follows that $\{v_1, \ldots, v_j\} = V_1 \cap S$, and in particular,

$\{v_1, \ldots, v_j\}$ is a subcluster. Finally, it was shown in the proof of Lemma 4.5.1 (4) that $|V_1 \cap S| = \Omega(\log n/\Delta^2)$.

We now compute the time complexity of Find: In a single iteration, the time to compute $f(v)$ for all $v \in S$ is $O(s\hat{l})$, and the time taken by the call to Partition is $O(l_t\hat{l})$. Therefore, the time complexity of Find is

$$\begin{aligned} O(s\hat{l}\log n + l\hat{l} + l'\hat{l}\log n) &= O(\frac{m^3}{\Delta^4}\log^4 n + \log^7 n + \frac{m^4}{\Delta^{-4}}\log^3 n + \frac{\Delta^4}{m^2}\log^9 n) \\ &= O(\frac{m^3}{\Delta^4}\log^3 n \cdot (m + \log n) + \log^9 n). \end{aligned}$$ ■

Combining Lemma 4.5.2 and Lemma 4.3.2 gives the following theorem:

Theorem 4.5.3. *The above algorithm solves the clustering problem w.h.p. when* $\Gamma \leq O(1/m\log n)$ *and* $k \geq \Omega(\Delta^{-1}\sqrt{n}\log n)$. *The running time of the algorithm is* $O(m^4\Delta^{-4}\log^3 n \cdot (m + \log n) + m\Delta^{-2}n\log n)$.

As noted above, the requirement $k \geq \Omega(\Delta^{-1}\sqrt{n}\log n)$ implies that $\Delta^{-1} \leq O(\sqrt{n}/m\log n)$. Therefore, $m^4\Delta^{-4}\log^3 n \leq O(n^2/\log n)$ and $m\Delta^{-2}n\log n \leq O(n^2/m\log n)$. It follows that the time complexity of the algorithm is $O((m/\log n + 1)n^2)$.

Note that algorithm requires the knowledge of m and Δ. However, if we use take $\hat{l} = \Theta(M^2\log^2 n + \log^5 n)$ and $s = \Theta(M^2\log n)$ where $M \geq m/\Delta$, then procedure Find remains correct, and its time complexity becomes $O(M^4\log^4 n + \log^9 n)$. Since $m/\Delta \leq O(\sqrt{n}/\log n)$, one can use $M = \Theta(\sqrt{n}/\log n)$, and the time complexity of procedure Find will be $O(n^2)$. The overall time complexity of the algorithm in this case is $O(mn^2)$.

4.6 Unequal sized clusters

In this section we give an algorithm for finding a subcluster in the general case. We use algorithm Solve_3 and procedure Find_2 which is similar to procedure Find that was given in Section 4.5.

We first describe the main differences between procedures Find_2 and Find. First, unlike procedure Find, in which the vertex u was chosen randomly, in procedure Find_2 the vertex u must be chosen in a way that ensures that the size of the cluster that contains u is almost equal to the size of the largest cluster. This is done in the following way: Procedure Find_2 randomly chooses two sets of vertices S_1 and S_2, and then selects u to be the vertex from S_1 with highest d_{S_2}-value. A second difference comes from the fact that our analysis in the general case is less tight than the analysis in the case of almost equal sized clusters. As a result, procedure Find_2 can perform only a constant number of iterations. This leads to a different choice of parameters (namely, the sizes of the sets built by the algorithm).

Procedure Find_2 is as follows:

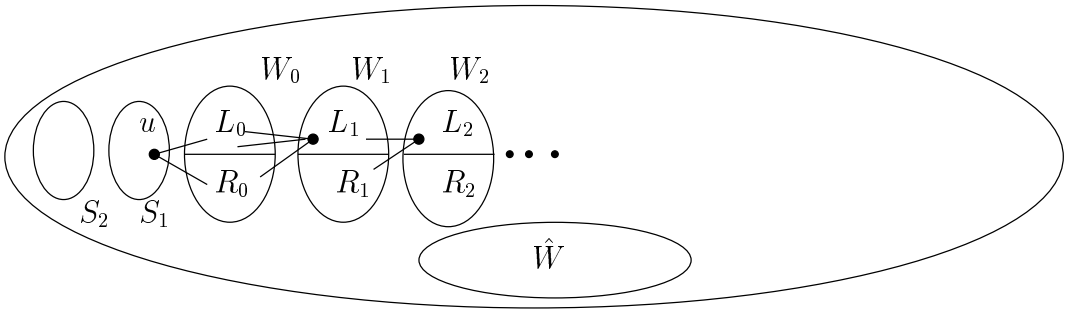

Figure 4.3: The sets created by procedure Find_2.

$\text{Find}_2(G, V')$

1. Let $\epsilon = \Theta(1)$, $c = \Theta(1)$, $\hat{l} = \Theta(m\Delta^{-2}\log^2 n + m^2\Delta^{-2(1+\epsilon)}\log n)$, $l = \Theta(\hat{l}/\log n)$, $s = \Theta(m\Delta^{-2}\log n)$, $s_1 = \Theta(m\log n)$, $s_2 = \Theta(n)$, and $D = \Theta(\sqrt{\hat{l}\log n})$.
2. Randomly select disjoint sets of vertices S_1, S_2, S, $\hat{W}$, and W_0 from V' of sizes s, s_1, s_2, $2\hat{l}$, and $2l$, respectively.
3. Let u be a vertex in S_1 such that $d_{S_2}(u)$ is maximum.
4. $\hat{L}_0, \hat{R}_0 \leftarrow \text{Partition}(G, \hat{W}, d_{\{u\}})$ and $L_0, R_0 \leftarrow \text{Partition}(G, W_0, d_{\{u\}})$.
5. For $t = 0, 1, \ldots, \lceil 1/\epsilon \rceil$ do:
   (a) Let $v_1, \ldots, v_s$ be an ordering of S such that $f(v_1) \geq f(v_2) \geq \cdots \geq f(v_s)$, where $f(v) = d_{\hat{L}_t}(v) - d_{\hat{R}_t}(v)$. If $\max_j\{f(v_j) - f(v_{j+1})\} \geq D$, then let j be the first index for which $f(v_j) - f(v_{j+1}) \geq cD$, and return $\{v_1, \ldots, v_j\}$. Otherwise, continue.
   (b) Randomly select a set W_{t+1} from V' of $2l$ unchosen vertices.
   (c) $\hat{L}_{t+1}, \hat{R}_{t+1} \leftarrow \text{Partition}(G, \hat{W}, d_{L_t} - d_{R_t})$.
   (d) $L_{t+1}, R_{t+1} \leftarrow \text{Partition}(G, W_{t+1}, d_{L_t} - d_{R_t})$.

See Figure 4.3. The total number of vertices selected by procedure Find_2 is $s + s_1 + s_2 + 2\hat{l} + (\lceil 1/\epsilon \rceil + 1)\cdot 2l$. We assume that the bound $k \geq \Omega(\Delta^{-1-\epsilon}\sqrt{n\log n})$ is satisfied, and in that case, $\Delta^{-1-\epsilon} \leq O(k/\sqrt{n\log n}) \leq O(\sqrt{n/\log n}/m)$. Therefore $s \leq O(m^{1-2/(1+\epsilon)}n^{1/(1+\epsilon)}\log^{1-1/(1+\epsilon)} n) \leq o(n)$ and $\hat{l} \leq \hat{o}(n)$. Furthermore, $m \leq n/k \leq O(\Delta^{1+\epsilon}\sqrt{n\log n}) \leq O(\sqrt{n\log n})$, so $s_1 \leq O(\sqrt{n}\log^{3/2} n)$. It follows that the total number of vertices selected by Find is at most $n/3$.

We assume w.l.o.g. that the vertex u chosen in Step 3 belongs to the cluster V_1. Let $K = \max_i |V_i|$. Recall that $A_i = |V_i|$.

Lemma 4.6.1. *If $k \geq \Omega(\sqrt{n\log n})$ then w.h.p., $A_1 \geq K - O(\Delta^{-1}\sqrt{n\log n})$.*

Proof. Let V_i be a cluster of maximum size. The probability that S_1 does not contain a vertex from V_i is at most $(1-a_i)^{s_1} \le e^{-a_i s_1} \le e^{-s_1/m} = 1/n^{\Omega(1)}$. Therefore, w.h.p., the set S_1 contains a vertex w from V_i. Denote $A'_j = |S_2 \cap V_j|$ for all j. By Corollary 4.2.5, w.h.p. $|d_{S_2}(v) - \mathrm{E}[d_{S_2}(v)]| \le O(\sqrt{s_2 \log s_2}) = O(\sqrt{n \log n})$ for every $v \in S_1$, where for $v \in V_j$ we have $\mathrm{E}[d_{S_2}(v)] = A'_j\Delta + s_2 r$. Thus, w.h.p.

$$A'_1\Delta + s_2 r + O(\sqrt{n\log n}) \ge d_{S_2}(u) \ge d_{S_2}(w) \ge A'_i\Delta + s_2 r - O(\sqrt{n\log n}),$$

so w.h.p. $A'_1 \ge A'_i - O(\Delta^{-1}\sqrt{n\log n})$. By Corollary 4.2.6, w.h.p., $|A'_j - \frac{A_j}{n}s_2| \le O(\sqrt{a_j s_2 \log n}) \le O(\sqrt{n\log n})$ for every j. Therefore, w.h.p.

$$A_1 \ge \frac{n}{s_2}A'_1 - O(\frac{n}{s_2}\sqrt{n\log n}) \ge \frac{n}{s_2}A'_i - O(\Delta^{-1}\sqrt{n\log n}) \ge A_i - O(\Delta^{-1}\sqrt{n\log n}).$$

■

In the following lemma we use the same definitions of b_i^t, $\hat{b}_i^t$, r_i^t, $\hat{r}_i^t$, c_i^t, and t^* as in Section 4.5.

Lemma 4.6.2. *Suppose that* $\max_i a_i < 2/3$, $k \ge \Omega(\Delta^{-1-\epsilon}\sqrt{n\log n})$ *for some* $\epsilon > 0$, $p, r \in [\frac{1}{4}, \frac{3}{4}]$, *and* $\Delta \le O(1)$. *Then, w.h.p.,*

1. *For all* i *and all* $0 \le t \le t^*$, $|b_i^t - \hat{b}_i^t| \le \hat{o}(a_1\Delta)$.
2. $b_1^0, \hat{b}_1^0 \ge (1-\hat{o}(1))\frac{2}{3}a_1\Delta$.
3. *If* $m \ge 3$, *then for all* $i > 1$, $-(1+\hat{o}(1)) \le r_i^0, \hat{r}_i^0 \le \hat{o}(1)$.
4. *For all* $1 \le t \le t^*$, $b_1^t, \hat{b}_1^t \ge \Omega(\Delta^{-\epsilon t}a_1\Delta)$.
5. *If* $m \ge 3$, *then for all* $1 \le t \le t^*$ *and all* $i > 1$, $-O(1) \le r_i^t, \hat{r}_i^t \le 1 - \Omega(1)$.

Proof. Again, we prove items 2–5 only for b_i^t and r_i^t.

(1) By Theorem 4.4.1 we have w.h.p. that $|b_i^t - c_i^t| \le d_i$ and $|\hat{b}_i^t - c_i^t| \le d_i$ for all i and t, where

$$d_i \le O(\sqrt{\frac{a_i}{l}\log n}(1+\sqrt{a_i m})).$$

By Lemma 4.6.1 we have $A_1 \ge K - O(\Delta^{-1}\sqrt{n\log n}) \ge (1-\hat{o}(1))K$. As $K \ge n/m$, we obtain that $a_1 \ge \Omega(1/m)$ and therefore $d_1 \le O(\sqrt{\frac{a_1}{l}\log n}\cdot\sqrt{a_1 m}) \le \hat{o}(a_1\Delta)$. Furthermore, for $i > 1$, $a_i \le O(a_1)$ so $d_i \le O(d_1) \le \hat{o}(a_1\Delta)$.

Throughout this proof we assume that these inequalities hold. In particular, $|b_i^t - \hat{b}_i^t| \le 2d_1 = \hat{o}(a_1\Delta)$.

(2) By Lemma 4.4.2, we have that

$$c_1^0 = 2a_1(1-a_1)\Delta > \frac{2}{3}a_1\Delta$$

Therefore, $b_1^0 \geq c_1^0 - d_1 \geq (1-\hat{o}(1))\frac{2}{3}a_1\Delta$.

(3) By Lemma 4.4.2, $c_i^0 = -2a_1a_i\Delta$ for every $i > 0$. Therefore, $b_i^0 \leq c_i^0 + d_i \leq d_i \leq \hat{o}(a_1\Delta)$. Furthermore, $c_i^0 \geq -c_1^0$ (as $a_i \leq 1 - a_1$), so $b_i^0 \geq c_i^0 - d_i \geq -(1+\hat{o}(1))c_1^0$. Thus, $-(1+\hat{o}(1)) \leq r_i^0 \leq \hat{o}(1)$.

(4) We will show that $b_1^{t+1} \geq \Delta^{-\epsilon}b_1^t$ for all $t < t^*$. W.l.o.g. assume that $b_1^t \geq b_2^t \geq \cdots \geq b_m^t$.

As in the proof of item 4 in Lemma 4.5.1, we have that $|b_i^t - b_j^t| \leq \alpha/\Delta\sqrt{l}$ for all i, j where $\alpha = \hat{o}(1)$. (In this case we have that $|b_i^t - b_j^t| \leq |\hat{b}_i^t - \hat{b}_j^t| + \hat{o}(a_1\Delta) \leq O(\hat{b}_1^t - \hat{b}_2^t)$. Furthermore, we need to show that $V_1 \cap S \neq \phi$ and $S - V_1 \neq \phi$. Since $|V_1| \geq \Omega(n/m)$, we have that $|V_1 \cap S| \geq \Omega(\log n/\Delta^2)$. Moreover, as $|V - V_1| > n/3$, we have that $|(V - V_1) \cap S| \geq \Omega(m \log n/\Delta^2)$. In particular, we have that $V_1 \cap S \neq \phi$ and $S - V_1 \neq \phi$.)

Denote $F_0 = \gamma\Delta\sqrt{l}$ (where γ is defined in Lemma 4.4.3), $F = F_0K/n$ and $Y = (\sum_{j=1}^m a_j b_j^t)/b_1^t$. By Lemma 4.4.3,

$$\begin{aligned} b_1^{t+1} &\geq F_0a_1(1 - O(\Delta) - \frac{2}{9}\alpha^2)(b_1^t - \sum_{j=1}^m a_j b_j^t) - 3\frac{a_1}{\sqrt{l}} - d_1 \\ &\geq (1-\hat{o}(1))(1-Y)Fb_1^t - 3\frac{a_1}{\sqrt{l}} - d_1 \end{aligned}$$

where the last inequality follows from the fact that $\Delta \leq O(1)$ and $A_1 \geq (1 - \hat{o}(1))K$. We have that $F_0 \geq \Omega(\Delta\sqrt{l}/m) \geq \Omega(1)$ and $b_1^t \geq \Omega(a_1\Delta)$. Therefore,

$$3\frac{a_1}{\sqrt{l}} \leq O(a_1\frac{\Delta}{\sqrt{m\log n}}) \leq o(F_0b_1^t)$$

and

$$d_1 \leq \hat{o}(a_1\Delta) \leq \hat{o}(F_0b_1^t).$$

It follows that $b_1^{t+1} \geq ((1-\hat{o}(1))(1-Y) - \hat{o}(1))F_0b_1^t$.

Now,

$$\sum_{j=1}^m a_j b_j^t \leq a_1b_1^t + \sum_{j=2}^m a_j b_2^t = a_1b_1^t + (1-a_1)b_2^t \leq \frac{2}{3}b_1^t + \frac{1}{3}b_2^t = (\frac{2}{3} + \frac{1}{3}r_2^t)b_1^t.$$

Therefore, $b_1^{t+1} \geq ((1-\hat{o}(1))\frac{1}{3}(1-r_2^t) - \hat{o}(1))F_0b_1^t \geq \Omega(F_0b_1^t) \geq \Omega(\Delta\sqrt{l}/m \cdot b_1^t) \geq \Delta^{-\epsilon}b_1^t$.

(5) By Lemma 4.4.2,

$$\begin{aligned} b_i^{t+1} &\le F_0 a_i (b_i^t - \sum_{j=1}^{m} a_j b_j^t + (O(\Delta) + \frac{2}{9}\alpha^2)(b_1^t - b_m^t)) + 3\frac{a_i}{\sqrt{l}} + d_i \\ &\le F_0 a_i (b_i^t - \sum_{j=1}^{m} a_j b_j^t) + \hat{o}(F_0 a_i b_1^t) + \hat{o}(F b_1^t) \\ &\le F \max(0, b_i^t - \sum_{j=1}^{m} a_j b_j^t) + \hat{o}(F b_1^t) \\ &= (\max(0, r_i^t - Y) + \hat{o}(1)) F b_1^t. \end{aligned}$$

We have shown above that

$$b_1^{t+1} \ge ((1 - \hat{o}(1))(1 - Y) - \hat{o}(1)) F b_1^t \ge (1 - \hat{o}(1))(1 - Y) F b_1^t,$$

where the last inequality follows as $1 - Y \ge \frac{1}{3}(1 - r_2^t) \ge \Omega(1)$. Hence

$$r_i^{t+1} \le \frac{\max(0, r_i^t - Y) + \hat{o}(1)}{(1 - \hat{o}(1))(1 - Y)} \le (1 + \hat{o}(1)) \frac{\max(0, r_i^t - Y)}{1 - Y} + \hat{o}(1).$$

As

$$\sum_{j=1}^{m} a_j b_j^t \ge \sum_{j=1}^{m} a_j b_m^t = b_m^t = r_m^t b_1^t,$$

we have that $Y \ge r_m^t$, and it follows that

$$r_i^{t+1} \le (1 + \hat{o}(1)) \frac{r_i^t - r_m^t}{1 - r_m^t} + \hat{o}(1).$$

Using similar arguments we obtain that

$$\begin{aligned} r_i^{t+1} &\ge (1 + \hat{o}(1)) \frac{\min(0, r_i^t - Y)}{1 - Y} - \hat{o}(1) \ge (1 + \hat{o}(1)) \frac{r_i^t - 1}{1 - (\frac{2}{3} + \frac{1}{3} r_2^t)} - \hat{o}(1) \\ &= (1 + \hat{o}(1)) \cdot 3 \frac{r_i^t - 1}{1 - r_2^t} - \hat{o}(1). \end{aligned}$$

From the lower and upper bounds on r_i^{t+1} and the fact that the number of iterations is constant we conclude that $-O(1) \le r_i^t \le 1 - \Omega(1)$ for all t. ∎

Lemma 4.6.3. *Suppose that* $\max_i a_i < 2/3$, $k \ge \Omega(\Delta^{-1-\epsilon}\sqrt{n \log n})$, *and* $p, r \in [\frac{1}{4}, \frac{3}{4}]$. *Then, w.h.p., procedure* Find_2 *returns a subcluster of size* $\Omega(\log n/\Delta^2)$. *The running time of procedure* Find_2 *is* $O(m^4 \Delta^{-4(1+\epsilon)} \log n + m^2 \Delta^{-4} \log^3 n + m\Delta^{-1} n \log n)$.

Proof. Let α be the constant hidden in the requirement $\Delta \leq O(1)$ of Lemma 4.6.2. If $\Delta \leq \alpha$, then from Lemma 4.6.2 we conclude (similarly to the proof of Lemma 4.5.2) that w.h.p., procedure Find_2 returns a subcluster of size $\Omega(\log n/\Delta^2)$.

We now consider the case when $\Delta > \alpha$. W.l.o.g. assume that $\hat{b}_1^0 \geq \hat{b}_2^0 \geq \cdots \geq \hat{b}_m^0$. The proof of items 2 and 3 in Lemma 4.6.2 does not require that $\Delta \leq \alpha$. Therefore, we have that $\hat{b}_1^0 - \hat{b}_2^0 \geq \frac{1}{2}\hat{b}_1^0 \geq \Omega(\Delta/m)$. Thus, $\min_{v \in S \cap V_1} f(v) - \max_{v \in S - V_1} f(v) \geq \Delta\hat{l}(\hat{b}_1^0 - \hat{b}_2^0) - D \geq \Omega(\hat{l}/m) - D \geq D$, hence procedure Find_2 stops at Step 5a and $\{v_1, \ldots, v_j\} = V_1 \cap S$.

As the number of iteration is at most $\lceil 1/\epsilon \rceil = O(1)$, the time complexity of Find_2 is $O(s\hat{l} + l\hat{l} + s_1 s_2) = O(l\hat{l} + s_1 s_2) = O(m^4\Delta^{-4(1+\epsilon)} \log n + m^2\Delta^{-4} \log^3 n + mn \log n)$. ∎

Combining Lemma 4.5.2 and the above lemma we conclude:

Theorem 4.6.4. *The above algorithm solves the clustering problem w.h.p. when* $k \geq \Omega(\Delta^{-1-\epsilon}\sqrt{n \log n})$ *for some* $\epsilon > 0$. *The running time of the algorithm is* $O(m^5\Delta^{-4(1+\epsilon)} \log n + m^3\Delta^{-4} \log^3 n + (m^2 + m\Delta^{-2})n \log n)$.

The requirement $k \geq \Omega(\Delta^{-1-\epsilon}\sqrt{n \log n})$ implies that $\Delta^{-1-\epsilon} \leq O(\sqrt{n/\log n}/m)$. Therefore,

$$m^5\Delta^{-4(1+\epsilon)} \log n \leq O(mn^2/\log n)$$
$$m^3\Delta^{-4} \log^3 n \leq O(m^{3-4/(1+\epsilon)} n^{2/(1+\epsilon)} \log^{3-2/(1+\epsilon)} n) \leq o(mn^2/\log n)$$

and

$$m\Delta^{-2} n \log n \leq O(m^{1-2/(1+\epsilon)} n^{1+1/(1+\epsilon)} \log^{1-1/(1+\epsilon)} n) \leq o(mn^2/\log n).$$

Furthermore, $m \leq O(\sqrt{n/\log n})$ so $m^2 n \log n \leq O(mn^{3/2}\sqrt{\log n}) \leq o(mn^2/\log n)$. Therefore, the time complexity of the algorithm is $O(mn^2/\log n)$.

4.7 The m-cluster editing problem

In this section, we show that the m-cluster editing problem is NP-hard for every $m \geq 2$. For a graph $G = (V, E)$, a set $F \subseteq V \times V$ is called an *m-cluster editing set* of G if $G' = (V, E \triangle F)$ is a cluster graph with m connected components (maximal cliques). The decision version of the m-cluster editing problem is given a graph G and a positive integer k, to decide if there is an m-cluster editing set of G of size at most k.

Note that there is a correspondence between the cuts of G and the 2-cluster editing sets of G: The cut $(S, V - S)$ corresponds to the set $F_S = (S \times S \cup (V - S) \times (V - S)) \triangle E$. Similarly, there is a correspondence between the m-cuts of G and the m-cluster editing sets of G. We will use these facts in the proofs below.

We first prove the hardness of 2-cluster editing. We need the following definitions: For a hypergraph $G = (V, E)$, a mapping $f : V \to \{0, 1\}$ is a 2-*coloring* if there is no edge in G in which all the vertices have the same color. f is called *balanced* if the number of vertices that have each color is the same. The *hypergraph* 2-*colorability problem* is to decide whether a given hypergraph has a 2-coloring. This problem was shown to be NP-complete by Lovasz [86], and it remains NP-Complete when all edges has size 3 [86]. The *hypergraph balanced* 2-*colorability problem* is to decide whether a hypergraph has a balanced 2-coloring.

Lemma 4.7.1. *The hypergraph balanced* 2-*colorability problem is NP-hard even when each edge in the input hypergraph has size* 3.

Proof. We give a reduction from hypergraph 2-colorability when all edge have size 3. Given a hypergraph $G = (V, E)$ with n vertices, we generate a hypergraph $G' = (V', E)$ by adding n isolated vertices to G. Clearly, a balanced 2-coloring of G' induces a 2-coloring on G, and a 2-coloring of G can be extended to a balanced 2-coloring of G' by coloring the isolated vertices appropriately. ■

Theorem 4.7.2. 2-*cluster editing is NP-hard.*

Proof. We give a reduction from hypergraph balanced 2-colorability when each edge have size 3. Given a hypergraph $G = (V, E)$, we build a graph $G' = (V', E')$ as follows: Suppose that $V = \{1, \ldots, n\}$ (where n is even) and let m be the number of hyperedges in G. Let $M = 2n^3$. The vertices of G' are $V' = \cup_{i=1}^{n} V_i$ where $V_i = \{v_{i,j} : j = 1, \ldots, M\}$. For a triplet of indices $i < j < l \leq n$ define the set $E_{i,j,l} = \{(v_{i,r}, v_{j,r}), (v_{j,r+1}, v_{l,r}), (v_{l,r+1}, v_{i,r+1})\}$, where $r = 2(n^2 i + nj + l) - 1$. We add edges to G' by building a clique on V_i for every $i \leq n$, and for each triplet of indices $i < j < l \leq n$ where $\{i, j, l\} \notin E$, we add the edges of $E_{i,j,l}$ to G'. Finally, we set $k = 2\binom{n/2}{2}(M^2 - (n-2)) + (\frac{n}{2})^2(n-2) - m$.

The sets $V_1, \ldots, V_n$ will be called *clusters*. We say that a cut $(S, V' - S)$ *splits* a cluster V_i if $V_i \cap S \neq \phi$ and $V_i \not\subseteq S$. For convenience we also define a graph $G'' = (V, E'')$ which is built like G' except that it contains the edges of $E_{i,j,l}$ for every triplet $i < j < l$.

The main idea of the reduction is as follows: Suppose that we wish to build a 2-cluster editing set F of G' with minimum size, and let $(S, V' - S)$ be the corresponding cut. The reduction guarantees that we receive a large penalty if the cut $(S, V' - S)$ splits a cluster: In that case, F must contain all the edges between vertices of clusters that are on different sides of the cut. Since M is large, this penalty is large, so the cut must not split a cluster. Furthermore, to obtain minimality, there must be exactly $n/2$ cluster in each side of the cut (otherwise, there is a large penalty). Finally, for every triplet $\{i, j, l\} \in E$, the nonexistence of the edges of $E_{i,j,l}$ in G' implies that that the clusters V_i, V_j, and V_k should not be on the same side of the cut.

To prove the correctness of this reduction, we have to show that there is a balanced 2-coloring of G iff there is a 2-cluster editing set of G' of size at

most k. Suppose first that $f\colon V \to \{0,1\}$ is a balanced 2-coloring of G. Let $S = \cup_{i:f(i)=0} V_i$, and let F' and F'' be the 2-cluster editing sets of G' and G'', respectively, that correspond to the cut $(S, V-S)$. Since f is balanced, each side of the cut $(S, V-S)$ consists of $n/2$ clusters. We first compute the size of F''. For two clusters V_i and V_j (with $i<j$), each set of the form $E_{i,j,l}$, $E_{i,l,j}$, or $E_{l,i,j}$ contains exactly one edge between V_i and V_j. Therefore, between each pair of clusters in G'' there are exactly $n-2$ edges. It follows that F'' contains $2\binom{n/2}{2}(M^2-(n-2))$ edges that are not in E'' between clusters in the same side of the cut $(S, V-S)$, and $(\frac{n}{2})^2(n-2)$ edges in E'' between clusters in different sides of the cut. Thus, $|F''| = 2\binom{n/2}{2}(M^2-(n-2)) + (\frac{n}{2})^2(n-2)$. We now compute the size of F'. For each edge $(i,j,l) \in E$, the edges of $E_{i,j,l}$ in G'' contribute two edges to F'' (as the fact that f is a 2-coloring implies that the clusters V_i, V_j, and V_l are not all on the same side of the cut). The nonexistence of the edges of $E_{i,j,l}$ in G' contribute only one edge to F' (a missing edge between the two clusters on the same side of the cut). It follows that $|F'| = |F''| - m = k$.

For the converse direction of the proof, suppose that F is a 2-cluster editing set of G' of minimum size, and $|F| \leq k$. Let $(S, V'-S)$ be the cut that corresponds to F. We first claim that $(S, V'-S)$ doesn't split a cluster.

Suppose conversely that $(S, V'-S)$ splits at least one cluster. If the cut $(S, V'-S)$ splits more than one cluster, then let V_i be a cluster that is split by the cut such that $|V_i \cap S|$ is minimum, and let V_j be another cluster that is split by the cut such that $|V_j \cap S|$ is maximum. Denote $a = |V_i \cap S|$ and $b = |V_j \cap S|$. Select some vertex $u \in V_i \cap S$ and a vertex $w \in V_j - S$. Let $S' = S \cup \{w\} - \{u\}$, and let F' be the 2-cluster editing set that corresponds to $(S', V'-S')$. We will show that $|F| - |F'| \geq 0$.

Note that if $\{i,j,l\} \neq \{i',j',l'\}$ then the edges of $E_{i,j,l}$ are incident on different vertices than the edges of $E_{i',j',l'}$. Therefore, a vertex v in cluster V_i has at most one neighbor outside of V_i. If such a neighbor exists, denote it by n_v.

The edges in F that are incident on u or w are:

1. $M-a$ edges (that are in E') between u and $V_i - S$.
2. A possible edge (in E') between u and n_u (if n_u exists and $n_u \in V'-S$).
3. Either $|S|-a$ or $|S|-a-1$ edges (that are not in E') between u and $S-(V_i \cap S)$ (the second term is for the case when n_u exists and $n_u \in S$).
4. b edges (in E') between w and $V_j \cap S$.
5. A possible edge (in E') between w and n_w (if n_w exists and $n_w \in S$).
6. Either $nM - |S| - (M-b)$ or $nM - |S| - (M-b) - 1$ edges (not in E') between w and $V' - S - (V_j - S)$ (the second term is for the case when n_w exists and $n_w \in V'-S$).

The total number of these edges is at least $nM - 2a + 2b - 2$. Similarly, the edges in F' that are incident on u or w are:

1. $a - 1$ edges (in E') between u and $V_i \cap S - \{u\}$.
2. A possible edge (in E') between u and n_u.
3. Either $nM - |S| - (M - a) - 1$ or $nM - |S| - (M - a) - 2$ edges (not in E') between u and $V' - S - (V_i - S) - \{w\}$.
4. $M - b - 1$ edges (in E') between w and $V_j - S - \{w\}$.
5. A possible edge (in E') between w and n_w.
6. Either $|S| - b - 1$ or $|S| - b - 2$ edges (not in E') between w and $S - (V_j \cap S) - \{u\}$.

The total number of these edges is at most $nM + 2a - 2b - 2$. It follows that

$$|F| - |F'| = (nM - 2a + 2b - 2) - (nM + 2a - 2b - 2) = 4(b - a) \geq 0.$$

If $a < b$, we have that $|F'| < |F|$, a contradiction to the minimality of F. In the case when $a = b$, namely the value of $|V_l \cap S|$ is equal amongst all the clusters, we have that $|F'| = |F|$. In that case, we build a set S'' from S' using the same process as above, and since $|V_l \cap S'|$ is not equal amongst all the clusters, it follows that the 2-cluster editing set F'' that corresponds to S'' satisfies $|F''| < |F'| = |F|$, and again we have a contradiction.

Now suppose that the cut $(S, V' - S)$ splits exactly one cluster, and denote this cluster by V_i. Let $a = |V_i \cap S|$. Out of the other $n - 1$ clusters, suppose that r clusters are contained in S, and $n - r - 1$ clusters are contained in $V' - S$. W.l.o.g. suppose that $n - r - 1 \leq r$, and since n is even we have $n - r - 1 \leq r - 1$. We build a set $S' = S - V_i$, and let F' be the corresponding 2-cluster editing set. For each $v \in V_i \cap S$, there are at least $rM - 1$ edges in F between v and $S - V_i$ (the term -1 is due to the possibility that n_v exists and $n_v \in S - V_i$), and $M - a$ edges between v and $V_i - S$, so the number of edges in F that are incident on v is at least $rM - 1 + M - a$. Furthermore, an edge in F' that is incident on v is either between v and n_v, or between v and $V' - S - V_i$. The number of edges of the latter type is $(n - r - 1)M$, so the number of edges in F that are incident on v is at most $(n - r - 1)M + 1 \leq (r - 1)M + 1$. It follows that

$$|F| - |F'| \geq a\,(rM - 1 + M - a - ((r - 1)M + 1)) = a\,(2M - a - 2) > 0,$$

and again, we get a contradiction to the minimality of F. We conclude that F does not split any cluster.

We now claim that the number of clusters that are contained in S is exactly $n/2$. Conversely, suppose w.l.o.g. that it is $r > n/2$. Let V_i be some cluster

contained in S. Let $S' = S - V_i$ and let F' be the corresponding 2-cluster editing set. Similarly to above, we have that

$$|F| - |F'| \geq M((r-1)M - 1 - ((n-r)M + 1)) \geq M(M-2) > 0,$$

a contradiction. Hence, S contains $n/2$ clusters.

Define a coloring $f\colon V \to \{0,1\}$ by $f(i) = 0$ iff $V_i \subseteq S$. By the argument above, f is balanced. It remains to show that f is a legal 2-coloring. For a hyperedge $(i,j,k) \in E$, if i,j,k have the same color then $|F \cap E_{i,j,l}| = 3$. Otherwise, $|F \cap E_{i,j,l}| = 1$ since two of the three edges must connect vertices in clusters on different sides of the cut $(S, V' - S)$. Hence, each unicolored hyperedge adds two to the size of F. By the first direction of the proof, for a legal 2-coloring, the corresponding editing set is of size exactly k, and thus no unicolored hyperedge is possible in f. It follows that f is a balanced 2-coloring of G. ■

Corollary 4.7.3. *m-cluster editing is NP-hard for every $m \geq 2$.*

Proof. Fix $m > 2$. We provide a reduction from 2-cluster editing to m-cluster editing. Given an input graph $G = (V,E)$ to the 2-cluster editing problem where $V = \{v_1, \ldots, v_n\}$, we build a graph $G' = (V', E')$ as follows: $V' = V \cup \cup_{i=1}^{m-2} V_i$ where $V_i = \{w_{i,j} : j = 1, \ldots, n^2\}$. The edges of G' include all the edges in E and a clique on each V_i. k is unchanged.

To prove the correctness of the reduction, we show that there is a 2-cluster editing set of G of size at most k iff there is an m-cluster editing set of G' of size at most k. The 'only if' part of the claim is trivial as every 2-cluster editing set of G is an m-cluster editing set of G'. For the other direction, suppose that F' is an m-cluster editing set of G' of size at most k, and let $(S_1, \ldots, S_m)$ be the corresponding m-cut.

If there is a set V_i such that $V_i \cap S_j \neq \phi$ and $V_i \not\subseteq S_j$ for some j, then F' contains an edge between every vertex from $V_i \cap S_j$ and a vertex from $V_i - S_j$. The number of such edges is at least $n^2 - 1 > k$, a contradiction. Therefore, every set V_i is contained in some set S_j. Furthermore, every set S_j contains at most one set V_i because otherwise, if a set S_j contains both V_i and $V_{i'}$ then F' contains an edge between every vertex from V_i and a vertex from $V_{i'}$, so $|F'| \geq n^4 > k$, a contradiction. It follows that all the edges in F' are incident on vertices of V, which implies that F' is a 2-cluster editing set of G. ■

List of Symbols

Symbol	Meaning	Introduced on page
Δ	$p - r$	47
Γ	$\max_i \lvert\frac{\lvert V_i\rvert}{n} - \frac{1}{m}\rvert$	55
A_i	$\lvert V_i\rvert$	55
a_i	$\lvert V_i\rvert/n$	55
b_i^t	$\mathrm{I}(V_i, L_t, R_t)$	67
$c_i(f)$	$2a_i \sum_{j\neq i} a_j(p_{ij}^{>}(f) - p_{ij}^{<}(f))$	61
c_i^0	$c_i(d_{\{u\}})$	68
c_i^t	$c_i(d_{L_t} - d_{R_t})$	68
$d_S(v)$	Number of neighbors of v in S	50
$\mathrm{I}(V_i, L, R)$	$\frac{\lvert V_i\cap L\rvert - \lvert V_i\cap R\rvert}{\lvert L\rvert}$	60
K	$\max_i \lvert V_i\rvert$	73
k	$\min_i \lvert V_i\rvert$	47
m	Number of clusters	47
n	Number of vertices	47
p	Probability for an edge between vertices of same cluster	47
$p_{ij}^{>}(f)$	$\mathrm{P}\left[f(v) > f(w) \mid v \in V_i, w \in V_j\right]$	61
r	Probability for an edge between vertices of different clusters	47
r_i^t	b_i^t/b_1^t	67
V_i	i-th underlying cluster	47

Chapter 5

Sequencing By Hybridization

5.1 Introduction

A DNA molecule can be viewed as a sequence over the alphabet $\{A, C, G, T\}$ which corresponds to the four types of nucleotides. One of the main current endeavors in Life Sciences and Medicine is efficient sequencing of very long DNA molecules. The prevalent sequencing technologies are currently gel-based. Sequencing by Hybridization (SBH) [13, 89] was proposed in the late Eighties as an alternative way to DNA sequencing. In this method, the target sequence is hybridized to a universal chip containing all 4^k sequences of length k. For each k-long sequence (or probe) in the chip, if its reverse complement appears in the target, then the two sequences will bind (or hybridize), and this hybridization can be detected (the reverse complement of a sequence S is obtained by reading the sequence backwards and replacing each letter in S by its Watson-Crick complement: The complements of A,C,G and T are T,G,C and A, respectively). Thus, from the above experiment one can obtain the multi-set of all k-long subsequences of a target sequence (all subsequences referred to in this chapter will be contiguous). This multi-set is called the *k-spectrum* of the target, or simply its *spectrum*. We note that in reality, only the *set* of all k-subsequences of the target can be obtained, but many studies on SBH (including this work) assume that the multi-set is known, as this assumption simplifies the analysis. This assumption is justified since the missing multiplicity data in the hybridization result can be considered as false negative errors. Pevzner has shown that reconstructing a sequence from its spectrum is polynomial [105]. Since copies of the universal chip can be economically produced and used to sequence any DNA target, and the computational reconstruction task is efficiently handled, this seems as a promising alternative to standard sequencing techniques.

Unfortunately, sequence reconstruction is often not unique: Other sequences can have the same spectrum as the target's. Therefore, we are interested in telling how likely this event is as a function of the length of the target. We say

that a sequence is *uniquely recoverable* from its spectrum if there is no other sequence with the same spectrum. By assuming a distribution on the sequences of a certain length, one can compute the probability that a random sequence is not uniquely recoverable from its k-spectrum. We refer to this as the *failure probability*. Denote the failure probability for the uniform distribution over N-long sequences by $P(n,k)$, where $n = N - k + 1$ (n is the number of k-tuples in the sequence). An asymptotically exact formula for $P(n,k)$ was given by Dyer et al. [49] and Arratia et al. [12].

The main shortcoming of classical SBH is ambiguous solutions: The maximum length $n = n(k,\epsilon)$ for which $P(n,k) \leq \epsilon$, is rather small. For example, $n(8, 0.1)$ is about 200 [107]. Roughly, $n(k, 0.1)$ is about 2^k, or in other words, to reconstruct a sequence of length n, a chip with $\Theta(n^2)$ probes is needed. Several methods for overcoming this limitation were proposed:

- Interactive protocols: Margaritis and Skiena [92] suggested performing SBH in rounds, where the chip in each round depends on the results of the previous rounds. Margaritis and Skiena gave a scheme for reconstructing sequences of length n in $O(\log n)$ rounds, where the number of probes in each round is $O(n)$. Frieze and Halldórsson [55] gave an improved scheme which uses only $O(1)$ rounds while still using $O(n)$ probes in each round.

- Alternative chip design: Pevzner et al. [107] suggested using gaps in the probes that are matched to any of the four bases. For example, the probe A**G matches the sequences TAAC, TACC, TAGC, etc. A gapped probed can be implemented using universal bases such as 5-nitroindole [85]. Preparata et al. [110] gave a gapped probe design such that sequences of length n can be reconstructed using a chip with $O(n)$ probes. A more efficient algorithm for reconstructing the sequence from the spectrum was given in [111]. Halperin et al. [62] gave a randomized gapped chip design that is robust to hybridization errors (their results are discussed in detail later).

- Positional SBH: Adleman [1] and Broude et al. [30] suggested methods that allow measuring for each k-tuple in the spectrum, an approximate position of the tuple in the target sequence. The complexity of reconstructing a sequence from the data was studied in [15, 63].

- Using a known homologous sequence: Pe'er and Shamir [102] studied the case where a sequence which is very similar to the target sequence is known in advance. Pe'er and Shamir gave an algorithm for this problem, and showed using simulations, that the algorithm can perfectly reconstruct a sequence of length 2000 bp given its 8-spectrum and a sequence that is 97% identical to the target sequence.

Currently, SBH is not considered competitive in comparison with standard gel-based sequencing technologies.

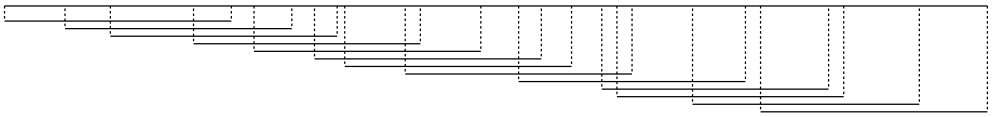

Figure 5.1: Partitioning a target sequence into IFs. The target (top) is partitioned into IFs by the endpoints of the clones (bottom). There are 13 clones and 25 IFs.

Drmanac et al. [46] proposed the following enhancement to SBH (compare Figure 5.1): Instead of reconstructing a single target sequence from its spectrum, one can obtain the spectra of many short overlapping fragments (clones) of the target. The larger the overlap between two clones, the more similar their spectra would be. Using this similarity one can infer the position of clones along the target. Moreover, the endpoints of the clones induce a partition of the target into even shorter subfragments, and the spectrum of each of those can be computationally inferred. The DNA stretches between consecutive clone endpoints are called *information subfragments* (IFs). By obtaining clones at high redundancy, the average length of an IF would be short enough to be uniquely recoverable from its spectrum with high probability. When all IFs are uniquely recoverable, so is the complete target. Drmanac et al. suggested sequencing a 10^6 bp long target by obtaining the spectra of 25,000 500 bp-long clones. They provided some simple computational arguments and performed error-free simulations to support their suggestion.

A fundamental limitation of SBH is that the hybridization experiments are error prone. In a *false positive* error, a certain k-tuple appears in the experimental spectrum while in fact it does not appear in the target. The converse occurs in a *false negative* error. The problem of reconstructing the sequence when there are errors is NP-hard [56]. However, several heuristics were proposed [20, 21, 44, 83, 105]. Halperin et al. [62] gave an algorithm that can reconstruct a sequence of length $O(2^{(1-3p)k})$ from a k-spectrum with false and positive errors, where p is the probability of a false negative error (and the probability of a false positive error is small). They also gave a gapped probe design such that sequences of length n can be reconstructed using a chip with $O((1-p)^2 n \log^2 n)$ probes (again, assuming that the probability of a false positive error is small). An algorithm for dealing with errors in interactive SBH was given in [109].

5.1.1 New results

We expand on the analysis of the Drmanac et al. strategy and of Arratia et al. in several ways. First, we improve the estimate of Arratia et al. on $P(n,k)$ for small (and realistic) values of k: We show that if $5(k-1) \leq n \leq 2^{k+1} + 4(k-2)$, then $P(n,k) = \Theta(n^4/4^{2k})$. We then use this result in order to investigate the Drmanac et al. strategy in the presence of hybridization errors: Under some simplifying

assumptions which will be described below, we prove that the introduction of false negative errors has a very small effect on the probability of unique recoverability. More precisely, the ratio between the failure probability in the presence of errors, and in the errorless case, is $1+O(p/(1-p)^4 \cdot 1/d)$, where d is the average IF length, and p is the probability of a false negative error. We also perform simulations with real DNA sequences which show that the technique can reconstruct a target longer than 30kb from 8-mer spectra containing 50% false negative errors, with an average of less than one miscalled base in 1000 bases. The results in this chapter were published in [120] and [122].

We need some notation in order to specify our assumptions: Denote the target sequence by A. One first clones many short random subfragments $C_1, \ldots, C_c$ of A, and obtains the k-spectrum of each clone. We assume that the clones completely cover A. The endpoints of the c clones form a partition of the target A into IFs $J_1, \ldots, J_l$ where $l \leq 2c-1$. We assume that the positions of the clones along the target have already been inferred from the spectra. In the absence of errors, if a k-tuple P is located in the IF J_i, then P will appear in the k-spectrum of each clone C_j that contains J_i. Using this observation, one can compute the spectrum of each IF.

By increasing the hybridization stringency, the number of false positives can be decreased, at the expense of increasing the false negatives rate, which as we shall show, has little effect on the success of the strategy. Moreover, in our model, a false positive error that appears in the spectrum of some clone is unlikely to appear in the spectra of the intersecting clones, while a real spectrum element is likely to appear in many of the spectra of the intersecting clones. Therefore, a simple procedure can be used to remove such false positives, perhaps at the expense of increasing the false negative rate. Therefore, we will assume that there are no false positive errors.

For false negative errors, our probabilistic model assumes that each k-tuple contained in some clone does not appear in its (experimental) spectrum with probability p, independently of the other k-tuples in the clone and of its appearance in other clones. False negative errors have two effects. First, it is possible that some k-tuple P in the target sequence of A will not appear in the spectra of any of the clones that contain P. Therefore, P will be missing from the spectrum which is built for A. As we assume that the clones cover the target with high redundancy, this effect has very low probability: If every point in the target is covered by at least l clones, then that probability is at most np^l. The second type of effect is that the k-tuple P may appear in some, but not all, the spectra of clones that contain it. Thus, when we decide which IF contains P, we may be wrong. This is the effect that we shall study in detail.

For the theoretical analysis we make several simplifying assumptions: First, we assume that clone positions are not random, but are spread uniformly across the sequence, and we denote by d the distance between the left endpoints of

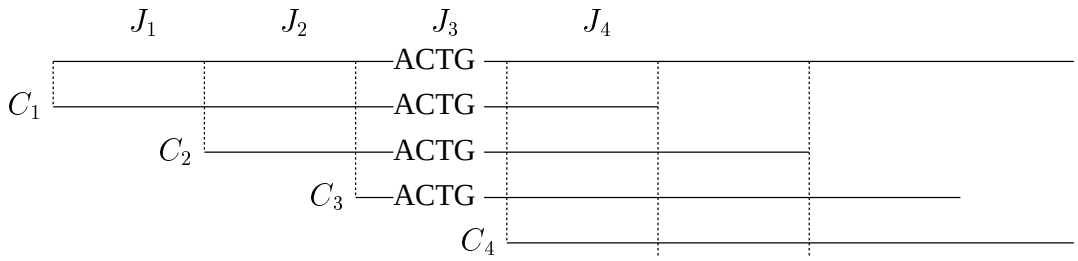

Figure 5.2: An example of attributing k-tuples to IFs. If there are no errors, then the 4-tuple ACTG appears in clones C_1, C_2, C_3, so we attribute it to IF J_3. If there are false negatives, ACTG can appear, for example, only in C_1, C_2, and therefore it will be mistakenly attributed to IF J_2. As the mistake is only in one direction, we know in this case that the k-tuple ACTG appears in $\cup_{i \geq 2} J_i$.

two consecutive clones. Second, for the partition into IFs, we ignore the right endpoints of the clones, and only consider the partition of the sequence A derived from the left endpoints of the clones. (Note that this assumption generates longer and fewer IFs than there really are, so removing this assumption would only improve the results). Technically, to achieve this we assume that the length of the clones is divided by d. Thus, the right endpoint of a clone is a left endpoint of some other clone, except for the last few clones. This partition forms c IFs $J_1, \ldots, J_c$. When we consider some k-tuple P in the spectrum of A, we attribute P to the fragment J_i where i is the maximum index of a clone for which P appears in its spectrum (for simplicity of representation we assumed here that P appears only in one IF). Since we assumed only false negative errors, the index i is always less than or equal to the correct index i'. So in the case of errors, instead of knowing that P is in fragment $J_{i'}$, we know that P is in the union $\cup_{j \geq i} J_j$. Moreover, the value of $i' - i$ is a random variable with geometric distribution with parameter p. See Figure 5.2 for an example of the situation described above.

The rest of this chapter is organized as follows: Section 5.2 contains problem definitions and preliminaries. In Section 5.3 we give the upper and lower bounds on the failure probability with no errors. The main result of this paper is given in Section 5.4. Finally, our simulations are described in Section 5.5.

5.2 Preliminaries

For a sequence $A = a_1 \cdots a_r$, let $A|_i^l$ denote the l-subsequence $a_i a_{i+1} \cdots a_{i+l-1}$. For two sequences $A = a_1 \cdots a_r$ and $B = b_1 \cdots b_s$, we denote $A|B$ if the $(s-1)$-suffix of A is equal to the $(s-1)$-prefix of B. If $A|B$, we denote by $A \diamond B$ the sequence $a_1 \cdots a_r b_s$.

The k-spectrum of a sequence A of length N is the multi-set of all the k-

subsequences of A, and is denoted by $\mathrm{SP}^k(A)$. The SBH problem is: Given a multi-set M of k-tuples, find a sequence A for which $\mathrm{SP}^k(A) = M$, if there is such a sequence. A sequence A is called *uniquely recoverable w.r.t. k* if there is no sequence $A' \neq A$ such that $\mathrm{SP}^k(A) = \mathrm{SP}^k(A')$.

An alternative way to define the SBH problem is as follows: Given a multi-set $M = \{A_1, \ldots, A_n\}$ of k-tuples, find a permutation $\pi\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$ for which $A_{\pi(i)}|A_{\pi(i+1)}$ for all $i < n$. For such a permutation π, define the sequence $A^\pi = A_{\pi(1)} \diamond A_{\pi(2)} \diamond \cdots \diamond A_{\pi(n)}$, and note that $\mathrm{SP}^k(A^\pi) = M$. Pevzner [105] gave a formulation of the SBH problem using graphs: For a multi-set M, define the *de-Bruijn* graph G_M to be a directed graph whose vertices are all the distinct $(k-1)$-tuples that appear in M, i.e. $\cup_{i=1}^n \{A_i|_1^{k-1}, A_i|_2^{k-1}\}$, and whose edges are $e_i = (A_i|_1^{k-1}, A_i|_2^{k-1})$ for $i = 1, \ldots, n$. A permutation π is a solution to the SBH problem iff $P_\pi = [e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(n)}]$ is an Eulerian path in G_M. Therefore, the SBH problem is polynomial.

The *Positional SBH problem* (PSBH) is defined as follows: Given a multi-set $M = \{A_1, \ldots, A_n\}$ and sets $S(i) \subseteq \{1, \ldots, n\}$ for all $i \leq n$, find a permutation π for which $A_{\pi(i)}|A_{\pi(i+1)}$ for all $i < n$, and $i \in S(\pi(i))$ for all $i \leq n$ (namely, the set $S(j)$ contains the positions in which the k-tuple A_j can appear). Such a permutation π is called a *solution* of the instance M, k, S. For example, suppose that $n = 3$, $A_1 = \mathrm{ACT}$, $A_2 = \mathrm{CTA}$, $A_3 = \mathrm{TAC}$, $S(1) = \{1, 3\}$, $S(2) = \{1, 2\}$, and $S(3) = \{2, 3\}$. Both $\pi_1(1), \ldots, \pi_1(3) = 1, 2, 3$ and $\pi_2(1), \ldots, \pi_2(3) = 2, 3, 1$ are solutions and $A^{\pi_1} = \mathrm{ACTAC}, A^{\pi_2} = \mathrm{CTACT}$. However, $\pi_3(1), \ldots, \pi_3(3) = 3, 1, 2$ is not a solution as $1 \notin S(3) = S(\pi_3(1))$. The PSBH problem is NP-hard even when $|S(i)| \leq 3$ for all i [15], while it is polynomial when $|S(i)| \leq 2$ for all i [15], or when each $S(i)$ is an interval of length $O(\log n)$ [63].

It is more convenient to denote an instance of PSBH by A, k, S where A is a sequence of length $N = n + k - 1$, which naturally defines the multi-set $M = \mathrm{SP}^k(A)$. A solution π of A, k, S is called *trivial* if $A^\pi = A$. A sequence A is called *uniquely recoverable w.r.t. k, S* if there is no nontrivial solution π of the instance A, k, S.

In this paper, we consider two special cases of the Positional SBH problem. Let $I = \{J_1, \ldots, J_c\}$ be a partition of the interval $[1, n]$ into disjoint intervals (i.e., there are integers $s_1 = 1, s_2, \ldots, s_c, s_{c+1} = n+1$ such that $J_i = [s_i, s_{i+1} - 1]$). For $i \leq n$, let $I(i)$ be the index such that $i \in J_{I(i)}$. An instance of the *Interval PSBH problem* (IPSBH), denoted by A, k, I, is an instance A, k, S_I of PSBH, where $S_I(i) = J_{I(i)}$ for $i \leq n$. A sequence A is called *uniquely recoverable w.r.t. k, I* if there is no nontrivial solution of the instance A, k, I. The instance A, k, S_I is equivalent to the instance A, k, S'_I, where $S'_I(i) = [s_{I(i)}, n]$. (The proof is simple: Suppose that π is a solution of A, k, S'_I. For $i \in J_1 = [1, s_2 - 1]$, we have $i \in S'_I(\pi(i)) = [s_{I(\pi(i))}, n]$, hence $I(\pi(i)) = 1$ which implies that $\pi(i) \in J_1$. Now, for $i \in J_2 = [s_2, s_3 - 1]$, we have $i \in [s_{I(\pi(i))}, n]$, so $I(\pi(i)) \leq 2$. But $\pi(j) \in J_1$ for all $j \in J_1$, so it follows that $\pi(i) \notin J_1$. Therefore, $\pi(i) \in J_2$, or in other words,

$I(\pi(i)) = 2$. By repeating the same argument, we conclude $I(\pi(i)) = I(i)$ for all i. Therefore, $i \in J_{I(i)} = S_I(\pi(i))$ for all i, namely π is a solution of A, k, S'_I. Conversely, if π is a solution of A, k, S_I then it is also a solution of A, k, S'_I as $S_I(i) \subseteq S'_I(i)$ for all i.)

Let $\Delta = (\Delta_1, \ldots, \Delta_n)$ be a vector of nonnegative integers. An instance of the *Inexact Interval PSBH problem* (IIPSBH), denoted by A, k, I, Δ, is an instance $A, k, S_{I,\Delta}$ of PSBH, where $S_{I,\Delta}(i) = [s_{\max(I(i)-\Delta_i,1)}, n]$. A sequence A is called *uniquely recoverable w.r.t.* k, I, Δ if there is no nontrivial solution of the instance A, k, I, Δ.

For the rest of this paper we assume that $n = cd$ for some integers c and d. Let I_d be the set of intervals $\{[1, d], [d + 1, 2d], \ldots, [(c - 1)d + 1, cd]\}$ (so $I(i) = \lceil i/d \rceil$). We denote by $P(n, k, d)$ the probability that for a random sequence A of length $n + k - 1$, A is not uniquely recoverable w.r.t. k, I_d. We also denote by $P(n, k, d, p)$ the probability that for a random sequence A of length $n + k - 1$ and for a vector $\Delta = (\Delta_1, \ldots, \Delta_n)$ of independent identically distributed random variables with geometric distribution with parameter p, A is not uniquely recoverable w.r.t. k, I_d, Δ. The main result of this paper is showing that $P(n, k, d, p) = (1 + O(c_p/d))P(n, k, d)$, where c_p is a term that depends on p.

Let A be a sequence of length $N = n + k - 1$. A pair (i, j) is called a *repeat* if $A|_i^{k-1} = A|_j^{k-1}$. A repeat (i, j) is called *rightmost* if $j \neq n + 1$ and $(i + 1, j + 1)$ is not a repeat (i.e., if $a_{i+k-1} \neq a_{j+k-1}$). (In the de-Bruijn graph, a repeat is manifest by two edges emanating from the same vertex. In a rightmost repeat, the two edges enter distinct vertices.) A repeat (i, j) is called *weakly rightmost* if either $j = dI(i)+1$, or (i, j) is rightmost. For example, let $k = 4$, $d = n = 11$, and $A =$ AGCTT ACGCT TCTT. The repeats of A are $(2, 8)$, $(3, 9)$, $(9, 12)$, $(3, 12)$, the weakly rightmost repeats are $(3, 9)$, $(9, 12)$, $(3, 12)$, and the single rightmost repeat is $(3, 9)$.

A pair of repeats $((i, j), (i', j'))$ is called *R-pair* if (i, j) is rightmost, and it is called *Rr-pair* if (i, j) is rightmost, and (i', j') is weakly rightmost. The pair is called *interleaved* if $i \leq i' < j < j'$ and $I(i) = I(j' - 1)$.

5.3 Estimating the failure probability

In this section we show that $P(n, k, d) = \Theta(d^3 n/4^{2k})$. We note that an asymptotically exact formula for $P(n, k)$ was given in [12, 49], but it does not give a good estimate for small values of n and k.

We begin with showing an upper bound on $P(n, k, d)$. A necessary and sufficient condition for unique recoverability w.r.t. k was given by Arratia et al. [12] (based on result from [106]). In the following theorem, we give a more general necessary and sufficient condition for unique recoverability w.r.t. k, I_d. We also note that our characterization is simpler than the one in [12].

Theorem 5.3.1. *A sequence A is not uniquely recoverable w.r.t. k, I_d iff either (1) A contains an interleaved R-pair, or (2) $A|_1^{k-1} = A|_{d+1}^{k-1} = \cdots = A|_{cd+1}^{k-1}$ and there are indices $i_1, \ldots, i_c$ with $(l-1)d+1 < i_l < ld+1$, and $A|_{i_1}^{k-1} = A|_{i_2}^{k-1} = \cdots = A|_{i_c}^{k-1} \neq A|_1^{k-1}$.*

Proof. If $((i,j),(i',j'))$ is an interleaved R-pair, then we have $A|_{i-1}^k \mid A|_j^k$ (if $i > 1$), $A|_{j-1}^k \mid A|_i^k$, $A|_{i'-1}^k \mid A|_{j'}^k$ (if $j' < n+1$), and $A|_{j'-1}^k \mid A|_{i'}^k$. Thus, we define a permutation π as follows: $\pi(1), \ldots, \pi(n) =$

$$1, 2, \ldots, i-1, j, j+1, \ldots, j'-1, i', i'+1, \ldots, j-1, i, i+1, \ldots, i'-1, j', j'+1, \ldots, n$$

and it is easy to verify that π is a solution of A, k, I_d. Furthermore, $A|_i^k \neq A|_j^k = A^\pi|_i^k$ (as (i,j) is a rightmost repeat), so $A \neq A^\pi$. Therefore, A is not uniquely recoverable w.r.t. k, I.

Suppose that A satisfies case (2) of the theorem with indices $i_1, \ldots, i_c$. We define a permutation π as follows: $\pi(1), \ldots, \pi(n) =$

$$i_1, i_1+1, \ldots, d, 1, 2, \ldots, i_1-1, i_2, i_2+1, \ldots, 2d, d+1, d+2, \ldots, i_2-1, i_3, \ldots, i_c-1.$$

The conditions of case (2) imply that π is a solution of A, k, I_d and $A \neq A^\pi$.

We now prove the second direction of the theorem. Suppose that A is not uniquely recoverable w.r.t. k, I_d, and let π be a nontrivial solution. To simplify the proof, we use the de-Bruijn graph $G = (V, E)$ of $\mathrm{SP}^k(A)$. Denote $E = \{e_1, \ldots, e_n\}$ where $e_i = (A|_i^{k-1}, A|_{i+1}^{k-1})$. For two edges e_i, e_j we write $e_i \equiv e_j$ if e_i and e_j are parallel edges, namely if $A|_i^k = A|_j^k$. Clearly, both $P = [e_1, e_2, \ldots, e_n]$ and $P' = [e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(n)}]$ are Eulerian paths in G. We also define subgraphs $G_l = (V, E_l)$ of G, where $E_l = \{e_{(l-1)d+1}, \ldots, e_{ld}\}$. Again, both $P_l = [e_{(l-1)d+1}, \ldots, e_{ld}]$ and $P'_l = [e_{\pi((l-1)d+1)}, \ldots, e_{\pi(ld)}]$ are Eulerian paths in G_l.

The proof is based on comparison of the paths P and P'. We will show that if the paths P and P' start from the same vertex (i.e. $A^\pi|_1^{k-1} = A|_1^{k-1}$) then case (1) happens, and otherwise case (2) happens.

We first consider the case when P and P' start from the same vertex. Let i be the minimum index such that $e_i \not\equiv e_{\pi(i)}$ and let $j = \pi(i)$. W.l.o.g. we assume that $\pi(l) = l$ for all $l < i$. (If π doesn't satisfy this requirement, then let l be the minimum index for which $l \neq \pi(l)$. Define a permutation π' by $\pi'(l) = l$, $\pi'(\pi^{-1}(l)) = \pi(l)$, and $\pi'(l') = \pi(l')$ for any $l' \neq l, \pi^{-1}(l)$. π' is a solution as $e_l \equiv e_{\pi(l)}$. This process can be repeated until we obtain a solution that satisfies the requirement.) Thus $\pi(i) \notin \{1, \ldots, i\}$, so $i < j$. From the minimality of i, and the assumption that P and P' start from the same vertex, it follows that the edges e_i and e_j have the same start vertex, so $A|_i^{k-1} = A|_j^{k-1}$. As $e_i \not\equiv e_j$, the edges e_i and e_j has different end vertices, hence $A|_{i+1}^{k-1} \neq A|_{j+1}^{k-1}$. Therefore, (i,j) is a rightmost repeat.

As π is a solution we have that $I(i) = I(j)$. The edges $e_i, e_{i+1}, \ldots, e_{j-1}$ form a cycle in $G_{I(i)}$, which we denote by C, and we denote by V_C the vertices of C. As $P'_{I(i)}$ is an Eulerian path, it must pass through all the edges of C, and let l be the minimum index such that $e_{\pi(l)}$ is in C. By definition, the edge $e_{\pi(i)} = e_j$ is not in C (though it can be parallel to an edge from C), and therefore, $l > i$. Since the end vertex of $e_{\pi(l-1)}$ is equal to the start vertex of $e_{\pi(l)}$, we have that $A|_{\pi(l)}^{k-1} = A|_{\pi(l-1)+1}^{k-1}$, namely $(\pi(l), \pi(l-1)+1)$ is a repeat. Clearly, $i \leq \pi(l) \leq j-1$, $\pi(l-1) \geq j$ (as the edge $e_{\pi(l-1)}$ is not in C), and $I(\pi(l-1)) = I(i)$ (as the edge $e_{\pi(l-1)}$ is in $G_{I(i)}$). Therefore, $((i,j),(\pi(l),\pi(l-1)+1))$ is an interleaved R-pair.

We now consider the case when P and P' start from different vertices (P starts from $A|_1^{k-1}$, and P' starts from $A^\pi|_1^{k-1}$). Since P'_1 is an Eulerian path in G_1 which doesn't start from vertex $A|_1^{k-1}$, it follows that the vertex $A|_1^{k-1}$ has equal in and out degrees in G_1. Since P_1 is an Eulerian path in G_1 which starts from the vertex $A|_1^{k-1}$, it must also end in $A|_1^{k-1}$, hence $A|_1^{k-1} = A|_{d+1}^{k-1}$. With similar arguments, we obtain that vertex $A^\pi|_1^{k-1}$ has equal in and out degrees in G_1 (as P_1 doesn't start from $A^\pi|_1^{k-1}$), and therefore P'_1 ends in $A^\pi|_1^{k-1}$. Thus, $A^\pi|_1^{k-1} = A^\pi|_{d+1}^{k-1}$.

Now, P'_2 is an Eulerian path in G_2 which doesn't start from vertex $A|_{d+1}^{k-1}$ (P'_2 starts from $A^\pi|_{d+1}^{k-1} = A^\pi|_1^{k-1}$) and therefore vertex $A|_{d+1}^{k-1}$ has equal in and out degrees in G_2. Again, it follows that $A|_{d+1}^{k-1} = A|_{2d+1}^{k-1}$. We also obtain that $A^\pi|_{d+1}^{k-1}$ has equal in and out degrees in G_2 and $A^\pi|_{d+1}^{k-1} = A^\pi|_{2d+1}^{k-1}$.

By repeating the same arguments, we obtain that $A|_1^{k-1} = A|_{d+1}^{k-1} = \cdots = A|_{cd+1}^{k-1}$ and $A^\pi|_1^{k-1} = A^\pi|_{d+1}^{k-1} = \cdots = A^\pi|_{cd+1}^{k-1}$. We therefore define $i_l = \pi((l-1)d+1)$ for $l = 1, \ldots, c$, and A satisfies the conditions of case (2) with the indices $i_1, \ldots, i_c$. ∎

For bounding the failure probability, we will use a slightly different characterization:

Theorem 5.3.2. *A sequence A is not uniquely recoverable w.r.t. k, I_d iff either A contains an interleaved Rr-pair, or case (2) of Theorem 5.3.1 happens.*

Proof. It suffices to prove that if A contains an interleaved R-pair, then it also contains an interleaved Rr-pair. Let $((i,j),(i',j'))$ be an interleaved R-pair. If (i',j') is weakly rightmost, then we are done. Otherwise, $j' \neq dI(i')+1$ and (i',j') is not rightmost. It follows that $I(j') = I(j'-1) = I(i)$ and $j' \leq n$. As (i',j') is not rightmost and $j' \neq n+1$, we have that $(i'+1, j'+1)$ is a repeat. If $i' < j-1$ then let $i'_2 = i'+1$ and $j'_2 = j'+1$. Otherwise ($i' = j-1$), let $i'_2 = i$ and $j'_2 = j'+1$ (note that (i'_2, j'_2) is a repeat as (i,j) and $(j, j'+1)$ are repeats). In both cases, $((i,j),(i'_2,j'_2))$ is an interleaved R-pair. We repeat this process until we reach a pair $((i,j),(i'_r,j'_r))$ which is an interleaved Rr-pair. ∎

By Theorem 5.3.2, $P(n,k,d)$ is less than or equal to the probability that there is an interleaved Rr-pair, plus the probability that case (2) happens. The latter probability is less than $1/4^{(k-1)n/d}$. Let $P_{i,j,i',j'}$ denote the probability that $((i,j),(i',j'))$ is an interleaved Rr-pair.

Lemma 5.3.3. *For all $i \leq i' < j < j'$, $P_{i,j,i',j'} \in \{0, 9/4^{2k}\}$ for $j' < dI(i)+1$, and $P_{i,j,i',j'} \in \{0, 12/4^{2k}\}$ for $j' = dI(i)+1$.*

Proof. We only prove the case $j < dI(i)+1$, as the proof of the second case is similar. Let $a_1, \ldots, a_N$ denote the letters of the sequence A. By definition,

$$P_{i,j,i',j'} = \mathrm{P}\left[\begin{array}{c} a_i = a_j, a_{i+1} = a_{j+1}, \ldots, a_{i+k-2} = a_{j+k-2}, a_{i+k-1} \neq a_{j+k-1}, \\ a_{i'} = a_{j'}, a_{i'+1} = a_{j'+1}, \ldots, a_{i'+k-2} = a_{j'+k-2}, a_{i'+k-1} \neq a_{j'+k-1} \end{array}\right].$$

For the proof of the lemma we build a graph $G_{i,i',j,j'}$. The vertices of $G_{i,i',j,j'}$ are the indices of the letters that appear in the above equalities and inequalities, and the edges correspond to the equalities. Formally, the vertices of $G_{i,i',j,j'}$ are $\{i, \ldots, i+k-1\} \cup \{j, \ldots, j+k-1\} \cup \{i', \ldots, i'+k-1\} \cup \{j', \ldots, j'+k-1\}$ and its edges are $\{(i+r, j+r) | r = 0, \ldots, k-2\} \cup \{(i'+r, j'+r) | r = 0, \ldots, k-2\}$. Let $V_1, \ldots, V_b$ be the connected components of $G_{i,i',j,j'}$, and let n_l and m_l denote the number of vertices and edges in V_l, respectively. The pairs (i,j) and (i',j') are repeats iff for each connected component V_l, all the corresponding letters in A are equal. The probability of this event is exactly $\prod_{l=1}^{b} (1/4)^{n_l - 1}$.

We consider three cases. In case 1, we assume that $G_{i,i',j,j'}$ contains parallel edges, which implies that $(i+r_1, j+r_1) = (i'+r_2, j+r_2)$ for $r_1, r_2 \in \{0, \ldots, k-2\}$ where $r_1 > r_2$. Therefore, $i' - i = j' - j = r$ for some $r \in \{1, \ldots, k-2\}$. A repeat at $(i',j') = (i+r, j+r)$ implies that $a_{i+r+l} = a_{j+r+l}$ for all $l < k-1$, and in particular, for $l = k-1-r$ we get $a_{i+k-1} = a_{j+k-1}$. But a rightmost repeat at (i,j) implies that $a_{i+k-1} \neq a_{j+k-1}$. Thus, $((i,j),(i',j'))$ can not be an interleaved Rr-pair, so $P_{i,j,i',j'} = 0$.

Let $G'_{i,i',j,j'}$ be the graph obtained from $G_{i,i',j,j'}$ by adding the edges $e_1 = (i+k-1, j+k-1)$ and $e_2 = (i'+k-1, j'+k-1)$ (corresponding to the two inequalities). For case 2, assume that $G'_{i,i',j,j'}$ has no cycles, and therefore $G_{i,i',j,j'}$ has no cycles, so $m_l = n_l - 1$ for every l. Therefore, the probability that (i,j) and (i',j') are repeats is $\prod_{l=1}^{b} 1/4^{m_l} = 1/4^{\sum_{l=1}^{b} m_l} = 1/4^{2(k-1)}$ where the last equality follows from the fact that $G_{i,i',j,j'}$ has $2(k-1)$ edges. Furthermore, as the edges e_1 and e_2 do not create a cycle, it follows that $P_{i,j,i',j'} = (1/4)^{2(k-1)}(3/4)^2 = 9/4^{2k}$.

Now, for case 3, suppose that $G'_{i,i',j,j'}$ contains a cycle. Note that a cycle in $G'_{i,i',j,j'}$ cannot pass through e_2 as the vertex $j'+k-1$ has only one neighbor (the vertex $i'+k-1$). We claim that $G'_{i,i',j,j'}$ contains a cycle that passes through e_1. Let $C = [v_1, v_2, \ldots, v_{r-1}, v_r = v_1]$ be some cycle in $G'_{i,i',j,j'}$. If C passes through e_1 we are done. Otherwise, for any edge $e = (v_l, v_{l+1})$ in C, as $e \neq e_1, e_2$, then $(v_l+1, v_{l+1}+1)$ is also an edge in $G'_{i,i',j,j'}$. Therefore, $C' = [v_1+1, v_2+1, \ldots, v_r+1]$ is also a cycle in $G'_{i,i',j,j'}$. We repeat this process until we obtain a cycle that passes

through e_1 (and doesn't pass through e_2). Therefore the vertices $i+k-1$ and $j+k-1$ are in the same connected component of $G_{i,i',j,j'}$ which implies that $a_{i+k-1} = a_{j+k-1}$ and thus $((i,j),(i',j'))$ can not be an interleaved Rr-pair. Thus, $P_{i,j,i',j'} = 0$. ■

Note that a result similar to Lemma 5.3.3 was given in [12]. Arratia et al. proved the bound on $P_{i,j,i',j'}$ provided that $\max(j'-j, j-i', i'-i) \geq k$, and used computer computations to bound the other cases. Our proof of the first two cases is similar to theirs, while the third case is new.

Corollary 5.3.4. $P(n,k,d) \leq (\frac{3}{8}d^3 + \frac{5}{4}d^2) \cdot n/4^{2k} + 1/4^{(k-1)n/d}$.

Proof. There are $\frac{n}{d}\binom{d}{4} + \frac{n}{d}\binom{d}{3} = \frac{n}{d}\binom{d+1}{4}$ ways to choose the indices i, i', j, j' with $j' < dI(i) + 1$ (the first term is the number of ways with $i < i'$, and the second term is the number of ways with $i = i'$), and $\frac{n}{d}\binom{d}{3} + \frac{n}{d}\binom{d}{2} = \frac{n}{d}\binom{d+1}{3}$ ways to choose them with $j' = dI(i) + 1$. By Lemma 5.3.3, the probability that there is an interleaved Rr-pair is at most

$$\frac{n}{d}\binom{d+1}{4}\frac{9}{4^{2k}} + \frac{n}{d}\binom{d+1}{3}\frac{12}{4^{2k}} \leq \frac{9}{4!}(d^3 - 2d^2)\frac{n}{4^{2k}} + \frac{12}{3!}d^2\frac{n}{4^{2k}} \leq (\frac{3}{8}d^3 + \frac{5}{4}d^2)\frac{n}{4^{2k}}.$$

The term $1/4^{(k-1)n/d}$ bounds the probability of case (2) in Theorem 5.3.2. ■

We now give a lower bound on $P(n,k,d)$. Denote $D = \lfloor \frac{d}{4} \rfloor$.

Lemma 5.3.5. *If $d \geq 4k$ then $P(n,k,d) \geq L(n,k,d)(1 - L(n,k,d)/2)$ where*

$$L(n,k,d) = \frac{n}{d}(D-k+1)^4 \frac{9}{4^{2k}} \left(1 - (D-k+1)^2 \frac{3}{4^k}\right)^2.$$

Proof. The proof is based on looking at a large number of Rr-pair events, and estimating their contribution to $P(n,k,d)$. The dependency between these events is controlled by choosing the indices of the Rr-pairs such that the corresponding k-tuples are from different sections of the sequence.

For $r = 0, \ldots, n/d-1$, let $I_{r,1} = [rd+1, rd+D-k+1] \times [rd+2D+1, rd+3D-k+1]$ and $I_{r,2} = [rd+D+1, rd+2D-k+1] \times [rd+3D+1, rd+4D-k+1]$. Let X_r denote the event that there is an interleaved Rr-pair $((i,j),(i',j'))$ for some indices i, i', j, j' with $(i,j) \in I_{r,1}$ and $(i',j') \in I_{r,2}$. By Theorem 5.3.1, $P(n,k,d) \geq \mathrm{P}\left[\bigvee_{r=0}^{n/d-1} X_r\right]$. Clearly, the events $X_0, \ldots, X_{n/d-1}$ are independent and have equal probabilities, so $\mathrm{P}\left[\bigvee_{r=0}^{n/d-1} X_r\right] = 1 - (1 - \mathrm{P}[X_1])^{n/d}$.

We will now bound $P[X_1]$. Let Z be the event that there is $(i,j) \in I_{1,1}$ such that (i,j) is a rightmost repeat, and let Z' be the event that there is $(i',j') \in I_{1,2}$ such that (i',j') is a rightmost repeat (and in particular weakly rightmost repeat). The events Z and Z' are independent and have equal probabilities.

Thus, $\mathrm{P}[X_1] = \mathrm{P}[Z \wedge Z'] = \mathrm{P}[Z]^2$. For a pair $\alpha = (i,j) \in I_{1,1}$, we denote by Z_α the event that (i,j) is rightmost repeat, and let $Y_\alpha = Z_\alpha \wedge \bigwedge_{\beta \in I_{1,1}-\{\alpha\}} \overline{Z_\beta}$. The events $\{Y_\alpha\}_{\alpha \in I_{1,1}}$ are disjoint, so $\mathrm{P}[Z] \geq \mathrm{P}\left[\bigvee_{\alpha \in I_{1,1}} Y_\alpha\right] = \sum_{\alpha \in I_{1,1}} \mathrm{P}[Y_\alpha]$, and

$$\begin{aligned}\mathrm{P}[Y_\alpha] &= \mathrm{P}\left[Z_\alpha \wedge \bigwedge_{\beta \in I_{1,1}-\{\alpha\}} \overline{Z_\beta}\right] = \mathrm{P}[Z_\alpha]\,\mathrm{P}\left[\bigwedge_{\beta \in I_{1,1}-\{\alpha\}} \overline{Z_\beta} \,\middle|\, Z_\alpha\right] \\ &= \mathrm{P}[Z_\alpha]\left(1 - \mathrm{P}\left[\bigvee_{\beta \in I_{1,1}-\{\alpha\}} Z_\beta \,\middle|\, Z_\alpha\right]\right) \geq \mathrm{P}[Z_\alpha]\left(1 - \sum_{\beta \in I_{1,1}-\{\alpha\}} \mathrm{P}[Z_\beta | Z_\alpha]\right).\end{aligned}$$

For any $\alpha = (i,j)$, as $j - i \geq D + k > k$, we have from [12, p. 437] that $\mathrm{P}[Z_\beta | Z_\alpha] \in \{0, 3/4^k\}$ for all β. Thus, $P[Y_\alpha] \geq (3/4^k)\cdot(1-(D-k+1)^2\cdot 3/4^k)$ for all α. Therefore, $\mathrm{P}[X_1] \geq \frac{d}{n} L(n,k,d)$. Using the inequality $(1-x)^r \leq 1 - rx + \frac{1}{2}r^2x^2$ (which can be proved by induction on r) we have

$$\begin{aligned}P(n,k,d) &\geq 1 - (1 - \mathrm{P}[X_1])^{n/d} \geq 1 - (1 - \frac{d}{n}L(n,k,d))^{n/d} \\ &\geq L(n,k,d)(1 - L(n,k,d)/2).\end{aligned}$$

■

Corollary 5.3.6. *If* $5k - 5/4 \leq d \leq 2^{k+1}/c^{1/4} + 4(k-1)$ *then* $P(n,k,d) = \Omega(d^3 n/4^{2k})$.

Proof. The bounds on d imply that $D-k+1 \geq (d/4-3/4)-(d/5+1/4)+1 = d/20$ and $(D-k+1)^2 \cdot 3/4^k \leq (d/4-k+1)^2 \cdot 3/4^k \leq (2^{k-1}/c^{1/4})^2 \cdot 3/4^k = 3/4\sqrt{c}$. Thus, $L(n,k,d) \leq c((D-k+1)^2 \cdot 3/4^k)^2 \leq c(3/4\sqrt{c})^2 = 9/16$. By Lemma 5.3.5, $P(n,k,d) \geq \frac{23}{32} L(n,k,d) \geq \frac{23}{32}\frac{n}{d}(d/20)^4 \cdot (9/4^{2k}) \cdot (1 - 3/4\sqrt{c})^2 = \Omega(d^3 n/4^{2k})$. ■

Our results are valid for the classical SBH model with no subintervals, by taking $d = n$. The resulting bounds on $P(n,k)$ improve over Arratia et al. for small (realistic) values of k. See Table 5.1 for a comparison between the results.

5.4 Estimating the failure probability in the presence of errors

Let X denote the event that a random sequence is uniquely recoverable w.r.t. k, I_d but is not uniquely recoverable w.r.t. k, I_d, Δ. We call this event *failure due to noise*. In this section we show that $\mathrm{P}[X] = O(p/(1-p)^4 \cdot d^2 n/4^{2k})$, and therefore $P(n,k,d,p) = P(n,k,d) + \mathrm{P}[X] = (1 + O(p/(1-p)^4 \cdot 1/d))P(n,k,d)$.

We denote by $M_{j,i}$ the event that $i \in S_{I_d,\Delta}(j)$, namely, according to the noisy data, the j-th k-tuple can appear at position i in the solution. Note that a permutation π is a solution of A, k, I_d, Δ iff event $M_{\pi(i),i}$ happens for every $i \leq n$,

		Arratia et al.		This paper		
n	k	lower	upper	lower	upper	Simulation
193	8	0	0.5923	0.0051	0.1233	0.0907
791	10	0	0.2648	0.0083	0.1341	0.0996
3175	12	0.0502	0.1500	0.0094	0.1356	0.1009
12195	14	0.0742	0.1000	0.0084	0.1152	0.0875

Table 5.1: Comparison between the bounds in this paper and in Arratia et al. [12] for $P(n,k)$, and simulation estimates of $P(n,k)$. For the setup of the simulation see Section 5.5.

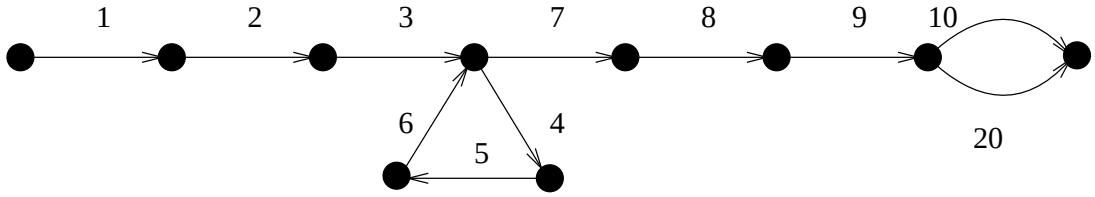

Figure 5.3: A portion of a de-Bruijn graph showing a example of a bad R-pair (for clarity, not all edges are drawn). The numbers on the edges correspond to the trivial solution $1,\ldots,n$. Let π be a solution with $\pi(1),\ldots,\pi(7) = 1,2,3,7,8,9,20$. $((4,7),(10,20))$ is an ordered R-pair. Since $\pi(4) = 7$, it follows that event $M_{7,4}$ happens, and similarly events $M_{8,5}$, $M_{9,6}$, and $M_{20,7}$ happen. Thus, $((4,7),(10,20))$ is a bad pair.

and $A|^k_{\pi(i)} \mid A|^k_{\pi(i+1)}$ for every $i \leq n-1$. As event $M_{j,i}$ happens iff $I(j)-\Delta_j \leq I(i)$, and since the random variable Δ_j has geometric distribution, it follows that $\mathrm{P}[M_{j,i}] = p^{I(j)-I(i)}$ for $i \leq j$ and $\mathrm{P}[M_{j,i}] = 1$ for $i > j$.

A pair of repeats $((i,j),(i',j'))$ is called *ordered* if $i < j < j'$, $i' \notin [j,j']$, and $I(i') \geq I(i)$ (note that i' can be either bigger or smaller than j'). Similarly to Lemma 5.3.3, the probability that $((i,j),(i'j'))$ is an ordered R-pair is either 0 or $12/4^{2k}$. An ordered R-pair is called *bad* if event $M_{i',j'-j+i}$ happens, event $M_{l,l-j+i}$ happens for every $j \leq l \leq j'-1$, and either (1) $I(i) < I(j'-1)$, or (2) $j'-1 = dI(i)$ and $j' < i'$. The role of a bad pair is similar to the role of an interleaved Rr-pair in Section 5.3: We shall show that if event X happens then there is a bad pair (an example is given in Figure 5.3), so an upper bound on the probability that there is bad pair gives us an upper bound on $\mathrm{P}[X]$. A bad pair that satisfies condition (1) is called *of type 1,* and otherwise it is called *of type 2.* As events $M_{a,a'}$ and $M_{b,b'}$ are independent if $a \neq b$, we get that the probability that $((i,j),(i',j'))$ is a bad pair is either 0 or

$$\frac{12}{4^{2k}}\mathrm{P}\left[M_{i',j'-j+i} \wedge \bigwedge_{l=j}^{j'-1} M_{l,l-j+i}\right] = \frac{12}{4^{2k}}\mathrm{P}\left[M_{i',j'-j+i}\right]\prod_{l=j}^{j'-1}\mathrm{P}\left[M_{l,l-j+i}\right] = \frac{12}{4^{2k}}p^{Q_{i,j,i',j'}}$$

where $Q_{i,j,i',j'} = \max(I(i') - I(j' - j + i), 0) + \sum_{l=j}^{j'-1} I(l) - I(l - j + i)$.

For brevity, we will use the term solution when referring to a solution of A, k, I_d, Δ. We say that a sequence A is *cyclic* if $A|_1^{k-1} = A|_{n+1}^{k-1}$. Let π be a solution, and define $\pi(0) = 0$. An index l is called a *jump point of* π if $\pi(l) \neq \pi(l-1) + 1$. If $\pi(l) > \pi(l-1) + 1$ then the index l is called a *forward jump* and if $\pi(l) < \pi(l-1) + 1$ then it is called a *backward jump*. Clearly, any nontrivial solution contains at least one forward jump and at least one backward jump. For a nontrivial solution π, we denote by i_1^π the minimum forward jump in π, and by i_2^π the minimum backward jump (clearly, $i_1^\pi < i_2^\pi$).

Claim 5.4.1. *If A is acyclic and i is a jump point in a solution π, then $(\pi(i-1)+1, \pi(i))$ is a repeat.*

Proof. If $i = 1$, then by the proof of Theorem 5.3.1, we have that $A^\pi|_1^{k-1} = A|_1^{k-1}$ and therefore $A|_{\pi(i-1)+1}^{k-1} = A|_{\pi(i)}^{k-1}$ (note that $\pi(i-1)+1 = 1$). Otherwise, since π is a solution, we have $A|_{\pi(i-1)}^k \mid A|_{\pi(i)}^k$ and therefore $A|_{\pi(i-1)+1}^{k-1} = A|_{\pi(i)}^{k-1}$. ■

We define a complete ordering on the nontrivial solutions as follows: For two nontrivial solutions π and π', $\pi > \pi'$ if the series of jump points of π, sorted in increasing order, is lexicographically larger than the corresponding series of π'. The maximum nontrivial solution has the following property, which we will use later:

Lemma 5.4.2. *Let π be the maximum nontrivial solution. If A is acyclic, $i < i_2^\pi$ is a forward jump, and $I(\pi(i-1)+1) = I(i_1^\pi)$ then $(\pi(i-1)+1, \pi(i))$ is a rightmost repeat.*

Proof. Denote $j = \pi^{-1}(\pi(i-1)+1)$. As i_2^π is the first backward jump, it follows that $j \geq i_2^\pi$ and therefore $j > i$. By Claim 5.4.1, $(\pi(i-1)+1, \pi(i))$ is a repeat. By contradiction, suppose that it is not a rightmost repeat, so $A|_{\pi(i-1)+1}^k = A|_{\pi(i)}^k$, namely $A|_{\pi(j)}^k = A|_{\pi(i)}^k$.

Define a permutation π' as follows: $\pi'(i) = \pi(j)$, $\pi'(j) = \pi(i)$, and $\pi'(l) = \pi(l)$ for any $l \neq i, j$. Since π is a solution and $A|_{\pi(i)}^k = A|_{\pi(j)}^k$, it follows that $A|_{\pi'(i)}^k \mid A|_{\pi'(i+1)}^k$ for all $i \leq n-1$. For any $l \neq i, j$, event $M_{\pi'(l),l}$ happens as event $M_{\pi(l),l}$ happens. Furthermore, $I(\pi'(j)) - \Delta_{\pi'(j)} = I(\pi(i)) - \Delta_{\pi(i)} \leq I(i) \leq I(j)$ and $I(\pi'(i)) - \Delta_{\pi'(i)} \leq I(\pi'(i)) = I(\pi(j)) = I(\pi(i-1)+1) = I(i_1^\pi) \leq I(i)$ where the last inequality follows from the fact that $i_1^\pi \leq i$. Therefore, π' is a solution, and it is nontrivial as $A^{\pi'} = A^\pi$.

Now, i is a jump point in π, but not in π' (as $\pi'(i) = \pi(j) = \pi(i-1)+1 = \pi'(i-1)+1$). The only possible jump points which exists in π' but not in π are $i+1, j$, and $j+1$, all of which are greater than i. It follows that $\pi' > \pi$, contradicting the maximality of π. Therefore, $(\pi(i-1)+1, \pi(i))$ is a rightmost repeat. ■

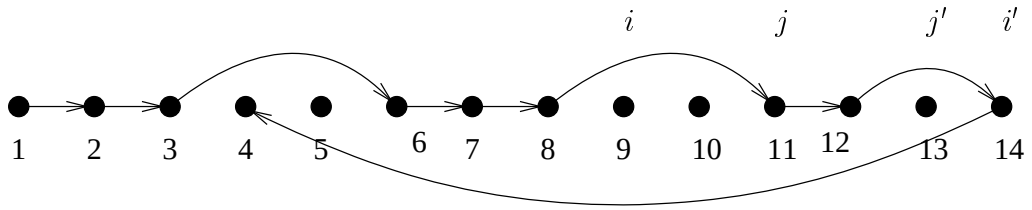

Figure 5.4: An illustration for the proof of Theorem 5.4.3. The vertices in the graph correspond to the k-tuples in the sequence numbered according to the trivial solution, and the edges correspond to the solution π, namely there is an edge $(\pi(l), \pi(l+1))$ for every $l \leq n-1$ (note that the graph is not the de-Bruijn graph). For clarity, only a portion of the graph is drawn. In this example, we have $\pi(1), \ldots, \pi(10) = 1, 2, 3, 6, 7, 8, 11, 12, 14, 4$, so $i_1^\pi = j_1 = 4$, $j_2 = 7$, $j_3 = 9$, and $i_2^\pi = j_4 = 10$. Assuming that $d = 10$, we have that $b = 2$ (as $I(\pi(j_2 - 1) + 1) = I(9) = I(4) = 1$ and $I(\pi(j_3 - 1) + 1) = I(13) = 2$). Therefore, $i = 9$, $j = 11$, $j' = 13$ and $i' = 14$.

Theorem 5.4.3. *If failure happens due to noise then either A is cyclic or there is a bad pair.*

Proof. Suppose that event X happens, namely A is uniquely recoverable w.r.t. k, I_d but A is not uniquely recoverable w.r.t. k, I_d, Δ. For the rest of the proof, assume that A is acyclic. Let π be the maximum nontrivial solution of A, k, I_d, Δ. We denote by $i_1^\pi = j_1 < j_2 < \cdots$ all the jump points of π, and let $j_0 = 1$. Let b be the maximum index for which j_b satisfies conditions of Lemma 5.4.2, namely $j_b^\pi < i_2^\pi$ and $I(\pi(j_b - 1) + 1) = I(j_1)$ (such an index exists as $\pi(j_1 - 1) + 1 = j_1$, so $I(\pi(j_1 - 1) + 1) = I(j_1)$). Since $j_1, \ldots, j_b$ are forward jumps, for any $0 \leq l \leq b$ we have that $\pi(j_l), \ldots, \pi(j_{l+1} - 1) = j_l + c_l, \ldots, j_{l+1} - 1 + c_l$, where $c_0 < c_1 < \cdots < c_b$. See Figure 5.4 for an example.

Denote $i = \pi(j_b - 1) + 1 = j_b + c_{b-1}$, $j = \pi(j_b) = j_b + c_b$, $j' = \pi(j_{b+1} - 1) + 1 = j_{b+1} + c_b$ and $i' = \pi(j_{b+1})$. We will show that $((i, j), (i', j'))$ is a bad pair. Clearly $i < j < j'$. Furthermore, i' do not belong to any interval of the form $[j_l + c_l, j_{l+1} - 1 + c_l]$, and in particular, to $[1, j_1 - 1]$ or $[j, j' - 1]$. Moreover, $i' \neq j'$ as j_{b+1} is a jump point, hence $i' \notin [1, j_1 - 1] \cup [j, j']$. Since $i \geq j_1$, it follows that $I(i') \geq I(j_1) = I(i)$. From the definition of j_b and Lemma 5.4.2, we have that (i, j) is a rightmost repeat. Furthermore, by Claim 5.4.1, (i', j') is a repeat. Therefore $((i, j), (i', j'))$ is an ordered R-pair. As π is a solution, event $M_{l,\pi^{-1}(l)}$ happens for every l. As $j_b \leq i$, for any $l \in [j, j' - 1]$ we have $\pi^{-1}(l) = l - c_b = l - j + j_b \leq l - j + i$, and since event $M_{l,\pi^{-1}(l)}$ happens, it follows that event $M_{l,l-j+i}$ happens. Similarly, $\pi^{-1}(i') = j_{b+1} = j' - j + j_b \leq j' - j + i$, hence event $M_{i',j'-j+i}$ happens.

To establish that $((i, j), (i', j'))$ is a bad pair, it remains to show that either case (1) or case (2) in the definition of a bad pair happens. We shall show that if (1) does not occur then (2) holds. For the rest of the proof assume that

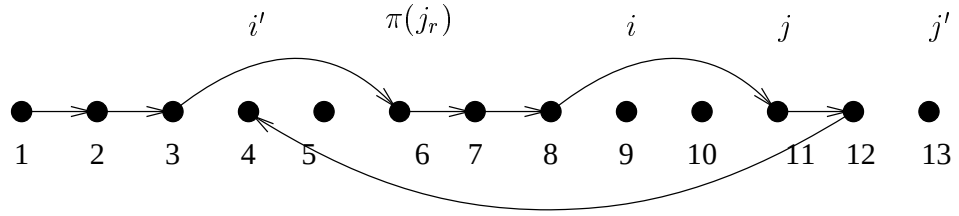

Figure 5.5: An example showing the case when $I(i) = I(j'-1)$ and j_{b+1} is a backward jump. Here $\pi(1), \ldots, \pi(9) = 1, 2, 3, 6, 7, 8, 11, 12, 4$, so $i_1^\pi = j_1 = 4$, $j_2 = 7$, and $i_2^\pi = j_3 = 9$. Assuming that $d = 20$, we have that $b = 2$, $r = 1$, $i = 9$, $j = 11$, $j' = 13$, $i' = \pi(j_r - 1) + 1 = 4$, and $\pi(j_r) = 6$. $((4, 6), (4, 13))$ is an R-pair.

$I(i) = I(j'-1)$ $(= I(j_1))$. We claim that j_{b+1} is a forward jump.

Suppose conversely that j_{b+1} is a backward jump. We have that $i' = \pi(j_{b+1}) \in [j_r + c_{r-1}, j_r + c_r - 1]$ for some index r. See Figure 5.5 for an example. Clearly, $\pi(j_r - 1) + 1 \leq i' < \pi(j_r) < j'$ and $I(\pi(j_r - 1) + 1) = I(j' - 1)$ (We have $j_1 \leq \pi(j_r - 1) + 1 \leq \pi(j_b - 1) + 1$. Therefore, $I(j_1) \leq I(\pi(j_r - 1) + 1) \leq I(\pi(j_b - 1) + 1) = I(j_1)$ so $I(\pi(j_r - 1) + 1) = I(j_1) = I(j' - 1)$). By Lemma 5.4.2, $(\pi(j_r - 1) + 1, \pi(j_r))$ is a rightmost repeat. Hence, $((\pi(j_r - 1) + 1, \pi(j_r)), (i', j'))$ is an interleaved R-pair, and by Theorem 5.3.1, A is not uniquely recoverable w.r.t. k, I_d, a contradiction.

We conclude that j_{b+1} is a forward jump, so $j' < i'$. Furthermore, by the maximality of j_b it follows that $I(j') > I(j_1) = I(i)$. Since $I(j' - 1) = I(i)$, we conclude that $j' - 1 = dI(i)$. Thus, $((i, j), (i', j'))$ is a bad pair of type 2. ■

By Theorem 5.4.3, $\mathrm{P}[X]$ is less than or equal to the probability that A is cyclic plus the probability there is a bad pair. The former probability is exactly $1/4^{k-1}$. Let P_r denote the probability that there is a bad pair $((i, j), (i', j'))$ with $I(i) = r$. It is easy to verify that $P_1 \geq P_2 \geq \cdots \geq P_{n/d-1}$ and $P_{n/d} = 0$, so the probability that there is a bad pair is at most $(n/d - 1)P_1$. We now bound P_1. We consider five cases, where in the first four cases we consider bad pairs of type 1, and in the fifth case we consider bad pairs of type 2.

Case 1: $I(j) < I(j'-1)$ and $j' < i'$. Denote $q = \lceil (j-i)/d \rceil - 1$, $x = (j-i) - qd$, $r = I(j' - 1) - I(j) - 1$, $y = (j' - 1) - d(I(j' - 1) - 1)$, and $s = \lceil (i' - j')/d \rceil - 1$. Note that $q, r, s \geq 0$ and $1 \leq x, y \leq d$.

Claim 5.4.4. *In case 1,* $Q_{i,j,i',j'} \geq s + r + q + \min(x, y)$.

Proof. Recall that $Q_{i,j,i',j'} = I(i') - I(j' - j + i) + \sum_{l=j}^{j'-1} I(l) - I(l - j + i)$. If $r > 0$, then for $t = 0, \ldots, r-1$, let $l_t = d(I(j) + t) + 1$. Let $L_1 = \{j\}$, $L_2 = \{l_0, \ldots, l_{r-1}\}$, and $L_3 = [d(I(j' - 1) - 1) + 1, j' - 1]$. Denote $Q_t = \sum_{l \in L_t} I(l) - I(l - j + i)$, and clearly $Q_{i,j,i',j'} \geq I(i') - I(j' - j + i) + Q_1 + Q_2 + Q_3$. The claim is proven by observing the following:

1. As $I(l) = \lceil l/d \rceil$, we get that $I(i') \geq I(j') + I(i' - j') - 1 = I(j') + s \geq I(j' - j + i) + s$. Hence $I(i') - I(j' - j + i) \geq s$.

2. $I(j) \geq I(i) + I(j - i) - 1 = I(i) + q$, so $Q_1 = I(j) - I(i) \geq q$.

3. Any index $l_t \in L_2$ satisfies $I(l_t) - I(l_t - j + i) \geq 1$ (as $I(l_t) = I(j) + t + 1$ and $I(l_t - j + i) \leq I(l_t - 1) = I(j) + t$). Therefore, $Q_2 \geq r$.

4. For any $l \in L_3$, if $l - d(I(j' - 1) - 1) \leq x$ then $I(l) - I(l - j + i) \geq 1$. Thus, $Q_3 \geq \min(x, y)$. (Note that if $q > 1$ then $Q_3 \geq y$.) ■

We note that the bound in Claim 5.4.4 is very crude, but it is suffice for our needs. For fixed values of q, x, r, y, and s, there are at most d ways to choose a value for i (as $I(i) = 1$), and at most d ways to choose a value for i' (the values of j and j' are fixed after choosing a value for i). Therefore, the contribution of case 1 to P_1 is at most

$$\begin{aligned}
\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{r\geq 0}\sum_{s\geq 0} d^2 \frac{12}{4^{2k}} p^{Q_{i,j,i',j'}} &\leq \frac{12}{4^{2k}} d^2 \sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{r\geq 0}\sum_{s\geq 0} p^{s+r+q+\min(x,y)} \\
&\leq \frac{12}{4^{2k}} d^2 \sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{r\geq 0} \frac{1}{1-p} p^{r+q+\min(x,y)} \\
&\leq \frac{12}{4^{2k}} d^2 \sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0} \frac{1}{(1-p)^2} p^{q+\min(x,y)} \\
&\leq \frac{12}{4^{2k}} d^2 \sum_{x=1}^{d}\sum_{y=1}^{d} \frac{1}{(1-p)^3} p^{\min(x,y)} \\
&\leq \frac{12}{4^{2k}} d^2 \frac{1}{(1-p)^3} \cdot 2 \sum_{y=1}^{d}\sum_{x=y}^{d} p^y \\
&\leq \frac{12}{4^{2k}} d^2 \frac{2}{(1-p)^3} \sum_{y=1}^{d} d p^y \\
&\leq \frac{12}{4^{2k}} \frac{2p}{(1-p)^4} d^3.
\end{aligned}$$

Case 2: $I(j) < I(j' - 1)$ and $i' < j'$ (so $i' < j$). Define q, x, r, and y as in case 1. Here $Q_{i,j,i',j'} \geq r + q + \min(x, y)$. For fixed values of q, x, r, and y, there are at most d ways to choose a value for i, and $(q + 2)d$ ways to choose a value for i' (as $I(i) \leq I(i') \leq I(j)$ and $I(j) - I(i) \leq I(j - i) = q + 1$). The contribution of

case 2 to P_1 is at most

$$\begin{aligned}\frac{12}{4^{2k}}\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{r\geq 0}(q+2)d^2p^{Q_{i,j,i',j'}} &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{r\geq 0}(q+2)p^{r+q+\min(x,y)}\\ &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\frac{1}{1-p}(q+2)p^{q+\min(x,y)}\\ &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\frac{2-p}{(1-p)^3}p^{\min(x,y)}\\ &\leq \frac{12}{4^{2k}}\frac{2p(2-p)}{(1-p)^4}d^3.\end{aligned}$$

Case 3: $I(j)=I(j'-1)$ and $j'<i'$. Let $z=j-d(I(j)-1)-1$. We have that $j-i>z$ because otherwise, $I(i)=I(j)=I(j'-1)$ contradicting the assumption that $((i,j),(i',j'))$ is a bad pair of type 1. Denote $q=\lceil (j-i-z)/d\rceil-1$, $x=(j-i-z)-qd$, $y=j'-j$, and $s=\lceil (i'-j')/d\rceil-1$. Then, $Q_{i,j,i',j'}\geq s+q+\min(x,y)$. For fixed values of q,x,y, and s, there are at most d ways to choose a value for j, and d ways to choose a value for i'. Therefore, the contribution of case 3 to P_1 is at most

$$\begin{aligned}\frac{12}{4^{2k}}\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{s\geq 0}d^2p^{Q_{i,j,i',j'}} &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}\sum_{s\geq 0}p^{s+q+\min(x,y)}\\ &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\frac{1}{(1-p)^2}p^{\min(x,y)}\\ &\leq \frac{12}{4^{2k}}\frac{2p}{(1-p)^3}d^3.\end{aligned}$$

Case 4: $I(j)=I(j'-1)$ and $i'<j'$. With z defined as in case 3, we have again that $j-i>z$. Define q,x,r, and y as in case 3. Here $Q_{i,j,i',j'}\geq q+\min(x,y)$. For fixed values of q,x, and y, there are at most d ways to choose a value for j, and $(q+2)d$ ways to choose a value for i'. Therefore, the contribution of case 4 to P_1 is at most

$$\begin{aligned}\frac{12}{4^{2k}}\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}(q+2)d^2p^{Q_{i,j,i',j'}} &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\sum_{q\geq 0}(q+2)p^{q+\min(x,y)}\\ &\leq \frac{12}{4^{2k}}d^2\sum_{x=1}^{d}\sum_{y=1}^{d}\frac{2-p}{(1-p)^2}p^{\min(x,y)}\\ &\leq \frac{12}{4^{2k}}\frac{2p(2-p)}{(1-p)^3}d^3.\end{aligned}$$

Case 5: $j' - 1 = dI(i)$ and $j' < i'$. Denote $s = \lceil (i' - j')/d \rceil - 1$. Here $Q_{i,j,i',j'} \geq s+1$. For a fixed value of s, there are at most $\binom{d}{2}$ ways to choose values for i and j, and d ways to choose a value for i'. Therefore, the contribution of case 5 to P_1 is at most

$$\frac{12}{4^{2k}} \sum_{s \geq 0} d \cdot \binom{d}{2} \cdot p^{Q_{i,j,i',j'}} \leq \frac{12}{4^{2k}} \frac{d^3}{2} \sum_{s \geq 0} p^{s+1}$$
$$\leq \frac{12}{4^{2k}} \frac{p}{2(1-p)} d^3.$$

Combining all cases, we obtain that $P_1 = O(p/(1-p)^4 \cdot d^3/4^{2k})$. Therefore, we proved the following theorem:

Theorem 5.4.5. $\mathrm{P}[X] = O(1/4^{k-1} + p/(1-p)^4 \cdot d^2 n/4^{2k})$.

Combining Theorem 5.4.5 and Corollary 5.3.6 gives the following theorem:

Theorem 5.4.6. *If* $5k - 5/4 \leq d \leq 2^{k+1}/c^{1/4} + 4(k-1)$ *and* $P(n,k,d) = \Omega(1/p \cdot d/4^k)$ *then* $P(n,k,d,p) = (1 + O(p/(1-p)^4 \cdot 1/d))P(n,k,d)$.

5.5 Experimental results

To complement our theoretical results, we performed simulations with random and real DNA sequences. In the first set of simulations, we randomly generated a sequence, partitioned it into d-long IFs, and computed the error-prone assignment of k-mers into IFs according to our probabilistic model, assuming 50% false negative errors. A simple backtracking algorithm was then used to compute the set of all distinct sequences that are consistent with the data. The same process was performed with noiseless data. (Formally, if the target sequence is A, we computed the set B_0 of all the sequences A' such that $A' = A^\pi$ for a solution π of A, k, I_d, and the set $B_{0.5}$ of all the sequences A' such that $A' = A^\pi$ for a solution π of A, k, I_d, Δ.) The program was run 10,000 times for each combination of n, k and d. Let p_t denote the fraction of runs in which the sequence was not uniquely recoverable in case of false negative probability t ($|B_t| > 1$). Clearly, p_0 and $p_{0.5}$ are estimates of $P(n,k,d)$ and $P(n,k,d,0.5)$. The results are given in Table 5.2, and they show that indeed $p_{0.5}/p_0 = 1 + O(1/d)$ as stated in Theorem 5.4.6.

Another set of experiments tested the power of the strategy in a more realistic scenario. Here we no longer assumed that the IFs are of equal size. Instead, we randomly chose clone positions so that they are uniformly distributed across the target sequence, and the average distance between adjacent left endpoints of clones is $\bar{d}$. Here we considered both endpoints of the clones for partitioning the sequence into IFs. Note that we assume that the order of the clones and the positions of the endpoints are known. 1000 random target sequences were

n	k	d	p_0	$p_{0.5}$	$p_{0.5}/p_0$
18880	8	40	0.0985	0.1353	1.374
9550	8	50	0.1025	0.1260	1.229
5520	8	60	0.0994	0.1190	1.197
3500	8	70	0.0979	0.1114	1.138
2320	8	80	0.0956	0.1074	1.123
1620	8	90	0.0903	0.1006	1.114
1200	8	100	0.0896	0.0964	1.076
880	8	110	0.0890	0.0950	1.067
720	8	120	0.0945	0.0998	1.056

Table 5.2: An experimental estimation of $P(n, k, d)$ and $P(n, k, d, 0.5)$. p_0 and $p_{0.5}$ are the estimates of $P(n, k, d)$ and $P(n, k, d, 0.5)$, respectively. For each value of d, we chose a value for n so that p_0 will be approximately 0.1.

n	k	$\bar{d}$	P_0	$P_{0.5}$	E_0	$E_{0.5}$
5000	9	40	0.038	0.046	0.00022	0.00028
10000	9	40	0.098	0.108	0.00034	0.00038
20000	9	40	0.158	0.192	0.00028	0.00035
30000	9	40	0.228	0.277	0.00028	0.00036
40000	9	40	0.316	0.360	0.00034	0.00040

Table 5.3: Results of simulations with random sequences and uniformly distributed clone positions. P_t estimates the fraction of times the reconstructed sequence differs from the target sequence for false-negative probability t. E_t estimates the fraction of incorrectly reconstructed positions in the sequence.

generated for each value of n, and for each one, the first solution generated by the backtracking algorithm (representing an arbitrary solution) was compared with the target sequence. Two statistics were computed: The fraction of the runs in which the two sequences differed, and the average rate of mismatches between them. (Technically, the algorithm chose one sequence A_0 from B_0, and one sequence $A_{0.5}$ from $B_{0.5}$. We measured P_0, the fraction of runs in which $A_0 \neq A$, and $P_{0.5}$, the fraction of runs in which $A_{0.5} \neq A$. We also computed E_t, the average over all runs of the fraction of positions in which A_t and A differ, for $t = 0, 0.5$.) Table 5.3 contains the results. While the odds of completely correct reconstruction decrease with target size, the average number of mismatch errors in the reconstruction was very low: between 2 and 4 in 10,000 bp.

Table 5.4 shows results of the same simulation using real (coding and non-coding) Human DNA sequences. For each target length, 10 disjoint sequences were used. As expected, due to the non-randomness of real DNA, the results

n	k	$\bar{d}$	P_0	$P_{0.5}$	E_0	$E_{0.5}$
5000	9	40	0.4	0.4	0.00114	0.00138
10000	9	40	0.5	0.5	0.00090	0.00094
20000	9	40	0.8	0.8	0.00067	0.00075
30000	9	40	0.8	1.0	0.00087	0.00104

Table 5.4: Results of simulations with Human DNA sequences and uniformly distributed clone positions.

worsen. In fact, with sequences of length 30,000 bp and error-prone data, none of the reconstructions was perfectly correct. However, even in that situation, the average number of miscalled base errors was only about one in 1000 bp.

In closing, we note that further simulations making even weaker assumptions can be performed: The assumption that clone order and endpoint positions are known can be removed (This requires an algorithm that given the hybridization data, finds the clone order and endpoint positions. While such algorithms exist, e.g. [96, 108], they have yet to be adapted to the situation studied here.), and the knowledge of the noisy multi-spectrum can be replaced by noisy spectrum without multiplicities. False positives can also be incorporated. We intend to pursue the above in the future. A key limitation in our analysis and simulations, is the assumption of independence of overlapping clone spectra. Though such dependencies definitely exist in real spectra, it is currently unclear how to model them adequately.

Bibliography

[1] L. M. Adleman. Location sensitive sequencing of DNA. Technical report, University of Southern California, 1998.

[2] T. Akutsu. An RNC algorithm for finding a largest common subtree of two trees. *IEEE Trans. Information Systems*, E75-D:95–101, 1992.

[3] T. Akutsu and M. M. Halldórsson. On the approximation of largest common subtrees and largest common point sets. In *Proc. 5th Int. Symposium on Algorithms and Computation*, LNCS 834, pages 405–413. Springer-Verlag, 1994.

[4] N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. In *Proc. Ninth Symposium on Discrete Algorithms (SODA 98)*, pages 594–598. ACM press, 1998.

[5] S. Anderson. Graphical representation of molecules and substructure-search queries in MACCS. *J. of Molecular Graphics*, 2:8–90, 1984.

[6] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. *BIT*, 25:2–33, 1985.

[7] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. on Algebraic and Discrete Methods*, 8:277–284, 1987.

[8] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. Assoc. Comput. Mach.*, 40:1134–1164, 1993.

[9] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs. *J. of Algorithms*, 12:308–340, 1991.

[10] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems on graphs embedded in k-trees. *Discrete Appl. Math.*, 23:11–24, 1989.

[11] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proc. 30th Symposium on the Theory of Computing (STOC 98)*, pages 106–113, 1998.

[12] R. Arratia, D. Martin, G. Reinert, and M. S. Waterman. Poisson process approximation for sequence repeats, and sequencing by hybridization. *J. of Computational Biology*, 3(3):425–463, 1996.

[13] W. Bains and G. C. Smith. A novel method for nucleic acid sequence determination. *J. Theor. Biology*, 135:303–307, 1988.

[14] G. Ball and D. Hall. A clustering technique for summarizing the multivariate data. *Behavioral Sciences*, 12(2):153–155, 1967.

[15] A. Ben-Dor, I. Pe'er, R. Shamir, and R. Sharan. On the complexity of positional sequencing by hybridization. In *Proc. 10th Annual Symposium on Combinatorial Pattern Matching (CPM 99)*, pages 88–100, 1999.

[16] A. Ben-Dor, R. Shamir, and Z. Yakhini. Manuscript, 1999.

[17] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *J. of Computational Biology*, 6:281–297, 1999.

[18] C. Berge. *Graphs and Hypergraphs*. North-Holland, Amsterdam, 1973.

[19] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *J. of Algorithms*, 8(2):216–235, 1987.

[20] J. Błażewicz, P. Formanowicz, M. Kasprzak, W. T. Markiewicz, and J. Węglarz. DNA sequencing with positive and negative errors. *J. of Computational Biology*, 6(1):113–123, 1999.

[21] J. Błażewicz, J. Kaczmarek, M. Kasprzak, W. T. Markiewicz, and J. Węglarz. Sequential and parallel algorithms for DNA sequencing. *CABIOS*, 13:151–158, 1997.

[22] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Proc. 15th Int. Colloq. Automata, Languages and Programming*, LNCS 317, pages 105–118. Springer-Verlag, 1988.

[23] H. L. Bodlaender. Dynamic algorithms for graphs with treewidth 2. In *Proc. 19th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG 93)*, pages 112–124, 1993.

[24] H. L. Bodlaender. On reduction algorithms for graphs with small treewidth. In *Proc. 19th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG 93)*, pages 45–56, 1993.

[25] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Computing*, 25:1305–1317, 1996.

[26] H. L. Bodlaender and B. de Fluiter. Reduction algorithms for constructing solutions of graphs with small treewidth. In *Proc. 2nd Int. Conf. on Computing and Combinatorics (COCOON 96)*, LNCS 1090, pages 199–208. Springer-Verlag, 1996.

[27] H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. In *Proc. 22nd Int. Colloq. on Automata, Languages and Programming*, LNCS 944, pages 268–279. Springer-Verlag, 1995.

[28] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proc. 28th Symposium on Foundation of Computer Science (FOCS 87)*, pages 280–285, 1987.

[29] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear algorithms from predicate calculus descriptions of problems on recursively constructed graphs. *Algorithmica*, 7:555–581, 1992.

[30] S. D. Broude, T. Sano, C. S. Smith, and C. R. Cantor. Enhanced DNA sequencing by hybridization. *Proc. Nat. Acad. Sci. USA*, 91:3072–3076, 1994.

[31] T. Carson and R. Impagliazzo. Hill-climbing finds random planted bisections. In *Proc. Twelfth Symposium on Discrete Algorithms (SODA 01)*, pages 903–909. ACM press, 2001.

[32] C. Chauve. Pattern matching in static trees. Research Report RR 1254-01, LaBRI Univ. Bordeaux, 2001.

[33] J. Cheriyan. Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM J. Computing*, 26:1635–1655, 1997.

[34] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statis.*, 23:493–507, 1952.

[35] M. J. Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *J. of Algorithms*, 8:106–112, 1987.

[36] E. Cockayne, S. Goodman, and S. Hedetniemi. A linear algorithm for the domination number of a tree. *Information Processing Letters*, 4(2):41–44, 1975.

[37] R. Cole and R. Hanharan. Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time. In *Proc. 29th Symposium on the Theory of Computing (STOC 97)*, pages 66–75, 1997.

[38] R. Cole, R. Hanharan, and P. Indyk. Tree pattern matching and subset matching in deterministic $O(n \log^3 n)$-time. In *Proc. 10th Symposium on Discrete Algorithms (SODA 99)*, pages 245–254, 1999.

[39] A. E. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001.

[40] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comp.*, 9:23–52, 1990.

[41] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[42] A. Dessmark, A. Lingas, and A. Proskurowski. Faster algorithms for subgraph isomorphism of k-connected partial k-trees. *Algorithmica*, 27(3):337–347, 2000.

[43] E. A. Dinic. An algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doklady*, 11:1277–1280, 1970.

[44] K. Doi and H. Imai. Sequencing by hybridization in the presence of hybridization errors. In *Genome Informatics 2000*, pages 53–62, 2000.

[45] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In *Proc. 10th Symposium on Discrete Algorithms (SODA 99)*, pages 291–299, 1999.

[46] R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4:114–128, 1989.

[47] M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. In *Proc. 31st Symposium on Foundation of Computer Science (FOCS 91)*, pages 145–149, 1990.

[48] M. E. Dyer and A. M. Frieze. The solution of some random NP-hard problems in polynomial expected time. *J. of Algorithms*, 10(4):451–489, 1989.

[49] M. E. Dyer, A. M. Frieze, and S. Suen. Ordering clone libraries in computational biology. *J. of Computational Biology*, 2:207–218, 1995.

[50] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proc. 6th Symposium on Discrete Algorithms (SODA 95)*, pages 632–640. ACM press, 1995.

[51] S. Even. *Graph Algorithms.* Computer Science Press, Rockville, Maryland, 1979.

[52] B. Everitt. *Cluster analysis.* Edward Arnold, London, 1993.

[53] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *Proc. 23rd Symposium on the Theory of Computing (STOC 91)*, pages 123–133. ACM press, 1991.

[54] U. Feige and J. Kilian. Heuristics for semirandom graph problems. *J. of Computer and System Sciences*, 63(4):639–671, 2001.

[55] A. M. Frieze and B. V. Halldórsson. Optimal sequencing by hybridization in rounds. *J. of Computational Biology*, 9(2):355–369, 2002.

[56] J. Gallant, D. Maier, and J. A. Storer. On finding minimal length superstrings. *J. of Computer and System Sciences*, 20:50–58, 1980.

[57] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., San Francisco, 1979.

[58] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. *Theor. Comput. Sci.*, 1:237–267, 1976.

[59] P. B. Gibbons, R. M. Karp, G. L. Miller, and D. Soroker. Subtree isomorphism is in random NC. *Discrete Applied Mathematics*, 29:35–62, 1990.

[60] R. Greenlaw. Subtree isomorphism is in DLOG for nested trees. *Int. J. of Foundations of Computer Science*, 7:161–167, 1996.

[61] A. Gupta and N. Nishimura. Characterizing the complexity of subgraph isomorphism for graphs of bounded path-width. In *Proc. 17th Int. Symp. Theoretical Aspects of Computer Science (STACS 96)*, LNCS 1046, pages 453–464. Springer-Verlag, 1996.

[62] E. Halperin, S. Halperin, T. Hartman, and R. Shamir. Handling long targets and errors in sequencing by hybridization. In *Proc. 6th Annual International Conference on Computational Molecular Biology (RECOMB '02)*, to appear.

[63] S. Hannenhalli, P. A. Pevzner, H. Lewis, and S. Skiena. Positional sequencing by hybridization. *Computer Applications in the Biosciences*, 12:19–24, 1996.

[64] P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79:191–215, 1997.

[65] J. A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.

[66] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(200):175–181, 2000.

[67] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *J. Assoc. Comput. Mach.*, 29(1):68–95, 1982.

[68] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Computing*, 2:225–231, 1973.

[69] J. E. Hopcroft and R. E. Tarjan. Isomorphism of planar graphs. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 131–152. Plenum Press, 1972.

[70] O. H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. of Algorithms*, 3:45–56, 1982.

[71] M. Inaba, N. Katoh, and H. Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In *Proc. 10th Symposium on Computational Geometry*, pages 332–339, 1994.

[72] M. Jerrum and G. B. Sorkin. The Metropolis algorithm for graph bisection. *Discrete Applied Math*, 8:155–175, 1998.

[73] A. Jules. *Topics in black box optimization*. PhD thesis, U. California, 1996.

[74] M. Karpinski and A. Lingas. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27–32, 1989.

[75] S. Khanna, R. Motwani, and F. F. Yao. Approximation algorithms for the largest common subtree problem. Technical Report STAN-CS-95-1545, Stanford University, Dept. Computer Science, 1995.

[76] T. Kikuno, N. Yoshida, and Y. Kakuda. A linear algorithm for the domination number of series-parallel graphs. *Discrete Applied Math*, 37:299–311, 1983.

[77] S. Kirkpatrick, C. D. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[78] S. R. Kosaraju. Efficient tree pattern matching. In *Proc. 30th Symposium on Foundation of Computer Science (FOCS 89)*, pages 178–183, 1989.

[79] R. Laskar, J. Pfaff, S. M. Hedetniemi, and S. T. Hedetniemi. On the algorithmic complexity of total domination. *SIAM J. on Algebraic and Discrete Methods*, 5(3):420–425, 1984.

[80] A. Lingas. An application of maximum bipartite C-matching to subtree isomorphism. In *Proc. 8th Colloquium on Trees in Algebra and Programming (CAAP 83)*, LNCS 159, pages 284–299. Springer-Verlag, 1983.

[81] A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.

[82] A. Lingas and M. M. Sysło. A polynomial-time algorithm for subgraph isomorphism of two-connected series-parallel graphs. In *Proc. 15th Int. Colloq. Automata, Languages and Programming*, LNCS 317, pages 394–409. Springer-Verlag, 1988.

[83] R. J. Lipshutz. Likelihood DNA sequencing by hybridization. *J. Biomolecular Structure and Dynamics*, 11:637–653, 1993.

[84] S. Lloyd. Least squares quantization in PCM. *IEEE Trans. on Information Theory*, 28:129–137, 1982.

[85] D. Loakes and D. M. Brown. 5-Nitroindole as a universal base analogue. *Nucleic Acid Research*, 22(20):4039–4043, 1994.

[86] L. Lovasz. Covering and coloring of hypergraphs. In *Proc. 4th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*. Utilitas Mathematica Publishing, 1973.

[87] L. Lovasz and M. D. Plummer. *Matching Theory*. North-Holland, Amsterdam, 1986.

[88] F. Luccio and L. Pagli. An efficient algorithm for some tree matching problems. *Information Processing Letters*, 39:51–57, 1991.

[89] Y. Lysov, V. Floretiev, A. Khorlyn, K. Khrapko, V. Shick, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Dokl. Acad. Sci. USSR*, 303:1508–1511, 1988.

[90] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297, 1965.

[91] S. Mahajan and J. G. Peters. Algorithms for regular properties in recursive graphs. In *Proc. 25th Allerton Conf. on Communications, Control and Computing*, pages 14–23, 1987.

[92] D. Margaritis and S. Skiena. Reconstructing strings from substrings in rounds. In *Proc. 36th Symposium on Foundation of Computer Science (FOCS 95)*, pages 613–620, 1995.

[93] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Math.*, 108:343–364, 1992.

[94] D. W. Matula. An algorithm for subtree identification. *SIAM Rev.*, 10:273–274, 1968.

[95] D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.*, 2:91–106, 1978.

[96] G. Mayraz and R. Shamir. Construction of physical maps from oligonucleotide fingerprints data. *J. of Computational Biology*, 6(2):237–252, 1999.

[97] F. McSherry. Spectral partitioning of random graphs. In *Proc. 42nd Symposium on Foundation of Computer Science (FOCS '01)*, pages 529–537, 2001.

[98] B. Mirkin. *Mathematical Classiffication and Clustering*. Kluwer, 1996.

[99] A. Natanzon. Complexity and approximation of some edge modification problems. Master's thesis, Tel Aviv Univ., 1999.

[100] A. M. Odlyzko. Asymptotic enumeration methods. In R. L. Graham, M. Grotschel, and L. Lovasz, editors, *Handbook of Combinatorics*, volume 2, pages 1063–1229. Elsevier and the MIT press, 1995.

[101] R. Ostrovsky and Y. Rabani. Polynomial time approximation schemes for geometric k-clustering. In *Proc. 41st Symposium on Foundation of Computer Science (FOCS '00)*, pages 349–358, 2000.

[102] I. Pe'er and R. Shamir. Spectrum alignment: Efficient resequencing by hybridization. In *Proc. 8th International Conference on Intelligent Systems in Molecular Biology (ISMB '00)*, pages 260–268, 2000.

[103] M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. In H. Burkhardt and B. Neumann, editors, *Computer Vision—ECCV 98*, LNCS 1407, pages 3–16. Springer-Verlag, 1998.

[104] V. V. Petrov. *Sums of independent random variables*. Springer-Verlag, 1975.

[105] P. A. Pevzner. l-tuple DNA sequencing: Computer analysis. *J. Biomolecular Structure and Dynamics*, 7:63–73, 1989.

[106] P. A. Pevzner. DNA physical mapping and alternating Eulerian cycles in colored graphs. *Algorithmica*, 13:77–105, 1995.

[107] P. A. Pevzner, Yu. P. Lysov, K. R. Khrapko, A. V. Belyavsky, V. L. Florentiev, and A. D. Mirzabekov. Improved chips for sequencing by hybridization. *J. Biomolecular Structure and Dynamics*, 9:399–410, 1991.

[108] P. A. Pevzner, H. Tang, and M. S. Waterman. A new approach to fragment assembly in DNA sequencing. In *Proc. 5th Annual International Conference on Computational Molecular Biology (RECOMB '01)*, pages 256–267, 2001.

[109] V. T. Phan and S. Skiena. Dealing with errors in interactive sequencing by hybridization. *Bioinformatics*, 17(10):862–870, 2001.

[110] F. Preparata, A. Frieze, and E. Upfal. Optimal reconstruction of a sequence from its probes. *J. of Computational Biology*, 6:361–368, 1999.

[111] F. Preparata and E. Upfal. Sequencing by hybridization at the information theory bound: an optimal algorithm. In *Proc. 4th Annual International Conference on Computational Molecular Biology (RECOMB '00)*, pages 88–100, 2000.

[112] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th Symposium on the Theory of Computing (STOC 97)*, pages 475–484. ACM press, 1997.

[113] P. Scheffler. Linear-time algorithms for NP-complete problems restricted to partial k-trees. Technical Report R-Math-03/87, Karl-Weierstrass-Inst. für Mathematik, Berlin, 1987.

[114] P. Scheffler and D. Seese. A combinatorial and logical approach to linear-time computability. In *Proc. European Conference on Computer Algebra*, LNCS 378, pages 379–380. Springer-Verlag, 1989.

[115] D. Seese. Tree-partite graphs and the complexity of algorithms. Technical Report P-MATH-08/86, Akademie der Wissenschaften der DDR, Karl-Weierstrass-Inst. für Mathematik, Berlin, 1986.

[116] R. Shamir and D. Tsur. Faster subtree isomorphism. In *Proc. 5th Israel Symposium on Theory of Computing and Systems, (ISTCS 97)*, pages 126–131, 1997.

[117] R. Shamir and D. Tsur. The maximum subforest problem: Approximation and exact algorithms. In *Proc. 9th Symposium on Discrete Algorithms (SODA 98)*, pages 394–399. ACM press, 1998.

[118] R. Shamir and D. Tsur. Faster subtree isomorphism. *J. of Algorithms*, 33:267–280, 1999.

[119] R. Shamir and D. Tsur. Improved algorithms for the random cluster graph model. Workshop on Interdisciplinary Applications on Graph Theory and Algorithms, Haifa, April 2001.

[120] R. Shamir and D. Tsur. Large scale sequencing by hybridization. In *Proc. 5th Annual International Conference on Computational Molecular Biology (RECOMB '01)*, pages 267–279, 2001.

[121] R. Shamir and D. Tsur. Improved algorithms for the random cluster graph model. In *Proc. 8th Scandinavian Workshop on Algorithm Theory (SWAT '02)*, LNCS 2368, pages 230–239. Springer-Verlag, 2002.

[122] R. Shamir and D. Tsur. Large scale sequencing by hybridization. *J. of Computational Biology*, 9(2):413–428, 2002.

[123] R. Sharan and R. Shamir. Cluster graph modification problems. Manuscript, 2000.

[124] R. Sharan, R. Shamir, and D. Tsur. Cluster graph modification problems. In *Proc. 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '02)*, To appear.

[125] J. Håstad. Some optimal inapproximability results. In *Proc. 29th Symposium on the Theory of Computing (STOC 97)*, pages 1–10. ACM press, 1997.

[126] R. E. Stobaugh. Chemical substructure searching. *J. of Chemical Information and Computer Sciences*, 25:271–275, 1985.

[127] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. Assoc. Comput. Mach.*, 29:623–641, 1982.

[128] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1:146–160, 1972.

[129] D. Tsur. Tree deletion problems with bounded-diameter obstruction sets. Manuscript, 2002.

[130] P. van Beek. *Z. Wahrscheinlichkeitstheorie verw. Geb.*, 23(3):187–196, 1972.

[131] Z. Yakhini. Personal communications, 2000.